

# **Artificial Life for Graphics, Animation, Multimedia, and Virtual Reality**

SIGGRAPH 98 Course 22 Notes

## **Organizer & Lecturer**

Demetri Terzopoulos  
Department of Computer Science, University of Toronto  
& Intel Corporation

## **Lecturers**

Bruce Blumberg  
Media Laboratory, Massachusetts Institute of Technology

Przemyslaw Prusinkiewicz  
Department of Computer Science, University of Calgary

Craig Reynolds  
DreamWorks SKG

Karl Sims  
Genetic Arts

Daniel Thalmann  
Computer Graphics Lab, Swiss Federal Institute of Technology

## **Abstract**

This course investigates the increasingly important role that concepts from the field of artificial life are playing across the breadth of computer graphics, including image synthesis, modeling, animation, multimedia, and virtual reality. Attendees will be systematically introduced to techniques for realistically modeling and animating objects that are alive. They will also explore graphics techniques that emulate phenomena fundamental to biological organisms, such as biomechanics, behavior, growth, and evolution. The challenge is to develop sophisticated graphics models that are self-creating, self-evolving, self-controlling, and/or self-animating, by simulating the natural mechanisms of life.

Topics include modeling and animation of plants, animals, and humans, behavioral animation, communication and interaction with synthetic characters in virtual worlds, and artificial evolution for graphics and animation.

# Contents

Abstract . . . . .	1
Lecturer Biographies . . . . .	4
Lecturer Contact Information . . . . .	6
Course Introduction and Overview . . . . .	7
Course Schedule . . . . .	9

## Session 1: Artificial Plants — Przemyslaw Prusinkiewicz

Notes: “The Artificial Life of Plants”

*P. Prusinkiewicz, M. Hammel, R. Měch, J. Hanan*

Paper: “Visual Models of Plants Interacting with Their Environments”

*R. Měch, P. Prusinkiewicz*

Paper: “Realistic Modeling and Rendering of Plant Ecosystems”

*O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, P. Prusinkiewicz*

## Session 2: Artificial Evolution for Graphics and Animation — Karl Sims

Paper: “Artificial Evolution for Computer Graphics”

*K. Sims*

Paper: “Evolving 3D Morphology and Behavior by Competition”

*K. Sims*

## Session 3: Behavioral Animation and Evolution of Behavior — Craig Reynolds

Slides: “Building Behaviors for Animation and Interactive Multimedia”

*C. W. Reynolds*

Paper: “Flocks, Herds, and Schools: A Distributed Behavioral Model”

*C. W. Reynolds*

Paper: “Evolution of Corridor Following Behavior in a Noisy World”

*C. W. Reynolds*

Paper: “Competition, Coevolution and the Game of Tag”

*C. W. Reynolds*

## Session 4: Artificial Animals — Demetri Terzopoulos

Slides: “Artificial Animals in Realistic Virtual Worlds”

*D. Terzopoulos*

Paper: “Artificial Fishes: Autonomous Locomotion, Perception, Behavior, and Learning in a Simulated Physical World”

*D. Terzopoulos, X. Tu, R. Grzeszczuk*

Paper: “Perception and Learning in Artificial Animals”

*D. Terzopoulos, T. Rabie, R. Grzeszczuk*

Paper: “Realistic Modeling for Facial Animation”

*Y. Lee, D. Terzopoulos, K. Waters*

## **Session 5: Artificial Life of Virtual Humans — Daniel Thalmann**

Slides: “The Artificial Life of Virtual Humans”

*D. Thalmann*

Notes: “The Artificial Life of Virtual Humans”

*D. Thalmann*

Paper: “Introduction to the Artificial Life of Virtual Humans”

*N. Magnenat Thalmann, D. Thalmann*

Paper: “The Artificial Life of Synthetic Actors”

*N. Magnenat Thalmann, D. Thalmann*

Notes: “Realtime Deformable Actors”

*E. Chauvineau, S. Jianhua, D. Thalmann*

Paper: “The use of Space Discretizations for Autonomous Virtual Humans”

*S. Bandi, D. Thalmann*

Notes: “Autonomous Virtual Actors based on Virtual Sensors”

*D. Thalmann, H. Noser, Z. Huang*

Paper: “Playing Games through the Virtual Life Network”

*H. Noser, I. Pandzic, T. Capin, N. Magnenat Thalmann, D. Thalmann*

Paper: “A Model of Nonverbal Communication and Interpersonal Relationship between Virtual Actors”

*P. Becheiraz, D. Thalmann*

Paper: “Interacting with Virtual Humans through Body Actions”

*L. Emering, R. Boulic, D. Thalmann*

Paper: “A Model of Human Crowd Behavior: Group Interrelationship and Collision Detection Analysis”

*S.R. Musse, D. Thalmann*

## **Session 6: Interactive Synthetic Characters — Bruce Blumberg**

Slides: “Synthetic Characters: Behaving in Character”

*B. Blumberg*

Paper: “Multi-level Direction of Autonomous Creatures for Real-Time Virtual Environments”

*B. Blumberg, T. Galyean*

Paper: “No Bad Dogs: Ethological Lessons for Learning in Hamsterdam”

*B. Blumberg, P. Todd, P. Maes*

Paper: “Artificial Life Meets Entertainment: Lifelike Autonomous Agents”

*P. Maes*

## Lecturer Biographies

**Bruce Blumberg** is an Assistant Professor at the MIT Media Lab, where he has founded a new research group, “Synthetic Characters”. The group focuses on the problem of building interactive animated characters for use in virtual environments such as immersive story-telling systems, games, and web-based worlds. His research is on the development of an ethologically-inspired architecture for building autonomous animated creatures which live in 3D virtual worlds. Blumberg did his doctoral work at the MIT Media Lab in the Autonomous Agents group under the direction of Professor Pattie Maes and received his PhD in 1996. He is one of the chief architects of the ALIVE project at the Media Lab. Previously he worked at Apple, Inc., as product manager for the LaserWriter, and at NeXT, Inc., where he was the first employee after the founders. He has presented papers at SIGGRAPH and at AI and ALife conferences.

**Przemyslaw Prusinkiewicz** is a Professor of Computer Science at the University of Calgary. He has been conducting research in computer graphics since the late 1970s. In 1985, he originated a method for visualizing the structure and the development of plants based on L-systems, a mathematical model of development. He is a co-author of three textbooks and two monographs, *Lindenmayer Systems, Fractals and Plants* (Springer-Verlag 1989) and *The Algorithmic Beauty of Plants* (Springer-Verlag 1990), as well as approximately 50 technical papers. His current research includes the mathematical modeling and visualization of various aspects of morphogenesis. Professor Prusinkiewicz holds an M.S. and Ph.D., both in Computer Science, from the Technical University of Warsaw. Before joining the faculty of the University of Calgary, he was Professor at the University of Regina, and Assistant Professor at the University of Science and Technology of Algiers. He was also a Visiting Professor at Yale University (1988), at L’Ecole Polytechnique Fédérale de Lausanne (1990), and an invited researcher at the University of Bremen (1989) and the Centre for Tropical Pest Management in Brisbane (1993, 1994). Prusinkiewicz received SIGGRAPH’s 1997 *Computer Graphics Achievement Award* for his work on modeling and visualizing biological structures.

**Craig Reynolds** (SM ’78 MIT; SB ’75 EECS, MIT) recently joined the Feature Animation Division at DreamWorks SKG where he does R&D primarily in behavioral animation. Previously he was a Member of the Technical Staff at the Silicon Studio division at Silicon Graphics, where he designed behavioral systems for autonomous agents in animation and interactive multimedia. His project at Silicon Studio was the “Firewalker” multimedia authoring system. He has been previously affiliated with Electronic Arts (1992-94), Symbolics Graphics Division (1982-91), and Information International Inc. (“triple-I” 1979-82). He has screen credits on three feature films including *TRON* (1982) and *Batman Returns* (1992), and several animated shorts such as *Breaking the Ice* (1987) and *Ductile Flow*. He has authored research publications in the fields of computer animation and evolutionary computation. His 1987 *boids* system, a decentralized model of bird flocking, has become a landmark of behavioral animation and Artificial Life research, and has inspired related work in robotics and theoretical biology. Reynolds is a member of ACM and SIGGRAPH.

**Karl Sims** studied Life Sciences as an undergraduate at MIT and later studied computer graphics at the MIT Media Laboratory. After developing special effects software for Whitney Demos Productions, and co-founding Hollywood based Optomystic, he collaborated with Thinking Machines Corporation for several years as an artist in residence and research scientist. He currently works as an independent in Cambridge, Massachusetts and continues to explore new techniques for creating images with computers. His works of computer animation include “Panspermia,” “Liquid Selves,” “Primordial Dance,” and “Particle Dreams.” His interactive installation “Genetic Images” was recently exhibited at the Centre Pompidou in Paris.

**Demetri Terzopoulos** is Professor of Computer Science and Electrical and Computer Engineering at the University of Toronto, where he leads the Visual Modeling Group and is an NSERC Steacie Fellow and a Killam Fellow of the Canada Council for the Arts. He also heads the Computer Graphics Animation research group at Intel Corporation in Santa Clara, CA. After earning his PhD in Computer Science (AI) from MIT in 1984, he was a research scientist at the MIT Artificial Intelligence Lab through 1985. Prior to joining the UofT in 1989, he was a program leader at Schlumberger Corporation research centers in California and Texas. His published works comprise more than 200 research articles, primarily in computer graphics and vision, and also in medical imaging, CAD, artificial intelligence, and artificial life, including 8 SIGGRAPH papers and the recent edited volumes “Animation and Simulation” (Springer ’95) and “Real-Time Computer Vision” (Cambridge Univ. Press ’94). He has given hundreds of invited talks around the world, including several distinguished lectures and keynote addresses. A former Fellow of the Canadian Institute for Advanced Research, his contributions have been recognized with awards from the IEEE, the American Association for Artificial Intelligence, the Canadian Image Processing and Pattern Recognition Society, the International Digital Media Foundation, Ars Electronica, NICOGRAFH, and the University of Toronto. He serves on the editorial boards of *Graphical Models and Image Processing*, the *Journal of Visualization and Computer Animation*, *Medical Image Analysis*, and *Videre: Journal of Computer Vision Research*. He has served on ARPA, NIH, and NSF advisory committees and is program chair of the 1998 Computer Vision and Pattern Recognition Conference (CVPR’98).

**Daniel Thalmann** is full Professor and Director of the Computer Graphics Laboratory at the Swiss Federal Institute of Technology in Lausanne, Switzerland. He is also adjunct Professor at the University of Montreal, Canada. He received his diploma in nuclear physics and Ph.D in Computer Science from the University of Geneva. He is coeditor-in-chief of the *Journal of Visualization and Computer Animation*, member of the editorial board of the *Visual Computer*, the CADDM Journal (China Engineering Society) and *Computer Graphics* (Russia). He is cochair of the EUROGRAPHICS Working Group on Computer Simulation and Animation and member of the Executive Board of the Computer Graphics Society. Daniel Thalmann was member of numerous Program Committees, Program Chair of several conferences and chair of Computer Graphics International ’93 and Pacific Graphics ’95. He has also organized 4 courses at SIGGRAPH on human animation. Daniel Thalmann’s research interests include 3D computer animation, image synthesis, virtual reality, artificial life and multimedia. He has published more than 200 papers in these areas, is coeditor of 20 books, and coauthor of several books including: *Computer Animation: Theory and Practice* and *Image Synthesis: Theory and Practice*. He is also codirector of several computer-generated films with synthetic actors shown on many TV channels all over the world.

## **Lecturer Contact Information:**

### **Bruce Blumberg**, Professor

Media Laboratory, Massachusetts Institute of Technology  
20 Ames Street, Cambridge, MA 02139  
phone: 617-253-9832, fax: 617-253-6215, email: bruce@media.mit.edu

### **Przemyslaw Prusinkiewicz**, Professor

Department of Computer Science, University of Calgary  
2500 University Dr. NW, Calgary, AL, Canada, T2N 1N4  
phone: 403-220-5494, fax: 403-284-4707, email: pwp@cpsc.ucalgary.ca

### **Craig Reynolds**, Computer Scientist

DreamWorks SKG  
100 Universal Plaza, Building 601  
Universal City, CA, 91608  
email: cwr@anim.dreamworks.com

### **Karl Sims**, Founder

Genetic Arts  
8 Clinton Street, Cambridge, MA 02139  
Cambridge, MA  
email: ksims@media.mit.edu

### **Demetri Terzopoulos**, Professor

Department of Computer Science, University of Toronto  
10 King's College Road, Toronto, ON, Canada M5S 3G4  
phone: 416-978-7777, fax: 416-978-1455, email: dt@cs.toronto.edu

### **Daniel Thalmann**, Professor

Computer Graphics Lab, Swiss Federal Institute of Technology  
CH-1015 Lausanne, Switzerland  
Phone: +41-21-693-5214, fax: +41-21-693-5328, email: thalmann@lig.di.epfl.ch

## **Course Introduction and Overview**

### **Demetri Terzopoulos**

Computer graphics modeling for animation, multimedia, and virtual reality has made significant advances in the last decade. The field has witnessed the transition from an earlier generation of purely geometric models to more elaborate physics-based models. We can now simulate and animate a variety of real-world objects with stunning realism. Where do we go from here?

Graphics researchers have begun to explore a new frontier—a world of objects of enormously greater complexity than is typically accessible through physical modeling alone—objects that are *alive*. The modeling and simulation of living systems for computer graphics resonates with the burgeoning field of scientific inquiry called *Artificial Life*. Conceptually, artificial life transcends the traditional boundaries of computer science and biological science. The natural synergy between computer graphics and artificial life can be potentially beneficial to both disciplines. As this course will demonstrate, potential is becoming fulfillment.

The goal of the course is to investigate the vital role that concepts from artificial life can play in the construction of advanced graphics models for animation, multimedia, and virtual reality. The course will demonstrate and elucidate new models that realistically emulate a broad variety of living things—both plants and animals—from lower animals all the way up the evolutionary ladder to humans. Typically, these models inhabit virtual worlds in which they are subject to physical laws. Consequently, they often make use of physics-based modeling techniques. More significantly, however, they must also simulate many of the natural processes that uniquely characterize living systems—such as birth and death, growth, natural selection, evolution, perception, locomotion, manipulation, adaptive behavior, intelligence, and learning. The challenge is to develop sophisticated graphics models that are self-creating, self-evolving, self-controlling, and/or self-animating, by simulating the natural mechanisms fundamental to life.

The course shows how artificial life techniques are being exploited in graphics, animation, multimedia, and virtual reality and will progress according to the six sessions summarized below:

1. **Artificial Plants (Przemyslaw Prusinkiewicz):** This segment of the course will show how to use formalisms inspired by biological development processes to grow highly complex and realistic graphics models of plants. We will review Lindenmayer systems, introduced as a theoretical framework for studying the development of simple multicellular organisms and subsequently applied to the study of higher plants. Geometric and stochastic plant models expressed using L-systems have been extended in a manner suitable for simulating the interaction between a developing plant and its environment, including light, nutrients, and mechanical obstacles. We will also explain how to model the response of plants to pruning, which yields realistic synthetic images of sculptured plants found in topiary gardens.

- 
2. **Artificial Evolution for Graphics and Animation (Karl Sims):** This segment will show how artificial evolution allows virtual entities to be created without requiring detailed design and assembly. We will show how to evolve complex genetic codes that describe the computational procedures for automatically growing entities useful in graphics and animation. Fortunately, graphics practitioners are not required to understand these codes. Instead, they simply specify which results are more and less desirable as the entities evolve. This is a form of digital Darwinism. We will demonstrate the artificial evolution of several types of graphical entities, including virtual plants, textures, animations, 3D sculptures, and virtual creatures.
3. **Behavioral Animation and Evolution of Behavior (Craig Reynolds):** This segment will demonstrate how complex animations can emerge with minimal effort on the part of the animator from behavioral rules governing the interaction of many autonomous agents within their virtual world. We will review a classic experiment, the flocking of “boids,” that convincingly bridged the gap between artificial life and computer animation. We will explain how this behavioral animation technique has been used to create special effects for feature films, such as the animation of flocks of bats in *Batman Returns* and herds of wildebeests in *The Lion King*. We will also explain how to automatically evolve behaviors that allow multiple animate agents to perform useful tasks such as navigation and game playing for multimedia applications.
4. **Artificial Animals (Demetri Terzopoulos):** This segment will show how to build highly realistic models of animals for use in animation and virtual reality. We will present a modeling approach in which we simulate the physics of the animal in its world, the animal’s use of physics for locomotion, and its ability to link perception to action through adaptive behavior. As a concrete example, we will explain the details of an autonomous virtual fish model. The artificial fish has (i) a 3D body with internal muscles and functional fins which locomotes in accordance with biomechanical and hydrodynamic principles, (ii) sensors, including eyes that can image the virtual environment, and (iii) a brain with motor, perception, behavior, and learning centers. We will present a general approach to teaching artificial animals to perform complex locomotion tasks. Similar zoomimetic modeling principles are applicable to humans. In particular, we will explore the highly automated construction of anatomically correct, functional models of people’s heads from scanned data for facial animation.
5. **Artificial Life of Virtual Humans (Daniel Thalmann):** This segment of the course comprises an in-depth investigation of techniques for modeling and animating the most complex living systems—human beings. In particular, we will explore the increasingly important role of perception in human modeling. Virtual humans are made aware of their virtual world by equipping them with visual, tactile, and auditory sensors. These sensors provide information to support human behavior such as visually directed locomotion, manipulation of objects, and response to sounds and utterances. We will demonstrate sensor-based navigation, game playing, walking on challenging terrain, grasping, etc. We will also explore communication between virtual humans, behavior of crowds of virtual humans, and communication between real and virtual humans. Techniques for real-time virtual humans in real scenes will also be

discussed.

6. **Interactive Synthetic Characters (Bruce Blumberg):** In the final segment of the course, we explore the design and implementation of systems that enable full-body interaction between human participants and graphical worlds inhabited by artificial life forms that people find engaging. Entertaining agents can be modeled as autonomous, behaving entities. These agents have their own goals and can sense and interpret the actions of participants and respond to them in real time. We will explore immersive, nonintrusive interaction techniques requiring no goggles, data-gloves/suits, or tethers. The general approach will be illustrated with the ALIVE (Artificial Life Interactive Video Environment) system, which many participants have experienced at SIGGRAPH exhibitions.

## Course Schedule

Time	Topic	Speaker
08:30	Introduction	
08:45	Artificial Plants	
09:45	Artificial Evolution for Graphics and Animation	Sims
10:00	Break	
10:15	Artificial Evolution, cont'd	Sims
11:00	Behavioral Animation	Reynolds
12:00	Lunch	
13:30	Artificial Animals	Terzopoulos
14:30	Artificial Life of Virtual Humans	Thalmann
15:00	Break	
15:15	Virtual Humans, cont'd	Thalmann
15:45	Interactive Synthetic Characters	Blumberg
16:45	Questions & Answers	
17:00	Adjourn	

# The Artificial Life of Plants

Przemysław Prusinkiewicz, Mark Hammel, Radomír Měch

Department of Computer Science

University of Calgary

Calgary, Alberta, Canada T2N 1N4

e-mail: pwp|hammel|mech@cpsc.ucalgary.ca

Jim Hanan

CSIRO - Cooperative Research Centre for Tropical Pest Management

Brisbane, Australia

e-mail: jim@ctpm.uq.oz.au

May 15, 1995

## Abstract

In these notes we survey applications of L-systems to the modeling of plants, with an emphasis on the results obtained since the comprehensive presentation of this area in *The Algorithmic Beauty of Plants* [61]. The new developments include:

- a better understanding of theoretical issues pertinent to graphical applications of L-systems,
- extensions to programming techniques based on L-systems, and
- an extension of the range of phenomena expressible using L-systems.

**Keywords:** L-system, fractal, plant, modeling, simulation, realistic image synthesis, emergence, artificial life.

## 1 Introduction

In 1968, Aristid Lindenmayer introduced a formalism for simulating the development of multicellular organisms, subsequently named L-systems [36]. This formalism was closely related to abstract automata and formal languages, and attracted the immediate interest of theoretical computer scientists [67]. The vigorous development of the mathematical theory of L-systems [70, 27, 66] was followed by the application of the theory to the modeling of plants [18, 17, 16, 31, 42, 56, 61]. In the present notes we survey recent results, current work, and open problems pertinent to this latter domain. In this context, we emphasize the phenomenon of *data base amplification* [71], that is the possibility of generating complex

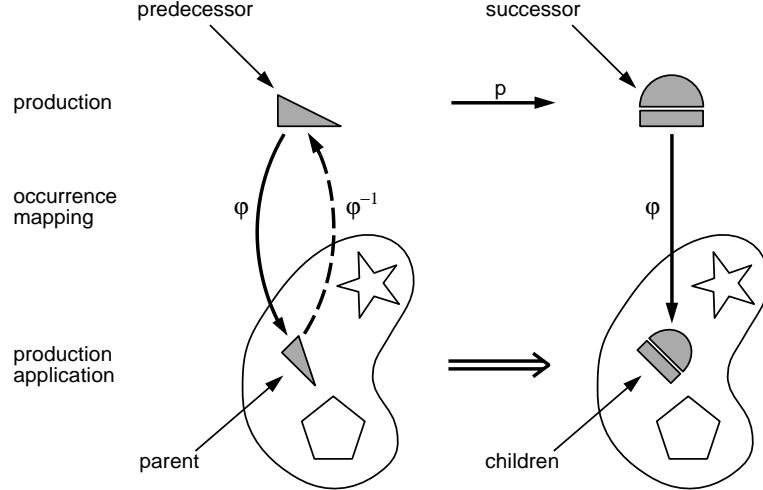


Figure 1: Illustration of the concept of rewriting applied to modules with geometric interpretation. A parent module is replaced by child modules in a sequence of transformations  $\varphi^{-1}p\varphi$ .

structures from small data sets, and the related notion of *emergence*. According to Taylor [74, page 31], emergence is a process in which a collection of interacting units acquires qualitatively new properties that cannot be reduced to a simple superposition of individual contributions. Studies of emergence are amongst the central themes of *artificial life* (*Problem 1.1*, see Section 11).

## 2 The modular structure of plants

L-systems were originally introduced to model the development of simple *multicellular* organisms (for example, algae) in terms of division, growth, and death of individual cells [36, 37]. The range of L-system applications has subsequently been extended to higher plants and complex branching structures, in particular inflorescences [18, 19], described as configurations of modules in space (*Problem 2.1*). In the context of L-systems, the term *module* denotes any discrete constructional unit that is repeated as the plant develops, for example an internode, an apex, a flower, or a branch (c.f. [4, page 284]) (*Problem 2.2*). The goal of modeling at the modular level is to describe the development of a plant as a whole, and in particular the emergence of plant shape, as the integration of the development of individual units (*Problem 2.3*).

## 3 Plant development as a rewriting process

The essence of development at the modular level can be conveniently captured by a parallel *rewriting system* that replaces individual *parent*, *mother*, or *ancestor* modules by configurations of *child*, *daughter*, or *descendant* modules. All modules belong to a finite *alphabet of module types*, thus the behavior of an arbitrarily large configuration of modules can be spec-

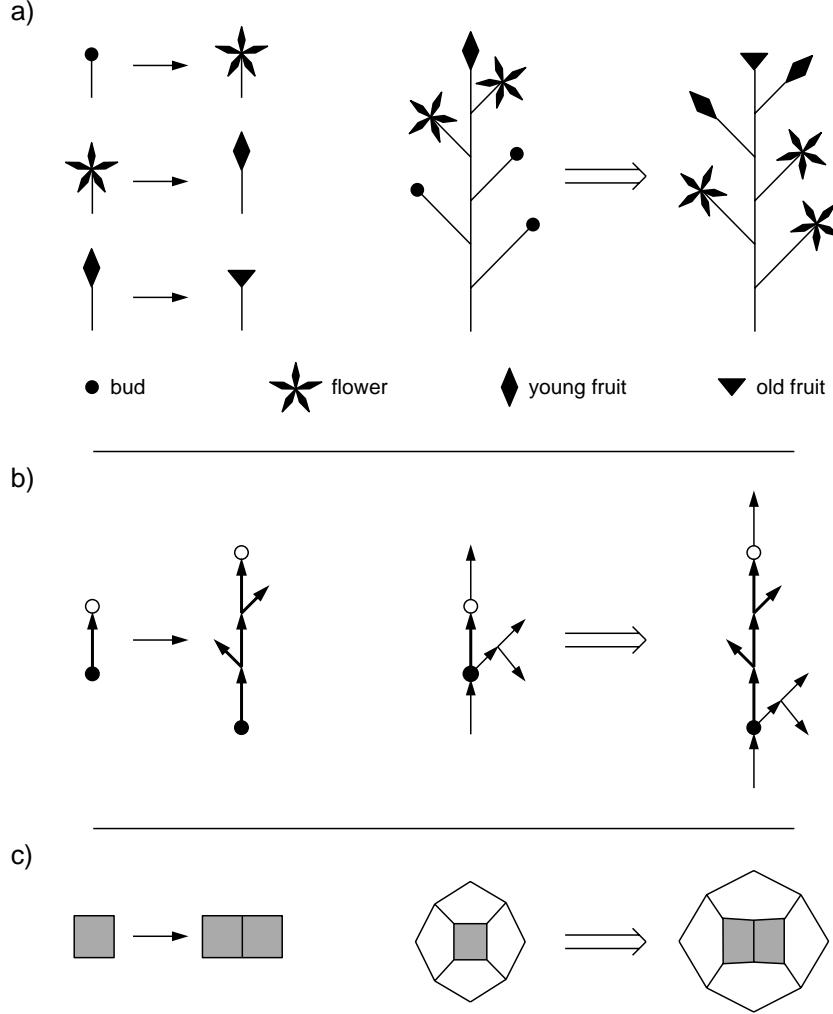


Figure 2: Examples of production specification and application: (a) development of a flower, (b) development of a branch, and (c) cell division.

ified using a finite set of *rewriting rules* or *productions*. In the simplest case of *context-free* rewriting, a production consists of a single module called the *predecessor* or the *left-hand side*, and a configuration of zero, one, or more modules called the *successor* or the *right-hand side*. A production  $p$  with the predecessor matching a given mother module can be applied by deleting this module from the rewritten structure and inserting the daughter modules specified by the production's successor. This process requires finding an *occurrence map*  $\varphi$  that transforms the predecessor into the mother module, and applying the same transformation to all the modules in the successor in order to produce the appropriate child modules (Figure 1).

Three examples of production application are shown in Figure 2. In case (a), modules located at the extremities of a branching structure are replaced without affecting the remainder of the structure. In case (b), productions that replace internodes divide the branching structure into a lower part (below the internode) and an upper part. The position of the

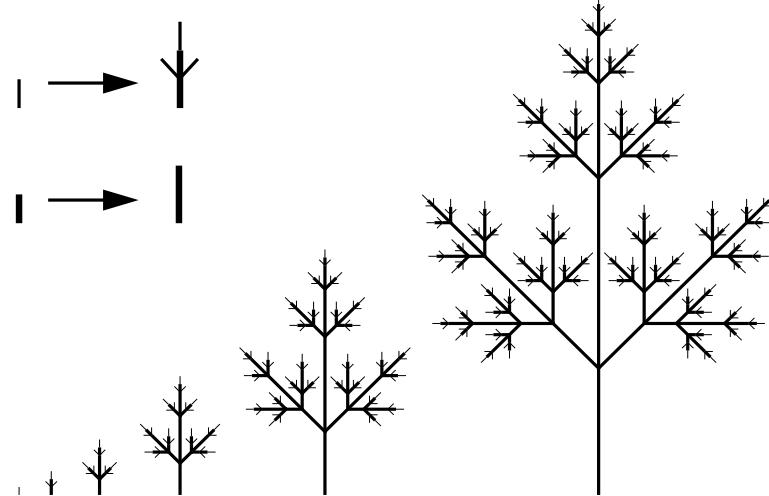


Figure 3: Developmental model of a compound leaf, modeled as a configuration of apices and internodes.

upper part is adjusted to accommodate the insertion of the successor modules, but the shape and size of both the lower and upper part are not changed. Finally, in case (c), the rewritten structures are represented by graphs with cycles. The size and shape of the production successor does not exactly match the size and shape of the predecessor, and the geometry of the predecessor and the embedding structure had to be adjusted to accommodate the successor. The last case is most complex, since the application of a local rewriting rule may lead to a global change of the structure's geometry. Developmental models of cellular layers operating in this manner have been presented in [11, 12, 15, 61]. In these notes we focus on the rewriting of branching structures corresponding to cases (a) and (b). (*Problem 3.1*).

Productions may be applied *sequentially*, to one module at a time, or they may be applied *in parallel*, with all modules being rewritten simultaneously in every *derivation step*. Parallel rewriting is more appropriate for the modeling of biological development, since development takes place simultaneously in all parts of an organism. A derivation step then corresponds to the progress of time over some interval. A sequence of structures obtained in consecutive derivation steps from a predefined *initial structure* or *axiom* is called a *developmental sequence*. It can be viewed as the result of a *discrete-time simulation* of development (Problem 3.2).

For example, Figure 3 illustrates the development of a stylized compound leaf including two module types, the *apices* (represented by thin lines) and the *internodes* (thick lines). An apex yields a structure that consists of two internodes, two lateral apices, and a replica of the main apex. An internode elongates by a constant scaling factor. In spite of the simplicity of these rules, an intricate branching structure develops from a single apex over a number of derivation steps. The fractal geometry of this structure can be viewed as an emergent property of the rewriting rules.

At first sight, Figure 3 resembles fractal generation using a Koch construction. We will see, however, that there are important differences between these two processes.

Mandelbrot [48, page 39] characterized Koch constructions as follows:

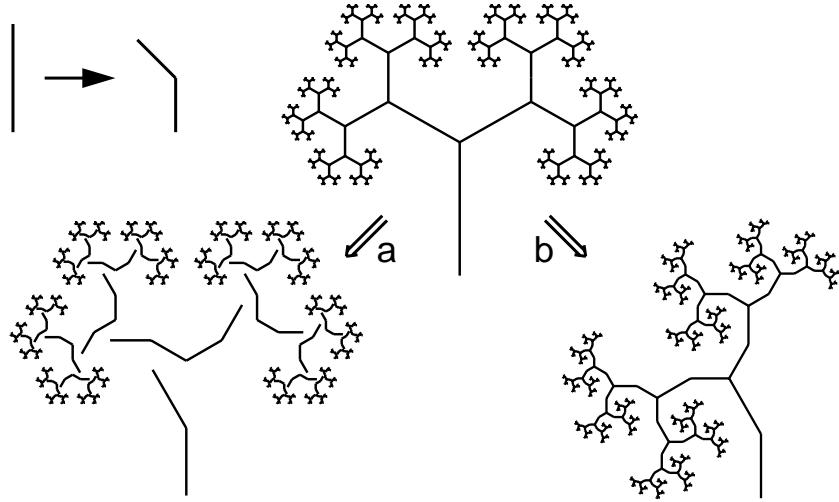


Figure 4: A comparison of the Koch construction (a) with a rewriting system preserving the branching topology of the modeled structures (b). The same production is applied in both cases, but the rules for incorporating the successor into the structure are different.

One begins with *two shapes*, an *initiator* and a *generator*. The latter is an oriented broken line made up of  $N$  equal sides of length  $r$ . Thus each stage of the construction begins with a broken line and consists in replacing each straight interval with a copy of the generator, reduced and displaced so as to have the same end points as those of the interval being replaced.

Mandelbrot introduced many extensions of this basic concept, including generators with lines of unequal length (pages 56–57) and with a branching topology (pages 71–73) (*Problem 3.3*). All these variants share one fundamental characteristic, namely that the position, orientation, and scale of the interval being replaced determine the position, orientation, and scale of the replacement (a copy of the generator). In models of plants, however, the position and orientation of each module should be determined by the chain of modules beginning at the base of the structure and extending to the module under consideration. For example, when the internodes of a plant elongate (as is the case in Figure 3), all the subtended branches are moved upwards in response. Similarly, when the internodes bend, the subtended branches are rotated and displaced (Figure 4b). A Koch construction cannot capture these phenomena, because it operates under the assumption that the parent modules determine all aspects of their descendants (Figure 4a). In contrast, in a developmental model of a branching structure the position and orientation of the descendants are determined by the subtending modules. The difference between these two cases is illustrated in Figure 5.

The information flow taking place during the development of a branching structure can be expressed directly in the geometric domain, using a proper modification of the Koch construction (*Problem 3.4*). A different approach was proposed by Lindenmayer [36, 37] and is essential to the resulting theory of L-systems. The generated structures and the rewriting rules are expressed symbolically using a string notation. The *geometric interpretation* of these strings automatically captures proper positioning of the higher branches on the lower ones.

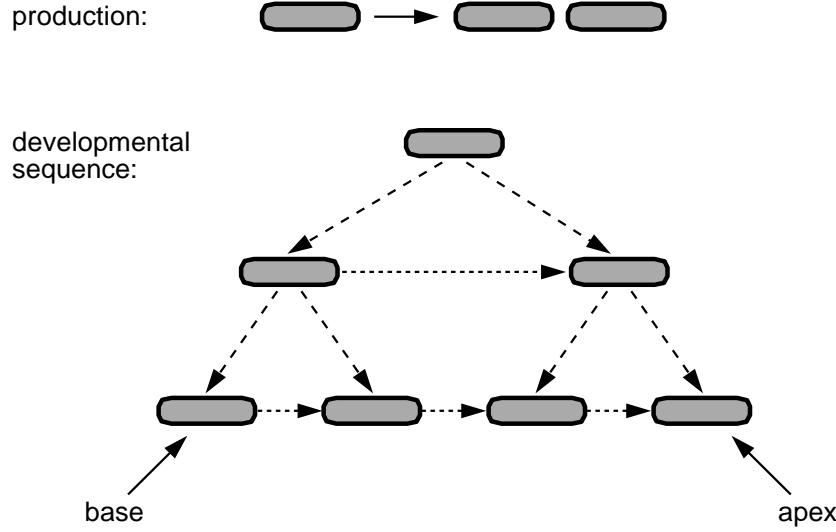


Figure 5: Information flow in a Koch construction and in a developing modular structure. In the Koch construction, information flows only from the parent modules to their descendants (continuous line). In the developmental model, positional information flows along the paths from the root to the apices of the branches (dashed line).

The basic notions of the theory of L-systems have been presented in many survey papers [39, 40, 41, 43, 44, 45] and books [27, 56, 61, 66, 70]. Consequently, we only describe parametric L-systems, which are a convenient programming tool for expressing models of plant development.

## 4 Parametric L-systems

Parametric L-systems extend the basic concept of L-systems by assigning numerical attributes to the L-system symbols. This extension was implemented as early as the 1970's in the first simulator based on L-systems, called CELIA (and acronym for CEllular Linear Iterative Array simulator) [3, 26, 27, 38], as a programming rather than theoretical construct. Our presentation closely follows the formalization introduced in [57, 61] (see also [25, 58]) (*Problems 4.1, 4.2*).

Parametric L-systems operate on *parametric words*, which are strings of *modules* consisting of *letters* with associated *parameters*. The letters belong to an *alphabet*  $V$ , and the parameters belong to the set of *real numbers*  $\Re$ . A module with letter  $A \in V$  and parameters  $a_1, a_2, \dots, a_n \in \Re$  is denoted by  $A(a_1, a_2, \dots, a_n)$ . Every module belongs to the set  $M = V \times \Re^*$ , where  $\Re^*$  is the set of all finite sequences of parameters. The set of all strings of modules and the set of all nonempty strings are denoted by  $M^* = (V \times \Re^*)^*$  and  $M^+ = (V \times \Re^*)^+$ , respectively.

The real-valued *actual* parameters appearing in the words correspond with *formal* parameters used in the specification of L-system productions. If  $\Sigma$  is a set of formal parameters, then  $C(\Sigma)$  denotes a *logical expression* with parameters from  $\Sigma$ , and  $E(\Sigma)$  is an *arithmetic expression* with parameters from the same set. Both types of expressions consist of formal

parameters and numeric constants, combined using the arithmetic operators  $+, -, *, /$ ; the exponentiation operator  $\wedge$ , the relational operators  $<, \leq, >, \geq, ==$ ; the logical operators  $!, \&\&, ||$  (not, and, or); and parentheses  $()$ . The operation symbols and the rules for constructing syntactically correct expressions are the same as in the C programming language [32]. Relational and logical expressions evaluate to zero for false and one for true. A logical statement specified as the empty string is assumed to have value one. The sets of all correctly constructed logical and arithmetic expressions with parameters from  $\Sigma$  are noted  $\mathcal{C}(\Sigma)$  and  $\mathcal{E}(\Sigma)$ .

A *parametric OL-system* is defined as an ordered quadruple  $G = \langle V, \Sigma, \omega, P \rangle$ , where

- $V$  is the *alphabet* of the system,
- $\Sigma$  is the *set of formal parameters*,
- $\omega \in (V \times \Re^*)^+$  is a nonempty parametric word called the *axiom*,
- $P \subset (V \times \Sigma^*) \times \mathcal{C}(\Sigma) \times (V \times \mathcal{E}(\Sigma))^*$  is a finite *set of productions*.

The symbols  $:$  and  $\rightarrow$  are used to separate the three components of a production: the *predecessor*, the *condition* and the *successor*. For example, a production with predecessor  $A(t)$ , condition  $t > 5$  and successor  $B(t + 1)CD(t \wedge 0.5, t - 2)$  is written as

$$A(t) : t > 5 \rightarrow B(t + 1)CD(t \wedge 0.5, t - 2). \quad (1)$$

A production *matches* a module in a parametric word if the following conditions are met:

- the letter in the module and the letter in the production predecessor are the same,
- the number of actual parameters in the module is equal to the number of formal parameters in the production predecessor, and
- the condition evaluates to *true* if the actual parameter values are substituted for the formal parameters in the production.

A matching production can be *applied* to the module, creating a string of modules specified by the production successor. The actual parameter values are substituted for the formal parameters according to their position. For example, production (1) above matches a module  $A(9)$ , since the letter  $A$  in the module is the same as in the production predecessor, there is one actual parameter in the module  $A(9)$  and one formal parameter in the predecessor  $A(t)$ , and the logical expression  $t > 5$  is true for  $t$  equal to 9. The result of the application of this production is a parametric word  $B(10)CD(3, 7)$ .

If a module  $a$  produces a parametric word  $\chi$  as the result of a production application in an L-system  $G$ , we write  $a \mapsto \chi$ . Given a parametric word  $\mu = a_1 a_2 \dots a_m$ , we say that the word  $\nu = \chi_1 \chi_2 \dots \chi_m$  is *directly derived* from (or *generated* by)  $\mu$  and write  $\mu \Rightarrow \nu$  if and only if  $a_i \mapsto \chi_i$  for all  $i = 1, 2, \dots, m$ . A parametric word  $\nu$  is generated by  $G$  in a *derivation of length n* if there exists a sequence of words  $\mu_0, \mu_1, \dots, \mu_n$  such that  $\mu_0 = \omega$ ,  $\mu_n = \nu$  and  $\mu_0 \Rightarrow \mu_1 \Rightarrow \dots \Rightarrow \mu_n$ .

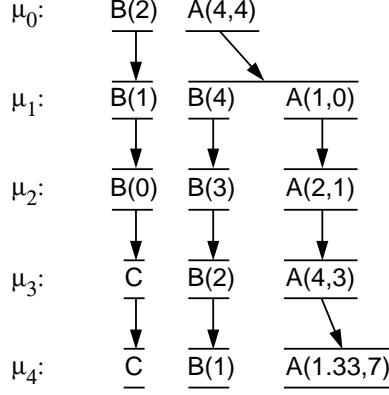


Figure 6: The initial sequence of strings generated by the parametric L-system specified in equation (2)

An example of a parametric L-system is given below.

$$\begin{aligned}
 \omega &: B(2)A(4,4) \\
 p_1 &: A(x,y) : y \leq 3 \rightarrow A(x*2, x+y) \\
 p_2 &: A(x,y) : y > 3 \rightarrow B(x)A(x/y, 0) \\
 p_3 &: B(x) : x < 1 \rightarrow C \\
 p_4 &: B(x) : x \geq 1 \rightarrow B(x-1)
 \end{aligned} \tag{2}$$

It is assumed that a module replaces itself if no matching production is found in the set  $P$ . The words obtained in the first few derivation steps are shown in Figure 6.

Productions in parametric OL-systems are context-free, *i.e.*, applicable regardless of the context in which the predecessor appears. A context-sensitive extension is necessary to model information exchange between neighboring modules. In the parametric case, each component of the production predecessor (the *left context*, the *strict predecessor* and the *right context*) is a parametric word with letters from the alphabet  $V$  and formal parameters from the set  $\Sigma$ . Any formal parameters may appear in the condition and the production successor.

A sample context-sensitive production is given below:

$$A(x) < B(y) > C(z) : x + y + z > 10 \rightarrow E((x+y)/2)F((y+z)/2). \tag{3}$$

The left context is separated from the strict predecessor by the symbol  $<$ . Similarly, the strict predecessor is separated from the right context by the symbol  $>$ . Production 3 can be applied to the module  $B(5)$  that appears in a parametric word

$$\cdots A(4)B(5)C(6) \cdots \tag{4}$$

since the sequence of letters  $A, B, C$  in the production and in parametric word (4) are the same, the numbers of formal parameters and actual parameters coincide, and the condition  $4 + 5 + 6 > 10$  is true. As a result of the production application, the module  $B(5)$  will be replaced by a pair of modules  $E(4.5)F(5.5)$ . Naturally, the modules  $A(4)$  and  $C(6)$  will be replaced by other productions in the same derivation step.

Productions in 2L-systems use context on both sides of the strict predecessor. 1L-systems are a special case of 2L-systems in which context appears only on one side of the productions.

When no production explicitly listed as a member of the production set  $P$  matches a module in the rewritten string, we assume that an appropriate *identity production* belongs to  $P$  and replaces this module by itself. Under this assumption, a parametric L-system  $G = \langle V, \Sigma, \omega, P \rangle$  is called *deterministic* if and only if for each module  $A(t_1, t_2, \dots, t_n) \in V \times \Re^*$  the production set includes exactly one matching production.

If more than one production in  $P$  matches a module, an additional mechanism is needed to select which production should be applied to this module. In *stochastic* L-systems, this decision is based on random factors. In the most extensive case, a production has the format:

$$id : lc < pred > rc : cond \rightarrow succ : \pi$$

where  $id$  is the production identifier (label),  $lc$ ,  $pred$ , and  $rc$  are the left context, the strict predecessor, and the right context,  $cond$  is the condition,  $succ$  is the successor, and  $\pi$  is an arithmetic expression returning a non-negative number called the *probability factor*. If  $\hat{P} \subseteq P$  is the set of productions matching a given module  $A(t_1, t_2, \dots, t_n) \in V \times \Re^*$  in the rewritten string, then the probability  $\text{prob}(p_k)$  of applying a particular production  $p_k \in \hat{P}$  to this module is equal to:

$$\text{prob}(p_k) = \frac{\pi(p_k)}{\sum_{p_i \in \hat{P}} \pi(p_i)}$$

In general, this probability is not a constant associated with a production, but may depend on the parameter values in the rewritten module and its context.

An example of a context-sensitive stochastic parametric L-system is given below.

$$\begin{aligned} \omega &: A(1)B(3)A(5) \\ p_1 &: A(x) \rightarrow A(x+1) : 2 \\ p_2 &: A(x) \rightarrow B(x-1) : 3 \\ p_3 &: A(x) : x > 3 \rightarrow C(x) : x \\ p_4 &: A(x) < B(y) > A(z) : y < 4 \rightarrow B(x+z)A(y) \end{aligned}$$

The productions  $p_1$ ,  $p_2$ , and  $p_3$  replace module  $A(x)$  by  $A(x+1)$ ,  $B(x-1)$ , or  $C(x)$ . If the value of parameter  $x$  is less than or equal to 3, only the first two productions match  $A(x)$ . The probabilities of applying each production are:  $\text{prob}(p_1) = 2/(2+3) = 0.4$ , and  $\text{prob}(p_2) = 3/(2+3) = 0.6$ . If parameter  $x$  is greater than 3, production  $p_3$  also matches the module  $A(x)$ , and the probability of applying each production depends on the value of  $x$ . For example, if  $x$  is equal to 5, these probabilities are:  $\text{prob}(p_1) = 2/(2+3+5) = 0.2$ ,  $\text{prob}(p_2) = 3/(2+3+5) = 0.3$ , and  $\text{prob}(p_3) = 5/(2+3+5) = 0.5$ . The context-sensitive production  $p_4$  replaces a module  $B(y)$  with left context  $A(x)$  and right context  $A(z)$  by the pair of modules  $B(x+z)A(y)$ . The application of this production is guarded by condition  $y < 4$ . Taking all these factors into account, the first derivation step may have the form:

$$A(1)B(3)A(5) \implies A(2)B(6)A(3)C(5)$$

It was assumed that, as a result of random choice, production  $p_1$  was applied to the module  $A(1)$ , and production  $p_3$  to the module  $A(5)$ . Production  $p_4$  was applied to the module  $B(3)$ , because it occurred with the required left and right context, and the condition  $3 < 4$  was true.

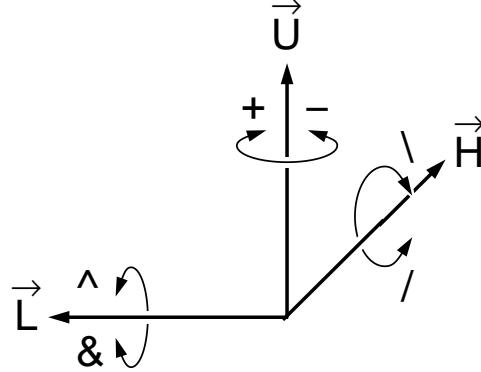


Figure 7: Controlling the turtle in three dimensions

## 5 The turtle interpretation of L-systems

Strings generated by L-systems may be interpreted geometrically in many different ways [57, 61]. Below we outline the *turtle interpretation* of parametric L-systems, introduced by Szilard and Quinton [73], and extended by Prusinkiewicz [51, 52] and Hanan [24, 25]. A tutorial exposition is included in [61], and subsequent results are presented in [25]. The summary below is based on [30, 52] and [61].

After a string has been generated by an L-system, it is scanned sequentially from left to right, and the consecutive symbols are interpreted as commands that maneuver a LOGO-style turtle [1, 50] in three dimensions. The turtle is represented by its *state*, which consists of turtle *position* and *orientation* in the Cartesian coordinate system, as well as various attribute values, such as current *color* and *line width*. The position is defined by a vector  $\vec{p}$ , and the orientation is defined by three vectors  $\vec{H}$ ,  $\vec{L}$ , and  $\vec{U}$ , indicating the turtle's *heading* and the directions to the *left* and *up* (Figure 7). These vectors have unit length, are perpendicular to each other, and satisfy the equation  $\vec{H} \times \vec{L} = \vec{U}$ . Rotations of the turtle are expressed by the equation:

$$[\vec{H}' \quad \vec{L}' \quad \vec{U}'] = [\vec{H} \quad \vec{L} \quad \vec{U}] \mathbf{R},$$

where  $\mathbf{R}$  is a  $3 \times 3$  rotation matrix [14]. Specifically, rotations by angle  $\alpha$  about vectors  $\vec{U}$ ,  $\vec{L}$  and  $\vec{H}$  are represented by the matrices:

$$\mathbf{R}_U(\alpha) = \begin{bmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$\mathbf{R}_L(\alpha) = \begin{bmatrix} \cos \alpha & 0 & -\sin \alpha \\ 0 & 1 & 0 \\ \sin \alpha & 0 & \cos \alpha \end{bmatrix},$$

$$\mathbf{R}_H(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix}.$$

Changes in the turtle's state are caused by interpretation of specific symbols, each of which may be followed by parameters. If one or more parameters are present, the value of the first parameter affects the turtle's state. If the symbol is not followed by any parameter, default values specified outside the L-system are used. The following list specifies the basic set of symbols interpreted by the turtle.

### Symbols that cause the turtle to move and draw

- $F(s)$  Move forward a step of length  $s$  and draw a line segment from the original to the new position of the turtle.
- $f(s)$  Move forward a step of length  $s$  without drawing a line.
- $@O(r)$  Draw a sphere of radius  $r$  at the current position.

### Symbols that control turtle orientation in space (Figure 7)

- $+(\theta)$  Turn left by angle  $\theta$  around the  $\vec{v}$  axis. The rotation matrix is  $\mathbf{R}_U(\theta)$ .
- $-(\theta)$  Turn right by angle  $\theta$  around the  $\vec{v}$  axis. The rotation matrix is  $\mathbf{R}_U(-\theta)$ .
- $\&(\theta)$  Pitch down by angle  $\theta$  around the  $\vec{z}$  axis. The rotation matrix is  $\mathbf{R}_L(\theta)$ .
- $\wedge(\theta)$  Pitch up by angle  $\theta$  around the  $\vec{z}$  axis. The rotation matrix is  $\mathbf{R}_L(-\theta)$ .
- $/( \theta )$  Roll left by angle  $\theta$  around the  $\vec{h}$  axis. The rotation matrix is  $\mathbf{R}_H(\theta)$ .
- $\backslash( \theta )$  Roll right by angle  $\theta$  around the  $\vec{h}$  axis. The rotation matrix is  $\mathbf{R}_H(-\theta)$ .
- | Turn  $180^\circ$  around the  $\vec{v}$  axis. This is equivalent to  $+(180)$  or  $-(180)$ .

### Symbols for modeling structures with branches

- [ Push the current state of the turtle (position, orientation and drawing attributes) onto a pushdown stack.
- ] Pop a state from the stack and make it the current state of the turtle. No line is drawn, although in general the position and orientation of the turtle are changed.

### Symbols for creating and incorporating surfaces

- { Start saving the subsequent positions of the turtle as the vertices of a polygon to be filled.
- }
- Fill the saved polygon.
- $\sim$  Draw the surface identified by the symbol immediately following the  $\sim$  at the turtle's current location and orientation.

$\text{@PS}(i, \text{basis})$  Begin definition of bicubic surface  $i$  by initializing its 16 control points to  $(0, 0, 0)$ . The optional parameter  $\text{basis}$  specifies the type of patch as:

1. Bézier,
2. B-spline,
3. Cardinal spline.

If no basis is given, use Bézier surface as the default.

$\text{@PC}(i, r, c)$  Assign the current position of the turtle to the control point of surface  $i$  in row  $r$  and column  $c$ .

$\text{@PD}(i, s, t)$  Draw surface  $i$  by subdividing it into  $s$  quadrangles along the rows and  $t$  along the columns.

### Symbols that change the drawing attributes

- $\#(w)$  Set line width to  $w$ , or increase the value of the current line width by the default width increment if no parameter is given.
- $!(w)$  Set line width to  $w$ , or decrease the value of the current line width by the default width decrement if no parameter is given.
- $;(n)$  Set the index of the color map to  $n$ , or increase the value of the current index by the default colour increment if no parameter is given.
- $,(n)$  Set the index of the color map to  $n$ , or decrease the value of the current index by the default colour decrement if no parameter is given.

## 6 Examples of L-systems

This section presents selected examples that illustrate the operation of L-systems with turtle interpretation. The material is based on [30, 59], and [61].

Fractal curves are useful in explaining the operation of L-systems that do not include branches. The following L-system generates the Koch snowflake curve.

$$\begin{aligned}\omega &: F(1) - (120)F(1) - (120)F(1) \\ p_1 &: F(s) \rightarrow F(s/3) + (60)F(s/3) - (120)F(s/3) + (60)F(s/3)\end{aligned}$$

The axiom  $F(1) - (120)F(1) - (120)F(1)$  draws an equilateral triangle, with the edges of unit length. Production  $p_1$  replaces each line segment with the polygonal shape shown in Figure 8. The productions for the  $+$  and  $-$  symbols are not listed, which means that the corresponding modules will be replaced by themselves during the derivation. The same effect could have been obtained by explicit inclusion of productions:

$$\begin{aligned}p_2 &: +(a) \rightarrow +(a) \\ p_3 &: -(a) \rightarrow -(a)\end{aligned}$$

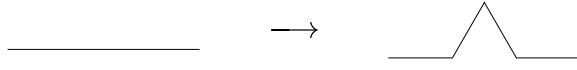


Figure 8: The production  $F(s) \rightarrow F(s/3) + (60)F(s/3) - (120)F(s/3) + (60)F(s/3)$

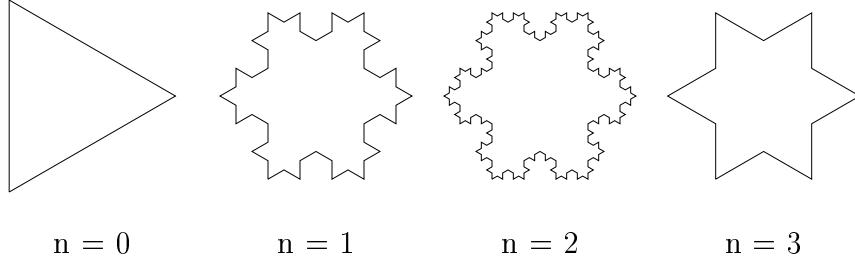


Figure 9: The snowflake curve after  $n = 0, 1, 2$ , and  $3$  derivation steps

The axiom and the first three derivation steps are illustrated in Figure 9.

The next L-system generates the developmental sequence of the compound leaf model presented in Figure 3.

$$\begin{aligned} \omega &: !(3)F(1, 1) \\ p_1 &: F(s, t) : t == 1 \rightarrow F(s, 2)[-!(1)F(s, 1)][+!(1)F(s, 1)]F(s, 2)!1F(s, 1) \\ p_2 &: F(s, t) : t == 2 \rightarrow F(2 * s, 2) \\ p_3 &: !(w) : w < 2 \rightarrow !(3) \end{aligned}$$

The axiom and productions  $p_1$  and  $p_2$  include modules  $F$  with two parameters. The first parameter specifies the length of the line representing the module. The second parameter determines whether the module is an apex ( $t == 1$ ) or an internode ( $t == 2$ ). The graphical interpretation of both productions is shown in Figure 3. The branching angle associated with symbols  $+$  and  $-$  is set to  $45^\circ$  by a global variable outside the L-system. Production  $p_3$  is used to make the lines representing the internodes wider than the lines representing the apices.

The following example illustrates the application of a stochastic L-system to the generation of a three-dimensional tree. The model is based on the analysis of tree growth by Borchert and Slade [7].

$$\begin{aligned} \omega &: FA(1) \\ p_1 &: A(k) \rightarrow /(\phi)[+(\alpha)FA(k+1)] - (\beta)FA(k+1) : \min\{1, (2k+1)/k^2\} \\ p_2 &: A(k) \rightarrow /(\phi)B - (\beta)FA(k+1) : \max\{0, 1 - (2k+1)/k^2\} \end{aligned}$$

The generation of the tree begins with a single internode  $F$  terminated by apex  $A(1)$ . The parameter of the apex ( $k$ ) acts as a counter of derivation steps. Production  $p_1$  describes the creation of two new branches, whereas production  $p_2$  describes the production of a branch segment and a dormant bud  $B$ . Probabilities of these events are equal to

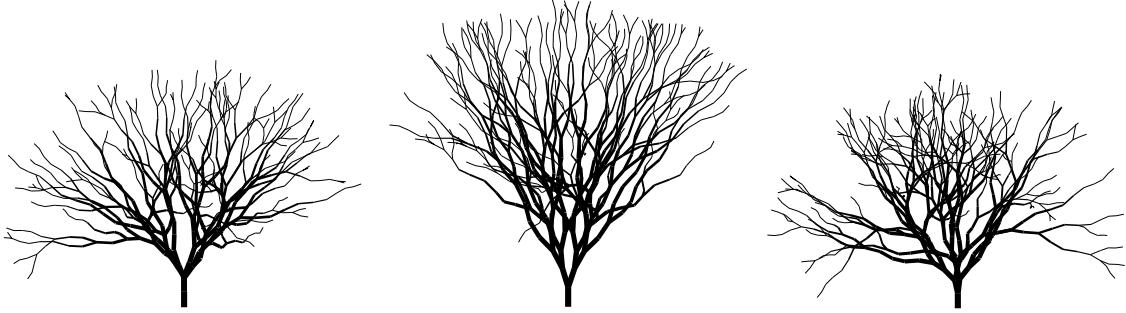


Figure 10: Sample tree structures generated using a stochastic L-system

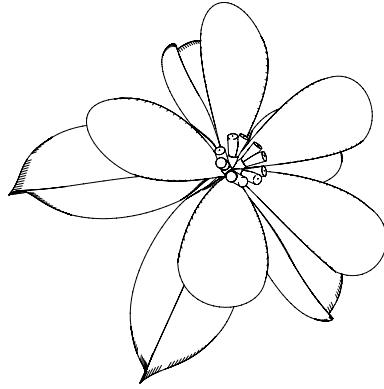


Figure 11: A stylized flower

$\text{prob}(p_1) = \min\{1, (2k+1)/k^2\}$  and  $\text{prob}(p_2) = 1 - \text{prob}(p_1)$ , respectively, thus the probability of branching (captured by production  $p_1$ ) gradually decreases as the tree grows older. A detailed justification of these formulas is given in [7, 59]. Figure 10 shows side views of three sample trees after 18 derivation steps. The branching angles, equal to  $\phi = 90^\circ$ ,  $\alpha = 32^\circ$ , and  $\beta = 20^\circ$ , yield a sympodial branching structure (new shoots do not continue the growth direction of the preceding segments). This structure is representative to the Leeuwenberg's model of tree architecture identified by Hallé *et al.* [23], although no attempt to capture a particular tree species was made.

The final example of this section illustrates the inclusion of predefined surfaces into a model. The following L-system generates the stylized flower shown in Figure 11.

$$\begin{aligned}\omega &: /(154)B \\ p_1 &: B \rightarrow [\&(72)\#F(5)!P] \\ p_2 &: P \rightarrow [S/(72)S/(72)S/(72)S/(72)S] \\ p_3 &: S \rightarrow [\wedge(103) \sim s][\wedge(72) \sim p][\wedge(34)F(1)\#[-F(1)][+F(1)]]\end{aligned}$$

Production  $p_1$  creates the stalk  $F(5)$  and the symbol  $P$ , which gives rise to the *perianth*. Production  $p_2$  describes the perianth as consisting of five sectors  $S$ , rotated with respect to



Figure 12: Simulated development of a bluebell flower

each other by  $72^\circ$ . According to production  $p_3$ , each sector consists of the *sepal* represented by a predefined surface  $s$ , the petal represented by a predefined surface  $p$ , and a configuration of line segments representing an *anther*. The exact shape of the sepals and petals is defined outside the L-system. This L-system does not simulate the process of flower development, but uses productions to capture the flower’s structure in a compact, hierarchical manner.

## 7 Life, death, and reproduction

The L-systems considered so far were *propagating*. Each module, once created, continued to exist indefinitely or gave rise to one or more children, but never disappeared without a trace. The natural processes of plant development, however, often involve the programmed death of selected modules and their removal from the resulting structures. We consider these phenomena in the present section.

The original approach to simulating module death was to use *non-propagating* L-systems, which incorporate *erasing* productions [27]. In the context-free case these productions have the form  $A \longrightarrow \varepsilon$ , where  $\varepsilon$  denotes the empty string. Intuitively, the module  $A$  is replaced by “nothing” and is removed from the structure. Erasing productions can faithfully simulate the disappearance of individual modules placed at the extremities of the branching structure (that is, not followed by other modules). For example, in the developmental sequence shown in Figure 12 (described in detail in [55]), erasing productions have been used to model the fall of petals.

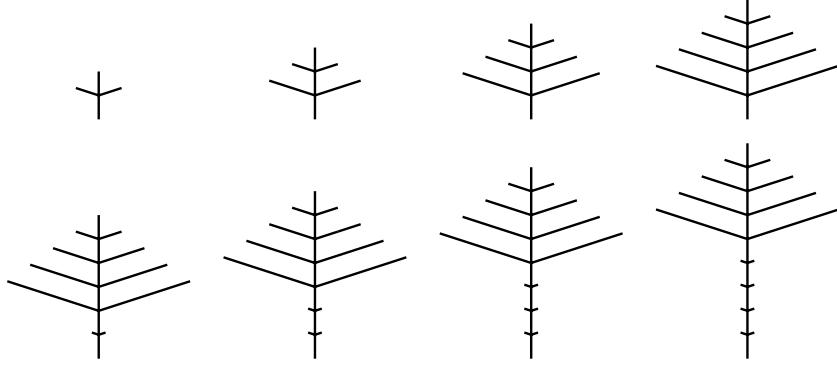


Figure 13: A developmental sequence generated by the L-system specified in Equation 5. The images shown represent derivation steps 2 through 9.

The modeling task becomes more difficult when an entire structure, such as a branch, is shed by the plant. A plant can control this process by *abscission*, that is the development of a pithy layer of cells that weakens the stem of a branch at its base. Obviously, abscission is represented more faithfully by cutting a branch off than by simultaneously erasing all of its modules. In order to simulate shedding, Hanan [25] extended the formalism of L-systems with the “cut symbol” %, which causes the removal of the remainder of the branch that follows it. For example, in the absence of other productions, the derivation step given below takes place:

$$a[b\%[cd)e[\%f]]g[h[\%i]j]k \implies a[b]g[h[]j]k$$

It is interesting to consider the operation of the cut symbol in the context of Figure 5 from Section 3. Information about the occurrence of a cut symbol does not flow from the parent module or its immediate neighbors to their children, but propagates from the cutting point to the extremities of the branches in the same manner positional information does (*Problem 7.1*).

A simple example of an L-system incorporating the cut symbol is given below:

$$\begin{aligned}
 \omega &: A \\
 p_1 : A &\rightarrow F(1)[-X(3)B][+X(3)B]A \\
 p_2 : B &\rightarrow F(1)B \\
 p_3 : X(d) &: d > 0 \rightarrow X(d - 1) \\
 p_4 : X(d) &: d == 0 \rightarrow U\% \\
 p_5 : U &\rightarrow F(0.3)
 \end{aligned} \tag{5}$$

According to production  $p_1$ , in each derivation step the apex of the main axis  $A$  produces an internode  $F$  of unit length and a pair of lateral apices  $B$ . Each apex  $B$  extends a branch by forming a succession of internodes  $F$  (production  $p_2$ ). After three steps from branch initiation (controlled by production  $p_3$ ), production  $p_4$  inserts the cut symbol % and an auxiliary symbol  $U$  at the base of the branch. In the next step, the cut symbol removes the branch, while symbol  $U$  inserts a marker  $F(0.3)$  indicating a “scar” left by the removed branch. The resulting developmental sequence is shown in Figure 13. The initial steps capture the growth of a *basipetal* structure (developed most extensively at the base).

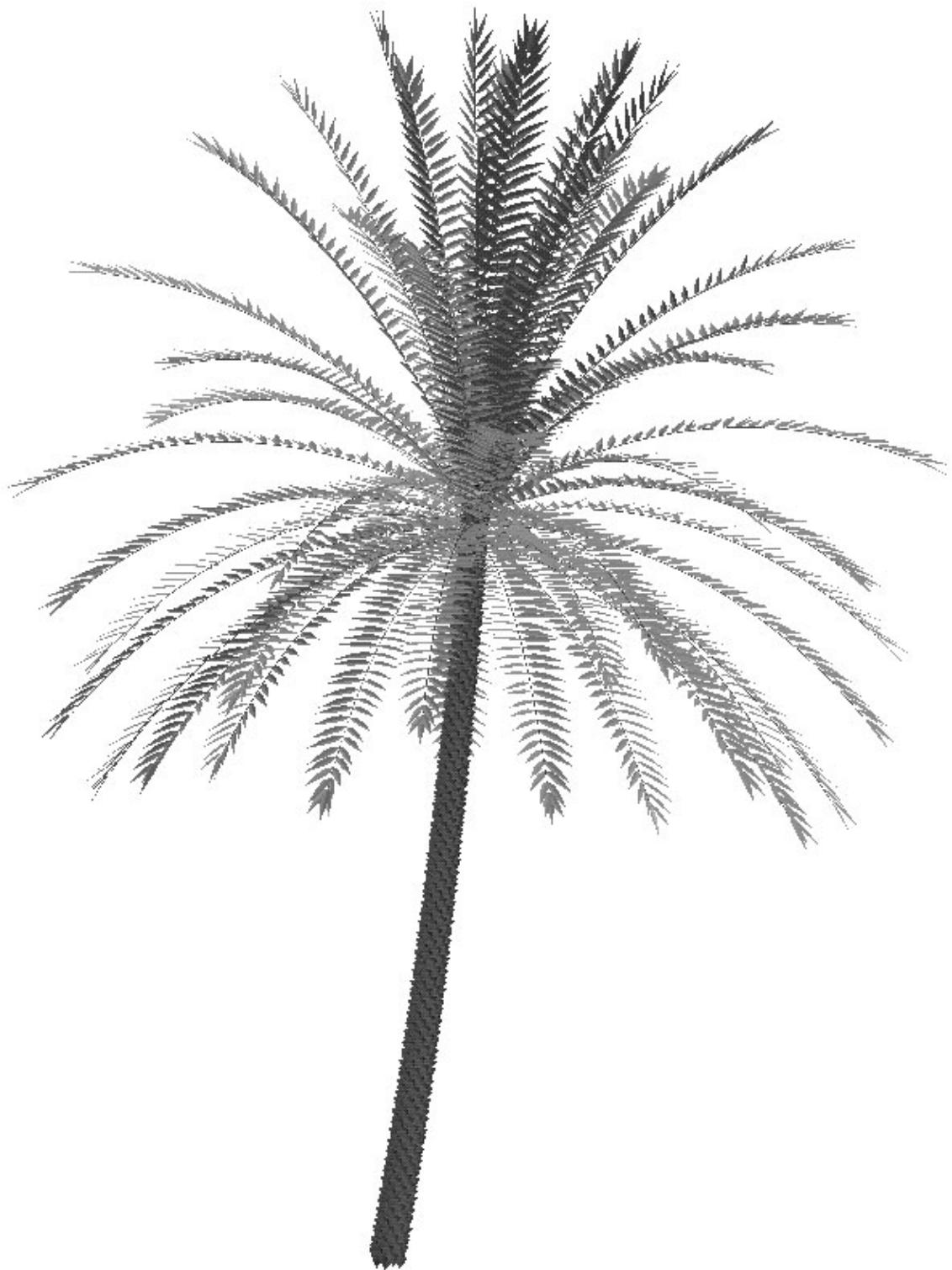


Figure 14: A model of the date palm (*Phoenix dactylifera*). This image was created using an L-system with the general structure specified in Equation 5.

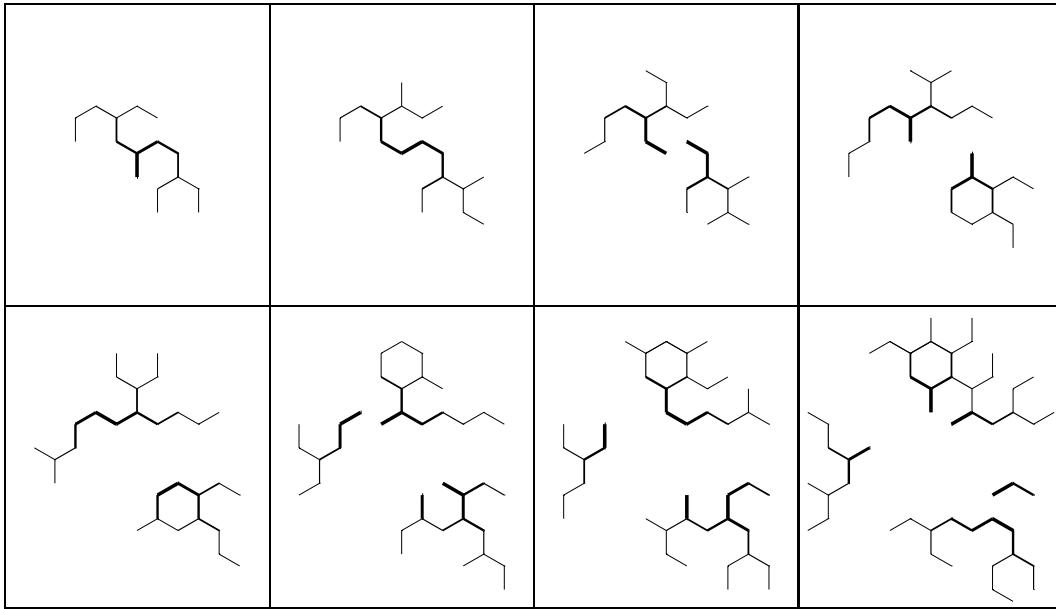


Figure 15: Development of the rhizomatous system of *Alpinia speciosa*. The images show consecutive stages of simulation generated in 6 to 13 derivation steps. Line width indicates the age of the rhizome segments. Each segment dies and disappears seven steps after its creation.

Beginning at derivation step 6, the oldest branches are shed, creating an impression of a tree crown of constant shape and size moving upwards. The crown is in a state of dynamic equilibrium: the addition of new branches and internodes at the apices is compensated by the loss of branches further down (*Problem 7.2*).

The state of dynamic equilibrium can be easily observed in the development of palms, where new leaves are created at the apex of the trunk while old leaves are shed at the base of the crown (Figure 14). Both processes take place at the same rate, thus an adult palm carries an approximately constant number of leaves. This phenomenon has an interesting physiological explanation: palms are unable to gradually increase the diameter of their trunk over time, thus the flow of substances through the trunk can support only a crown of a constant size.

In the case of falling leaves and branches, the parts separated from the main structure die. Separation of modules can also lead to the reproduction of plants. This phenomenon takes place, for example, when plants propagate through *rhizomes*, that is stems that grow horizontally below the ground and bear buds which produce vertical shoots. The rhizome segments (internodes) have a finite life span, and rot progressively from the oldest end, thus dividing the original plant into independent organisms.

A model of the propagation of rhizomes in *Alpinia specioza*, a plant of the ginger family, was proposed by Bell, Roberts, and Smith [5]. A simulation carried out using an L-system reimplementation of this model is shown in Figure 15. All rhizome segments are assumed

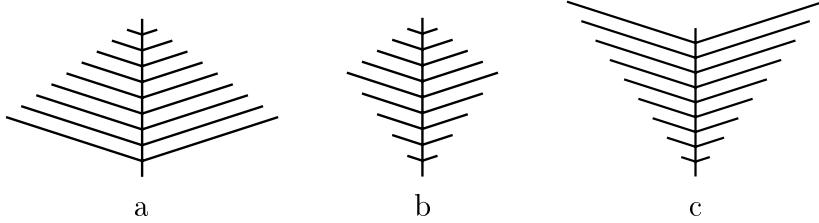


Figure 16: Basitonic (a), mesotonic (b), and acrotonic (c) branching structures differ by the position of the most developed branches on the stem.

to have the same length. Each year (one derivation step in the simulation), an apex produces one or two daughter segments. The decision mechanism is expressed using stochastic productions. The segments persist for seven years from their creation, then die off thereby dividing the plant.

The death of segments can be captured using productions of type  $F \rightarrow f$ , which replace “old” segments  $F$  by invisible links (turtle movements)  $f$ . This replacement guarantees that the separated organisms will maintain their relative positions. Although the effect is visually correct, maintaining any type of connection between the separated plants is artificial. An alternative solution is to extend the notion of L-systems so that they operate on *sets* of words, instead of individual words. In this case, once a branching structure becomes disconnected due to the death of an internode, each of the resulting structures develops separately (*Problems 7.3, 7.4*).

## 8 Information flow in growing plants

In this section we consider the propagation of control information through the structure of the developing plant (*endogenous information flow* [53]), which is captured by context-sensitive productions in the framework of L-systems. The conceptual elegance and expressive power of context-sensitive productions are among the most important assets of L-systems in modeling applications (*Problem 8.1*).

When modeling the development of branching structures, one can often divide aspects of a particular phenomenon into those that can be modeled using OL-systems, and those that cannot. For example, *acropetal* flowering sequences (with the flowering zone progressing upwards from the base of the plant) generally can be simulated using OL-systems (even without parameters), since the flowers develop in the same order in which they have been formed by the apices of their supporting branches. In contrast, *basipetal* sequences (with the flowering zone progressing downwards) require additional control mechanisms, and can be best explained in terms of the flow of control *signals* through the growing structures [31, 61, 62]. An analogous distinction can be observed between *basitonic* structures on the one hand, and *mesotonic* and *acrotonic* structures on the other hand (see Figure 16). It is intuitively clear that basitonic structures can be created using OL-systems: the lower branches are created first and, consequently, have more time to develop than the upper branches. For example, the following L-system simulates the development of the simple monopodial structure shown in Figure 17.

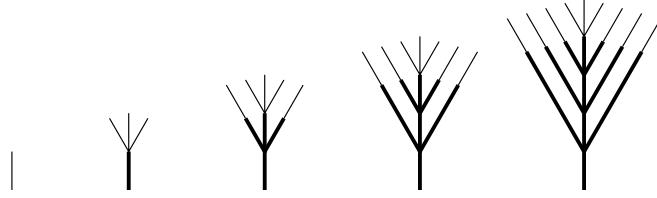


Figure 17: Development of a basitonic branching structure. The thin lines indicate segments created in the current derivation step.

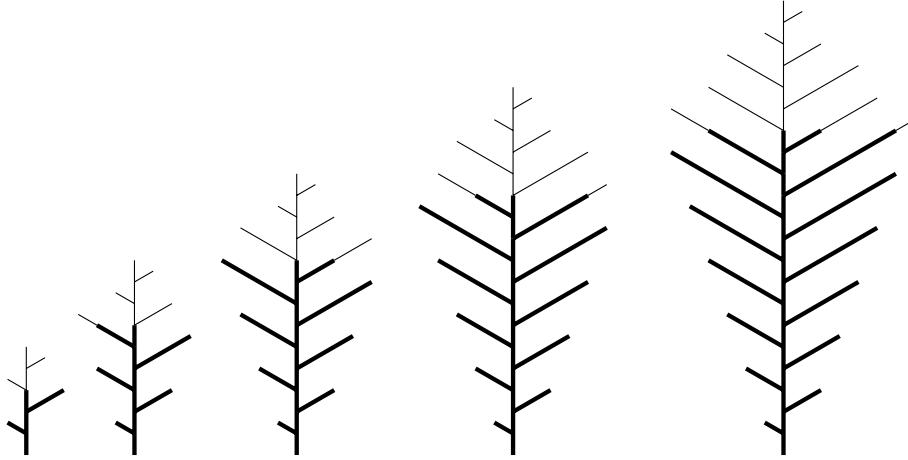


Figure 18: Development of a mesotonic branching structure controlled by an acropetal signal. Wide lines indicate the internodes reached by the signal. The stages shown correspond to derivation lengths 12 – 24 – 36 – 48 – 60

$$\begin{aligned}\omega &: A \\ p_1 &: A \rightarrow F[-B][+B]A \\ p_2 &: B \rightarrow FB\end{aligned}$$

In contrast, indeterminate (arbitrarily large) mesotonic and acrotonic structures cannot be generated using simple deterministic OL-systems without parameters [60]. The proposed mechanisms for modeling these structures can be divided into two categories: those using parameters to characterize the *growth potential* or *vigor* of individual apices [46, 47], and those postulating control of development by signals [18, 31]. The following L-system, adapted from [61, page 77], simulates the development of the mesotonic structure shown in Figure 18 using an acropetal (upward moving) signal.

```

#define m 3 /* plastochron of the main axis */
#define n 4 /* plastochron of the branch */
#define u 4 /* signal propagation rate in the main axis */
#define v 2 /* signal propagation rate in the branch */

ignore : + - /
 $\omega : S(0)F(1,0)A(0)$ 
 $p_1 : A(i) : i < m - 1 \rightarrow A(i + 1)$ 
 $p_2 : A(i) : i == m - 1 \rightarrow [+(60)F(1,1)B(0)]F(1,0)/(180)A(0)$ 
 $p_3 : B(i) : i < n - 1 \rightarrow B(i + 1)$ 
 $p_4 : B(i) : i == n - 1 \rightarrow F(1,1)B(0)$ 
 $p_5 : S(i) : i < u + v \rightarrow S(i + 1)$ 
 $p_6 : S(i) : i == u + v \rightarrow \varepsilon$ 
 $p_7 : S(i) < F(l,o) : (o == 0) \& \& (i == u - 1) \rightarrow \#F(l,o)!S(0)$ 
 $p_8 : S(i) < F(l,o) : (o == 1) \& \& (i == v - 1) \rightarrow \#F(l,o)!S(0)$ 
 $p_9 : S(i) < B(j) \rightarrow \varepsilon$ 

```

The above L-system operates under the assumption that the context-sensitive production  $p_9$  takes priority over  $p_3$  or  $p_4$ , and that the symbols  $+$ ,  $-$ ,  $/$  are ignored during the context matching (*c.f.* [61]). The axiom  $\omega$  describes the initial structure as an internode  $F$  terminated by an apex  $A$ . The signal  $S$  is placed at the base of this structure. According to productions  $p_1$  and  $p_2$ , the apex  $A$  periodically produces a lateral branch and adds an internode to the main axis. The period (called the *plastochrone* of the main axis) is controlled by the constant  $m$ . Productions  $p_3$  and  $p_4$  describe the development of the lateral branches, where new segments  $F$  are added with plastochrone  $n$ . Productions  $p_5$  to  $p_8$  describe the propagation of the signal through the structure. The signal propagation rate is  $u$  in the main axis, and  $v$  in the branches. Production  $p_9$  removes the apex  $B$  when the signal reaches it, thus terminating the development of the corresponding lateral branch. Figure 18 shows that, for the values of plastochrones and signal propagation rates indicated in Equation 6, the lower branches have less time to develop than the higher branches, and a mesotonic structure results.

In the above model, the parameter associated with the signal was used to control its propagation rate. The signal itself acted in a binary way: it was either present or absent in a particular internode, and controlled the development by simply removing apices  $B$  from the structure. In many models, however, signals represent quantifiable entities, such as the amount of mineral substances absorbed by roots and carried upwards, or the amount of photosynthate produced by the leaves and transported down the tree. For example, Figure 19 illustrates an extension of the tree model by Borchert and Slade (*c.f.* Section 6) with an endogenously controlled mechanism for shedding branches. The model operates under the assumption that photosynthates are produced by the leaves located on the apical branch segments (shoots) and are used at a constant rate by the internodes. Information about the balance of photosynthates is carried basipetally (from the shoots towards the trunk). A branch that produces less photosynthate than it uses becomes a liability and is shed by the tree. The shedding of branches has an important impact on the structure and visual

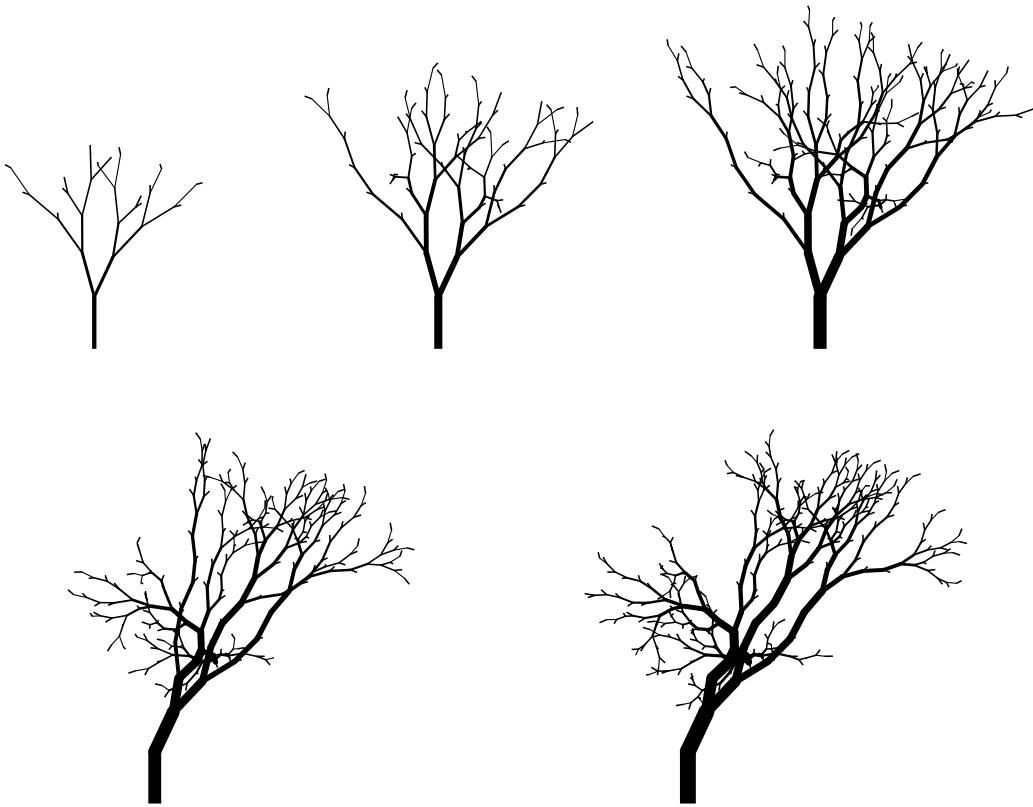


Figure 19: A modification of the tree model by Borchert and Slade [7]. The branches that produce less photosynthate than they use are shed by the tree. For clarity, leaves are not shown.

presentation of older trees (bottom row of Figure 19).

In the above model, the shedding had only a limited effect on the development of the remaining parts of the tree. In nature, trees may compensate for the loss of a branch by the more vigorous growth of other branches. A model that captures this phenomenon has been proposed by Borchert and Honda [6], and is illustrated in Figure 20. In this case, basipetal signals originating at each node carry information about the size of branches. This information is used to allocate “fluxes” that propagate acropetally and determine the vigor of the apices. Pruning of a branch redirects the fluxes to the remaining branches and accelerates their growth.

Context-sensitive L-systems can also be used to simulate phenomena other than endogenously propagating signals. For example, Figure 21 shows a simple model of a plant attacked by an insect. The insect feeds on the apical buds; a branch that no longer carries any buds wilts. The insect is assumed to move only along the branches, thus its motions can be captured using context-sensitive L-systems. In the example under consideration, the insect systematically visits all buds, using the depth-first strategy for traversing a tree structure. Extensions of this model, including different traversing and feeding strategies, numbers of insects, etc., can be introduced.

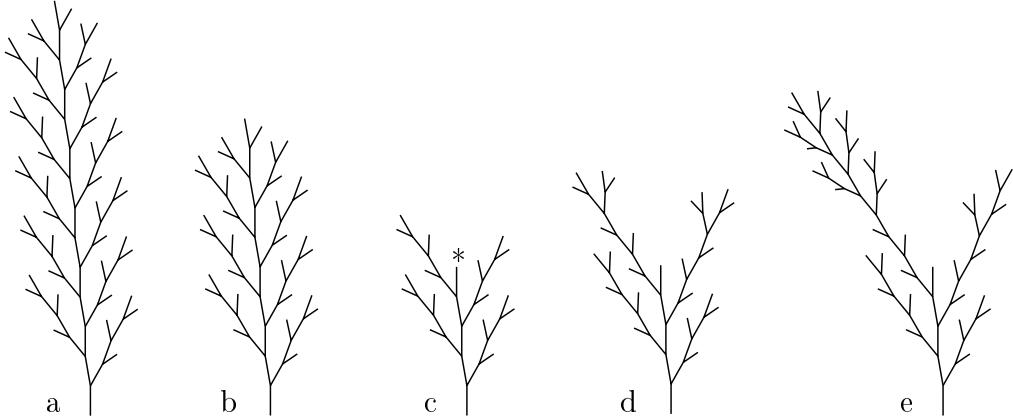


Figure 20: Development of a branching structure simulated using an L-system implementation of the model by Borchert and Honda. (a) Development not affected by pruning; (b, c) the structure immediately before and after pruning; (d, e) the subsequent development of the pruned structure.

## 9 Plants and the environment

The turtle interpretation of L-systems described in Section 5 creates the geometric representation of the model in a postprocessing step, with no effect on the operation of the L-system itself. Referring to Figure 5, the flow of information regarding position and orientation of the modules is postponed until all component modules are already determined. In this section we present an *environmentally-sensitive extension* of L-systems, which makes information about position and orientation of the modules available at each derivation step. Consequently, it is possible to model the influence of predefined environmental factors, such as the presence of obstacles, on a growing plant. Our presentation is based on the paper [59] and includes its edited sections.

In environmentally-sensitive L-systems, the generated string is interpreted after each derivation step, and turtle attributes found during the interpretation are returned as parameters to reserved *query modules* in the string. Each derivation step is performed as in parametric L-systems, except that the parameters associated with the query modules remain undefined. During interpretation, these modules are assigned values that depend on the turtle’s position and orientation in space. Syntactically, the query modules have the form  $?X(x, y, z)$ , where  $X = P, H, U$ , or  $L$ . Depending on the actual symbol  $X$ , the values of parameters  $x$ ,  $y$ , and  $z$  represent a position or an orientation vector. In the two-dimensional case, the coordinate  $z$  may be omitted.

The operation of the query module is illustrated by a simple environmentally-sensitive L-system, given below.

$$\begin{aligned} \omega &: A \\ p_1 : A &\rightarrow F(1)?P(x, y) - A \\ p_2 : F(k) &\rightarrow F(k + 1) \end{aligned}$$

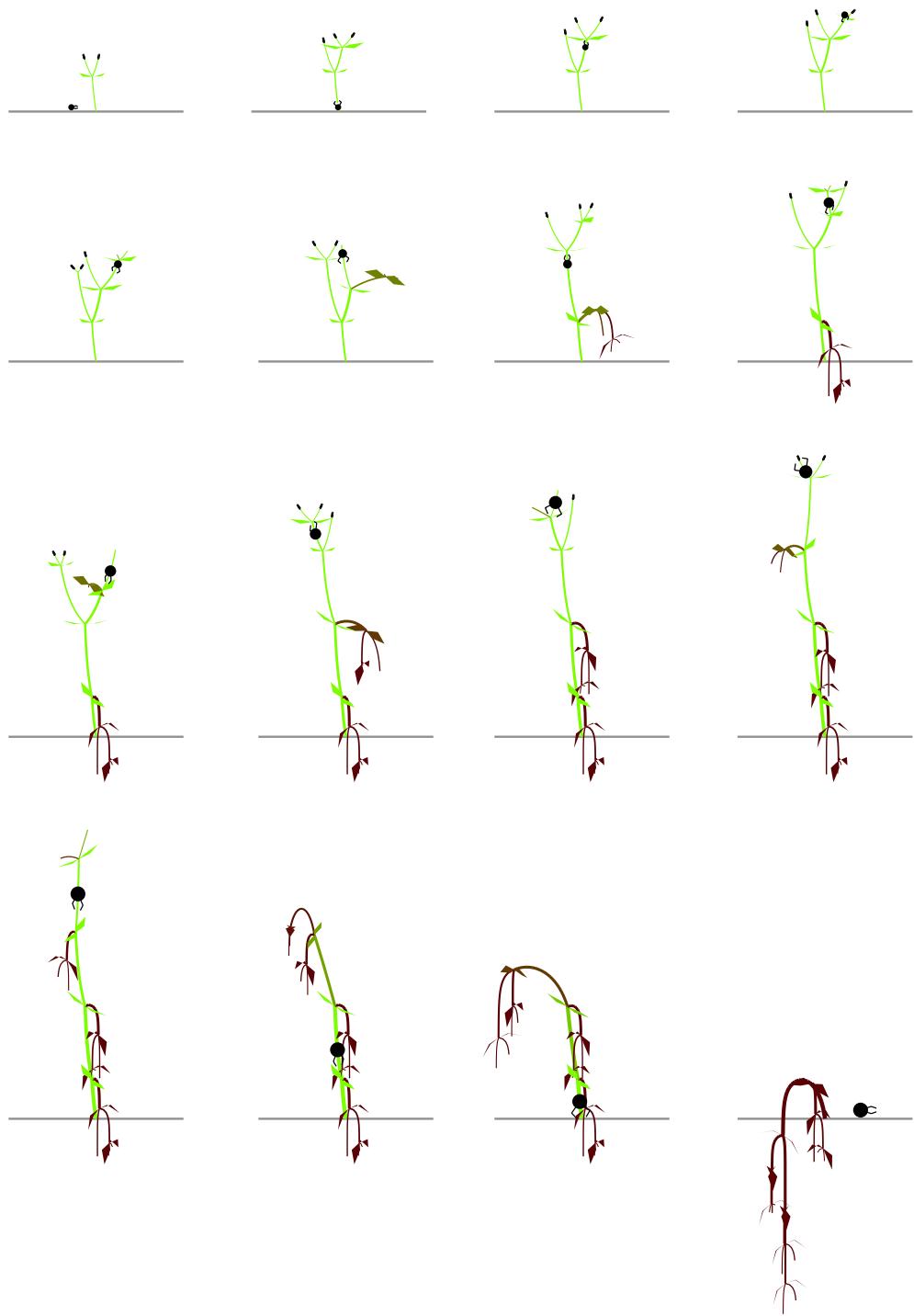


Figure 21: Simulation of the development of a plant attacked by an insect

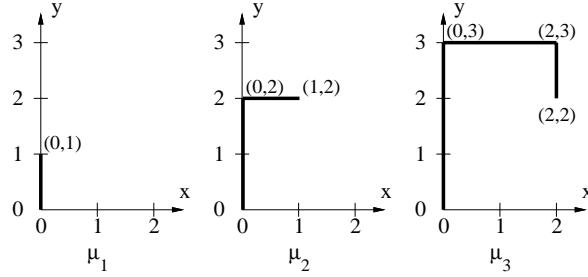


Figure 22: Assignment of values to query modules

The following strings are produced during the first three derivation steps.

$$\begin{aligned}
 \mu'_0 &: A \\
 \mu_0 &: A \\
 \mu'_1 &: F(1)?P(\star, \star) - A \\
 \mu_1 &: F(1)?P(0, 1) - A \\
 \mu'_2 &: F(2)?P(\star, \star) - F(1)?P(\star, \star) - A \\
 \mu_2 &: F(2)?P(0, 2) - F(1)?P(1, 2) - A \\
 \mu'_3 &: F(3)?P(\star, \star) - F(2)?P(\star, \star) - F(1)?P(\star, \star) - A \\
 \mu_3 &: F(3)?P(0, 3) - F(2)?P(2, 3) - F(1)?P(2, 2) - A
 \end{aligned}$$

Strings  $\mu'_0$ ,  $\mu'_1$ ,  $\mu'_2$ , and  $\mu'_3$  represent the axiom and the results of production application before the interpretation steps. Symbol  $\star$  indicates an undefined parameter value in a query module. Strings  $\mu_1$ ,  $\mu_2$ , and  $\mu_3$  represent the corresponding strings after interpretation. It has been assumed that the turtle is initially placed at the origin of the coordinate system, vector  $\vec{H}$  is aligned with the  $y$  axis, vector  $\vec{I}$  points in the negative direction of the  $x$  axis, and the angle of rotation associated with module “ $-$ ” is equal to  $90^\circ$ . Parameters of the query modules have values representing the positions of the turtle shown in Figure 22.

The above example illustrates the notion of derivation in an environmentally-sensitive L-system, but it is otherwise contrived, since the information returned by the query modules is not further used. A more realistic example, presenting a simple model of tree response to pruning, is given below. As described, for example, by Hallé *et al.* [23, Chapter 4] and Bell [4, page 298], during the normal development of a tree many buds do not produce new branches and remain dormant. These buds may be subsequently activated by the removal of leading buds from the branch system (*traumatic reiteration*), which results in an environmentally-adjusted tree architecture. The following L-system represents the extreme case of this process, where buds are activated only as a result of pruning.

$$\begin{aligned}
 \omega &: FA?P(x, y) \\
 p_1 &: A > ?P(x, y) : !\text{prune}(x, y) \rightarrow @oF/(180)A \\
 p_2 &: A > ?P(x, y) : \text{prune}(x, y) \rightarrow T\% \\
 p_3 &: F > T \rightarrow S \\
 p_4 &: F > S \rightarrow SF \\
 p_5 &: S \rightarrow \varepsilon \\
 p_6 &: @o > S \rightarrow [+FA?P(x, y)]
 \end{aligned}$$

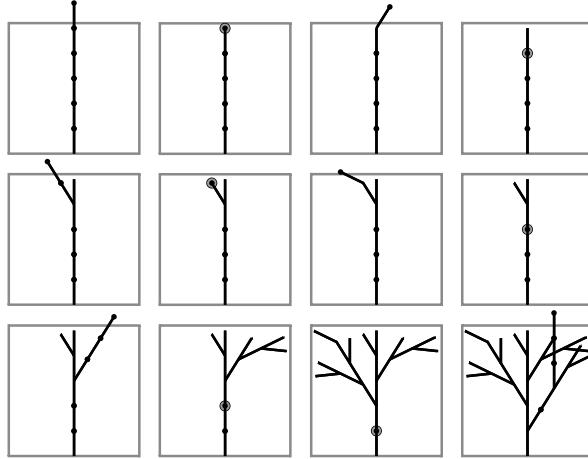


Figure 23: A simple model of a tree’s response to pruning. Top row: derivation steps 6,7,8, and 10; middle row: steps 12, 13, 14, and 17; bottom row: steps 20, 40, 75, and 94. Small black circles indicate dormant buds, the larger circles indicate the position of signal  $S$ .

The user defined function

$$\text{prune}(x, y) = (x < -L/2) \parallel (x > L/2) \parallel (y < 0) \parallel (y > L),$$

defines a square *clipping box* of dimensions  $L \times L$  that bounds the growing structure. According to axiom  $\omega$ , the development begins with an internode  $F$  supporting apex  $A$  and query module  $?P(x, y)$ . The initial development of the structure is described by production  $p_1$ . In each step, the apex  $A$  creates a dormant bud  $@o$  and an internode  $F$ . The module  $/\langle 180 \rangle$  rotates the turtle around its own axis (the heading vector  $\vec{H}$ ), thus laying a foundation for an alternating branching pattern. The query module  $?P(x, y)$ , placed by the axiom, is the right context for production  $p_1$  and returns the current position of apex  $A$ . When a branch extends beyond the clipping box, production  $p_2$  removes apex  $A$ , cuts off the query module  $?P(x, y)$  using the symbol  $\%$ , and generates the pruning signal  $T$ . In the presence of this signal, production  $p_3$  removes the last internode of the branch that extends beyond the clipping box and creates bud-activating signal  $S$ . Productions  $p_4$  and  $p_5$  propagate this signal basipetally (downwards), until it reaches a dormant bud  $@o$ . Production  $p_6$  induces this bud to initiate a lateral branch consisting of internode  $F$  and apex  $A$  followed by query module  $?P(x, y)$ . According to production  $p_1$ , this branch develops in the same manner as the main axis. When its apex extends beyond the clipping box, it is removed by production  $p_2$ , and signal  $S$  is generated again. This process may continue until all dormant buds have been activated.

Selected phases of the described developmental sequence are illustrated in Figure 23. In derivation step 6 the apex of the main axis grows out of the clipping box. In step 7 this apex and the last internode are removed from the structure, and the bud-activating signal  $S$  is generated. As a result of bud activation, a lateral branch is created in step 8. As it also extends beyond the bounding box, it is removed in step 9 (not shown). Signal  $S$  is generated again, and in step 10 it reaches a dormant bud. The subsequent development of the lateral branches, shown in the middle and bottom rows of Figure 23, follows a similar pattern.

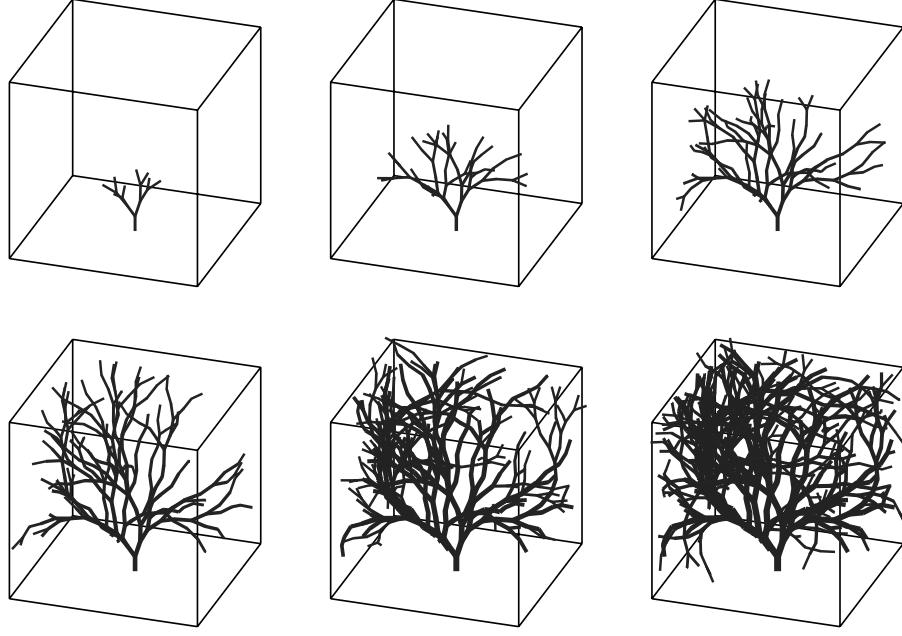


Figure 24: Simulation of tree response to pruning. The structures shown have been generated in 3, 6, 9, 13, 21, and 27 steps. Leaves have been omitted for clarity.

The above L-system simulates the response of a tree to pruning using a schematic branching structure. Below we incorporate a similar mechanism into the more realistic stochastic tree model by Borchert and Slade, discussed in Section 6.

$$\begin{aligned}
 \omega &: FA(1)?P(x, y, z) \\
 p_1 &: A(k) > ?P(x, y, z) : !\text{prune}(x, y, z) \rightarrow \\
 &\quad /(\phi)[+(\alpha)FA(k+1)?P(x, y, z)] - (\beta)FA(k+1) : \min\{1, (2k+1)/k^2\} \\
 p_2 &: A(k) > ?P(x, y, z) : !\text{prune}(x, y, z) \rightarrow \\
 &\quad /(\phi)B(k+1, k+1) - (\beta)FA(k+1) : \max\{0, 1 - (2k+1)/k^2\} \\
 p_3 &: A(k) > ?P(x, y, z) : \text{prune}(x, y, z) \rightarrow T\% \\
 p_4 &: F > T \rightarrow S \\
 p_5 &: F > S \rightarrow SF \\
 p_6 &: S \rightarrow \varepsilon \\
 p_7 &: B(m, n) > S \rightarrow [+(\alpha)FA(am + bn + c)?P(x, y, z)] \\
 p_8 &: B(m, n) > F \rightarrow B(m+1, n)
 \end{aligned}$$

According to axiom  $\omega$ , the development begins with a single internode  $F$  supporting apex  $A(1)$  and query module  $?P(x, y, z)$ . Productions  $p_1$  and  $p_2$  describe the spontaneous growth of the tree within the volume characterized by a user-defined clipping function  $\text{prune}(x, y, z)$ . Productions  $p_3$  to  $p_7$  specify the mechanism of the tree's response to pruning. Specifically, production  $p_3$  removes the apex  $A()$  after it has crossed the clipping surface, cuts off the query module  $?P(x, y, z)$ , and creates pruning signal  $T$ . Next,  $p_4$  removes the last internode of the pruned branch and initiates bud-activating signal  $S$ , which is propagated basipetally by productions  $p_5$  and  $p_6$ . When  $S$  reaches a dormant bud  $B()$ , production  $p_7$  transforms it into a branch consisting of an internode  $F$ , apex  $A()$ , and query module  $?P(x, y, z)$ .



Figure 25: A model of the topiary garden at Levens Hall, England

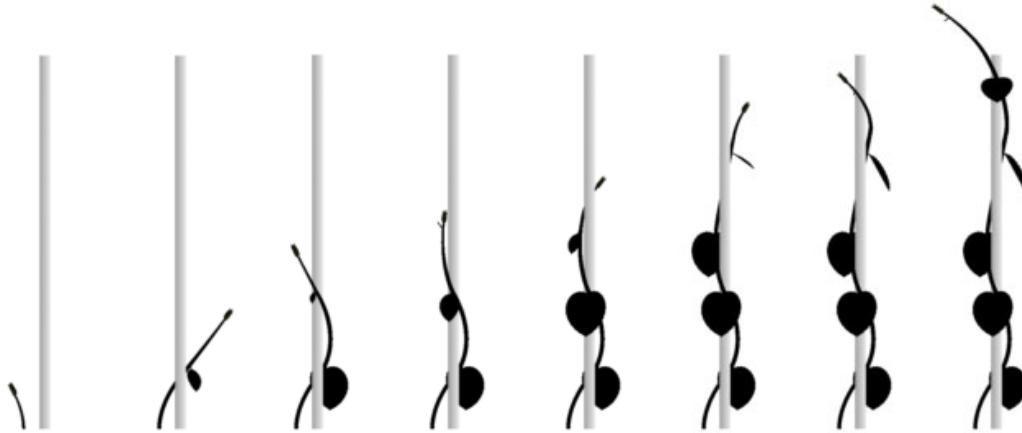


Figure 26: A simple model of a climbing plant

The parameter value assigned by production  $p_7$  to apex  $A()$  is derived as follows. According to production  $p_2$ , both parameters associated with a newly created bud  $B()$  are set to the age of the tree at the time of bud creation (expressed as the number of derivation steps). Production  $p_8$  updates the value of the first parameter ( $m$ ), so that it always indicates the actual age of the tree. The second parameter ( $n$ ) remains unchanged. The initial *biological age* [4, page 315] of the activated apex  $A()$  in production  $p_7$  is a linear combination of parameters  $m$  and  $n$ , calculated using the expression  $am + bn + c$ . Since rule  $p_1$  is more likely to be applied for young apices (for small values of parameter  $k$ ), by manipulating constants  $a$ ,  $b$ , and  $c$  it is possible to control the bifurcation frequency of branches created as a result of traumatic reiteration. This is an important feature of the model, because in nature the reiterated branches tend to be more juvenile and vigorous than the remainder of the tree [4, page 298].

The operation of this model is illustrated in Figure 24. The clipping form is a cube with an edge length 12 times longer than the internode length. The constant values used in production  $p_7$  are  $a = 0$ ,  $b = 1$ , and  $c = -5$ .

By changing the clipping function, one can shape plant models to a variety of artificial forms. For example, Figure 25 presents a synthetic image inspired by the Levens Hall garden in England, considered the most famous topiary garden in the world [9, pages 52–57].

Pruning is only one of a range of phenomena that can be modeled using environmentally-sensitive L-systems. A different example is given in Figure 26. In this case, a climbing plant detects the presence of a supporting pole and winds around it. In contrast to the earlier models of plants growing around obstacles [2, 21, 22], the L-system model captures the *nutation*, or the spiralling movement of the free stem tip “searching” for support. Once a collision with the pole has been detected, a signal is sent basipetally (down the stem), freezing further motions of the stem below the contact point.

## 10 Conclusions

L-systems were introduced almost thirty years ago and have been extensively studied, yet they continue to represent a fascinating area for further research. This situation is due to several factors. Computer graphics has made it possible to visualize the structures generated by L-systems, thus turning them from a theoretical concept to a programming language for synthesizing fractals and realistic plant images. The modeling power of L-systems, made apparent by the synthetic images, has attracted a growing number of biologists interested in modular architecture and plant development. Biological applications frequently require the inclusion of environmental factors into the models, which fuels the work on environmentally-sensitive extensions to L-systems. Furthermore, the interest of biologists in modeling actual plant species is complemented by the fundamental studies of emergence in the field of artificial life. These varied interests and applications place L-systems in the center of interdisciplinary studies bridging theoretical computer science, computer graphics, biology, and artificial life. Even the material in these notes raises many nontrivial questions. Some of them are listed in the next section.

## 11 Problems

- 1.1. The importance of simulation to the studies of emergent phenomena led Darley to the following characterization [10, Page 412]:

Emergent phenomena are those for which the amount of computation necessary for prediction from an optimal set of rules, classification and analysis, even derived from an idealised perfect understanding, can never improve upon the amount of computation necessary to simulate the system directly from our knowledge of the rules of its interactions.

Based on this characterization, propose a formal definition of emergence expressed in terms of algorithmic complexity. Illustrate your definition by examples of emergent and non-emergent phenomena. Compare your definition with Darley's own elaboration of his concept.

- 2.1. Hallé, Oldeman, and Tomlinson introduced the term *architectural model* to denote a program that determines successive stages of the development of a tree [23]. Compare this notion with the developmental models of plants expressed using L-systems.
- 2.2. Room, Maillette and Hanan classified different types of construction units that can be used to describe the modular growth of plants [64]. Analyze the relationship between these units and the notion of a module in L-systems.
- 2.3. Plant morphogenesis has been studied from different perspectives and at various levels of abstraction. The resulting models include:
  - *genetic* [49] and *mechanistic* [20] models operating at the molecular level,

- *reaction-diffusion* models operating at the chemical level [34],
- *L-system* models operating at the level of modules,
- models of organ distribution and overall plant shape based on *ecological* arguments [29].

Using the indicated references as the starting point for investigation, explain how models operating at different levels relate to each other. Can a counterpart of the *correspondence principle* (“when theories correspond, their predictions must correspond”) introduced by Niels Bohr for physics be applied to describe these relations?

- 3.1. Propose a possibly general yet precise definition of a production, applicable to a wide range of rewriting systems (for example, Chomsky grammars, graph grammars [13], shape grammars [72], and Koch constructions).
- 3.2. Biologists introduce a distinction between the *calendar age* of a plant, reflecting the objective progress of time, and *physiological age*, reflecting the state of plant development. Bell explains this distinction as follows [4, page 315],

A tree seedling in an open habitat may be one year old and rapidly approaching the juvenile state, whilst another individual of the same species growing in a closed habitat may be many years old, and held at the seedling state until light conditions improve.

Analyze the relationship between calendar age and physiological age in the context of simulating plant development.

- 3.3. Formulate a definition of a Koch construction, encompassing the different variants described by Mandelbrot [48]. Compare your definition with those proposed previously [54, 63].
- 3.4. Propose a modification of the Koch construction, capturing the derivation shown in Figure 4b.
- 4.1. Chien and Jürgensen [8] introduced a parametrized extension of L-systems called VD0L-systems. Compare this concept with the parametric L-systems described in Section 4.
- 4.2. Compare parametric L-systems and attribute grammars [33, 35].
- 4.3. Using the formal definition of a Koch construction obtained as a solution to Problem 3.3, investigate the relationship between the classes of figures and developmental sequences generated by Koch constructions and L-systems with turtle interpretation. Explain why many fractals can be generated using either method [51, 56, 61].
- 7.1. Propose a formal definition of derivation in L-systems with the cut symbol %.

7.2. In order to capture the phenomenon of dynamic equilibrium in a structure, Herman and Walker [28] (see also [66, pages 70–78]) introduced the notion of *adult languages*. By definition, a word  $w$  belongs to the adult language  $L_A(G)$  of an L-system  $G$  iff  $w$  is generated by  $G$  and it derives itself:  $w \implies w$ . Extend this definition to characterize biologically important situations where:

1. a part of the growing structure (such as the schematic tree crown shown in Figure 13) does not change over time, although other components of the structure may change;
  2. the entire structure or some part of it (such as the crown of the palm in Figure 14) undergoes a cyclic sequence of changes.
- 7.3. Propose a formal definition of L-systems suitable for modeling organisms that break up into separate structures. Assume that the organisms have branching topology. Extend turtle interpretation to properly handle the geometry of the resulting sets of structures.
- 7.4. Rozenberg, Ruohonen, and Salomaa [65] (see also [68, 69]) introduced *L-systems with fragmentation*, which can serve as a model of reproductive processes in plants. Compare L-systems with fragmentation with your solution to the previous problem.

- 8.1. Compare the notions of context-sensitivity, self-modifying code, and self-replication.
- 10.1. The following exercise concludes the first textbook on L-systems [27, page 341]:

Go out to a nearby field. Pick a flower. Simulate its development.

Solve this exercise for several plants. Identify simple and difficult components of the solutions. On this basis, propose areas for further research on the application of L-systems to plant modeling.

## 12 Acknowledgements

These notes incorporate edited versions of publications written with several co-authors. In this context, we acknowledge contributions by the late Professor Aristid Lindenmayer, and by Mark James. Many interesting ideas resulted from discussions with Dr. Peter Room; in particular, the idea of using L-systems for the simulation of interactions between plants and insects belongs to him. Lynn Mercer provided many useful comments, which we tried to incorporate into the final version of these course notes. The underlying research has been sponsored by grants and graduate scholarships from the Natural Sciences and Engineering Research Council of Canada.

## References

- [1] H. Abelson and A. A. diSessa. *Turtle geometry*. M.I.T. Press, Cambridge, 1982.
- [2] James Arvo and David Kirk. Modeling plants with environment-sensitive automata. In *Proceedings of Ausgraph'88*, pages 27 – 33, 1988.
- [3] R. Baker and G. T. Herman. Simulation of organisms using a developmental model, parts I and II. *Int. J. of Bio-Medical Computing*, 3:201–215 and 251–267, 1972.
- [4] A. Bell. *Plant form: An illustrated guide to flowering plants*. Oxford University Press, Oxford, 1991.
- [5] A. D. Bell, D. Roberts, and A. Smith. Branching patterns: the simulation of plant architecture. *Journal of Theoretical Biology*, 81:351–375, 1979.
- [6] R. Borchert and H. Honda. Control of development in the bifurcating branch system of *Tabebuia rosea*: A computer simulation. *Botanical Gazette*, 145(2):184–195, 1984.
- [7] R. Borchert and N. Slade. Bifurcation ratios and the adaptive geometry of trees. *Botanical Gazette*, 142(3):394–401, 1981.
- [8] T. W. Chien and H. Jürgensen. Parameterized L systems for modelling: Potential and limitations. In G. Rozenberg and A. Salomaa, editors, *Lindenmayer systems: Impacts on theoretical computer science, computer graphics, and developmental biology*, pages 213–229. Springer-Verlag, Berlin, 1992.
- [9] P. Coats. *Great gardens of the Western world*. G. P. Putnam's Sons, New York, 1963.
- [10] V. Darley. Emergent phenomena and complexity. In R. A. Brooks and P. Maes, editors, *Artificial Life IV. Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 411–416. MIT Press, Cambridge, 1994.
- [11] M. J. M. de Boer. *Analysis and computer generation of division patterns in cell layers using developmental algorithms*. PhD thesis, University of Utrecht, 1989.
- [12] M. J. M. de Boer, F. D. Fracchia, and P. Prusinkiewicz. A model for cellular development in morphogenetic fields. In G. Rozenberg and A. Salomaa, editors, *Lindenmayer systems: Impacts on theoretical computer science, computer graphics, and developmental biology*, pages 351–370. Springer-Verlag, Berlin, 1992.
- [13] H. Ehrig, M. Korff, and M. Löwe. Tutorial introduction to the algebraic approach of graph grammars based on double and single pushouts. In H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Graph grammars and their application to computer science; Fourth International Workshop*, Lecture Notes in Computer Science 532, pages 24–37. Springer-Verlag, Berlin, 1990.
- [14] J. D. Foley and A. Van Dam. *Fundamentals of interactive computer graphics*. Addison-Wesley, Reading, Massachusetts, 1982.

- [15] F. D. Fracchia, P. Prusinkiewicz, and M. J. M. de Boer. Animation of the development of multicellular structures. In N. Magnenat-Thalmann and D. Thalmann, editors, *Computer Animation '90*, pages 3–18, Tokyo, 1990. Springer-Verlag.
- [16] D. Frijters. Mechanisms of developmental integration of *Aster novae-angliae* L. and *Hieracium murorum* L. *Annals of Botany*, 42:561–575, 1978.
- [17] D. Frijters. Principles of simulation of inflorescence development. *Annals of Botany*, 42:549–560, 1978.
- [18] D. Frijters and A. Lindenmayer. A model for the growth and flowering of *Aster novae-angliae* on the basis of table (1,0)L-systems. In G. Rozenberg and A. Salomaa, editors, *L Systems*, Lecture Notes in Computer Science 15, pages 24–52. Springer-Verlag, Berlin, 1974.
- [19] D. Frijters and A. Lindenmayer. Developmental descriptions of branching patterns with paracladial relationships. In A. Lindenmayer and G. Rozenberg, editors, *Automata, languages, development*, pages 57–73. North-Holland, Amsterdam, 1976.
- [20] B. C. Goodwin. Generative explanations of plant form. In D. S. Ingram and A. Hudson, editors, *Shape and form in plant and fungi*, pages 3–16. Academic Press, London, 1994.
- [21] N. Greene. Organic architecture. SIGGRAPH Video Review 38, segment 16, ACM SIGGRAPH, New York, 1988.
- [22] N. Greene. Voxel space automata: Modeling with stochastic growth processes in voxel space. Proceedings of SIGGRAPH '89 (Boston, Mass., July 31–August 4, 1989), in *Computer Graphics* 23, 4 (August 1989), pages 175–184, ACM SIGGRAPH, New York, 1989.
- [23] F. Hallé, R. A. A. Oldeman, and P. B. Tomlinson. *Tropical trees and forests: An architectural analysis*. Springer-Verlag, Berlin, 1978.
- [24] J. S. Hanan. PLANTWORKS: A software system for realistic plant modelling. Master's thesis, University of Regina, 1988.
- [25] J. S. Hanan. *Parametric L-systems and their application to the modelling and visualization of plants*. PhD thesis, University of Regina, June 1992.
- [26] G. T. Herman and W. H. Liu. The daughter of CELIA, the French flag, and the firing squad. *Simulation*, 21:33–41, 1973.
- [27] G. T. Herman and G. Rozenberg. *Developmental systems and languages*. North-Holland, Amsterdam, 1975.
- [28] G. T. Herman and A. Walker. Context-free languages in biological systems. *International Journal of Computer Mathematics*, 4:369–391, 1975.

- [29] H. S. Horn. *The adaptive geometry of trees*. Princeton University Press, Princeton, 1971.
- [30] M. James, J. Hanan, and P. Prusinkiewicz. CPFG version 2.0 user's manual. Manuscript, Department of Computer Science, The University of Calgary, 1993, 50 pages.
- [31] J. M. Janssen and A. Lindenmayer. Models for the control of branch positions and flowering sequences of capitula in *Mycelis muralis* (L.) Dumont (Compositae). *New Phytologist*, 105:191–220, 1987.
- [32] B. W. Kernighan and D. M. Ritchie. *The C programming language. Second edition*. Prentice Hall, Englewood Cliffs, 1988.
- [33] D. E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):191–220, 1968.
- [34] A. J. Koch and H. Meinhardt. Biological pattern formation: from basic mechanisms to complex structures. Manuscript, Max-Planck-Institut für Entwicklungsbiologie, Tübingen, 1993.
- [35] P. M. Lewis II, D. J. Rosenkrantz, and R. E. Stearns. *Compiler design theory*. Addison-Wesley, Reading, 1978.
- [36] A. Lindenmayer. Mathematical models for cellular interaction in development, Parts I and II. *Journal of Theoretical Biology*, 18:280–315, 1968.
- [37] A. Lindenmayer. Developmental systems without cellular interaction, their languages and grammars. *Journal of Theoretical Biology*, 30:455–484, 1971.
- [38] A. Lindenmayer. Adding continuous components to L-systems. In G. Rozenberg and A. Salomaa, editors, *L Systems*, Lecture Notes in Computer Science 15, pages 53–68. Springer-Verlag, Berlin, 1974.
- [39] A. Lindenmayer. Developmental algorithms for multicellular organisms: A survey of L-systems. *Journal of Theoretical Biology*, 54:3–22, 1975.
- [40] A. Lindenmayer. Algorithms for plant morphogenesis. In R. Sattler, editor, *Theoretical plant morphology*, pages 37–81. Leiden University Press, The Hague, 1978.
- [41] A. Lindenmayer. Developmental algorithms: Lineage versus interactive control mechanisms. In S. Subtelny and P. B. Green, editors, *Developmental order: Its origin and regulation*, pages 219–245. Alan R. Liss, New York, 1982.
- [42] A. Lindenmayer. Positional and temporal control mechanisms in inflorescence development. In P. W. Barlow and D. J. Carr, editors, *Positional controls in plant development*. University Press, Cambridge, 1984.

- [43] A. Lindenmayer. Models for multicellular development: Characterization, inference and complexity of L-systems. In A. Kelemenová and J. Kelemen, editors, *Trends, techniques and problems in theoretical computer science*, Lecture Notes in Computer Science 281, pages 138–168. Springer-Verlag, Berlin, 1987.
- [44] A. Lindenmayer and H. Jürgensen. Grammars of development: Discrete-state models for growth, differentiation and gene expression in modular organisms. In G. Rozenberg and A. Salomaa, editors, *Lindenmayer systems: Impacts on theoretical computer science, computer graphics, and developmental biology*, pages 3–21. Springer-Verlag, Berlin, 1992.
- [45] A. Lindenmayer and P. Prusinkiewicz. Developmental models of multicellular organisms: A computer graphics perspective. In C. G. Langton, editor, *Artificial Life*, pages 221–249. Addison-Wesley, Redwood City, 1988.
- [46] H. B. Lück and J. Lück. Approche algorithmique des structures ramifiées acrotone et basitone des végétaux. In H. Vérine, editor, *La biologie théorique à Solignac*, pages 111–148. Polytechnica, Paris, 1994.
- [47] J. Lück, H. B. Lück, and M. Bakkali. A comprehensive model for acrotonic, mesotonic, and basitonic branching in plants. *Acta Biotheoretica*, 38:257–288, 1990.
- [48] B. B. Mandelbrot. *The fractal geometry of nature*. W. H. Freeman, San Francisco, 1982.
- [49] E. M. Meyerowitz. The genetics of flower development. *Scientific American*, pages 56–65, November 1994.
- [50] S. Papert. *Mindstorms: Children, computers and powerful ideas*. Basic Books, New York, 1980.
- [51] P. Prusinkiewicz. Graphical applications of L-systems. In *Proceedings of Graphics Interface '86 — Vision Interface '86*, pages 247–253, 1986.
- [52] P. Prusinkiewicz. Applications of L-systems to computer imagery. In H. Ehrig, M. Nagl, A. Rosenfeld, and G. Rozenberg, editors, *Graph grammars and their application to computer science; Third International Workshop*, pages 534–548. Springer-Verlag, Berlin, 1987. Lecture Notes in Computer Science 291.
- [53] P. Prusinkiewicz. Visual models of morphogenesis. *Artificial Life*, 1:61–74, 1994.
- [54] P. Prusinkiewicz and M. Hammel. Language-restricted iterated function systems, Koch constructions, and L-systems. In J. C. Hart, editor, *New directions for fractal modeling in computer graphics*, pages 4.1–4.14. ACM SIGGRAPH, 1994. Course Notes 13.
- [55] P. Prusinkiewicz, M. Hammel, and E. Mjolsness. Animation of plant development. *Computer Graphics (SIGGRAPH '93 Conference Proceedings)*, pages 351–360, August 1993.
- [56] P. Prusinkiewicz and J. Hanan. *Lindenmayer systems, fractals, and plants*, volume 79 of *Lecture Notes in Biomathematics*. Springer-Verlag, Berlin, 1989.

- [57] P. Prusinkiewicz and J. Hanan. Visualization of botanical structures and processes using parametric L-systems. In D. Thalmann, editor, *Scientific Visualization and Graphics Simulation*, pages 183–201. J. Wiley & Sons, Chichester, 1990.
- [58] P. Prusinkiewicz and J. Hanan. L-systems: From formalism to programming languages. In G. Rozenberg and A. Salomaa, editors, *Lindenmayer systems: Impacts on theoretical computer science, computer graphics, and developmental biology*, pages 193–211. Springer-Verlag, Berlin, 1992.
- [59] P. Prusinkiewicz, M. James, and R. Měch. Synthetic topiary. Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994), pages 351–358, ACM SIGGRAPH, New York, 1994.
- [60] P. Prusinkiewicz and L. Kari. Sub-apical L-systems. Technical Report 95/552/4, University of Calgary, March 1995.
- [61] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag, New York, 1990. With J. S. Hanan, F. D. Fracchia, D. R. Fowler, M. J. M. de Boer, and L. Mercer.
- [62] P. Prusinkiewicz, A. Lindenmayer, and J. Hanan. Developmental models of herbaceous plants for computer imagery purposes. Proceedings of SIGGRAPH '88 (Atlanta, Georgia, August 1–5, 1988), in *Computer Graphics* 22, 4 (August 1988), pages 141–150, ACM SIGGRAPH, New York, 1988.
- [63] P. Prusinkiewicz and G. Sandness. Koch curves as attractors and repellers. *IEEE Computer Graphics and Applications*, 8(6):26–40, November 1988.
- [64] P. M. Room, L. Maillette, and J. Hanan. Module and metamer dynamics and virtual plants. *Advances in Ecological Research*, 25:105–157, 1994.
- [65] G. Rozenberg, K. Ruohonen, and A. Salomaa. Developmental systems with fragmentation. *International Journal of Computer Mathematics*, 5:177–191, 1976.
- [66] G. Rozenberg and A. Salomaa. *The mathematical theory of L-systems*. Academic Press, New York, 1980.
- [67] G. Rozenberg and A. Salomaa. When L was young. In G. Rozenberg and A. Salomaa, editors, *The book of L*, pages 383–392. Springer-Verlag, Berlin, 1986.
- [68] K. Ruohonen. Developmental systems with interaction and fragmentation. *Information and Control*, 28:91–112, 1975.
- [69] K. Ruohonen. JL systems with non-fragmented axioms: the hierarchy. *International Journal of Computer Mathematics*, 5:143–156, 1975.
- [70] A. Salomaa. *Formal languages*. Academic Press, New York, 1973.

- [71] A. R. Smith. Plants, fractals, and formal languages. Proceedings of SIGGRAPH '84 (Minneapolis, Minnesota, July 22–27, 1984) in *Computer Graphics*, 18, 3 (July 1984), pages 1–10, ACM SIGGRAPH, New York, 1984.
- [72] G. Stiny. *Pictorial and formal aspects of shape and shape grammars*. Birkhäuser-Verlag, Basel and Stuttgart, 1975.
- [73] A. L. Szilard and R. E. Quinton. An interpretation for DOL systems by computer graphics. *The Science Terrapin*, 4:8–13, 1979.
- [74] C. E. Taylor. “Fleshing out” Artificial Life II. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II*, pages 25–38. Addison-Wesley, Redwood City, 1992.

# Visual Models of Plants Interacting with Their Environment

Radomír Měch and Przemysław Prusinkiewicz<sup>1</sup>

University of Calgary

## ABSTRACT

Interaction with the environment is a key factor affecting the development of plants and plant ecosystems. In this paper we introduce a modeling framework that makes it possible to simulate and visualize a wide range of interactions at the level of plant architecture. This framework extends the formalism of Lindenmayer systems with constructs needed to model bi-directional information exchange between plants and their environment. We illustrate the proposed framework with models and simulations that capture the development of tree branches limited by collisions, the colonizing growth of clonal plants competing for space in favorable areas, the interaction between roots competing for water in the soil, and the competition within and between trees for access to light. Computer animation and visualization techniques make it possible to better understand the modeled processes and lead to realistic images of plants within their environmental context.

**CR categories:** F.4.2 [Mathematical Logic and Formal Languages]: Grammars and Other Rewriting Systems: *Parallel rewriting systems*, I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism, I.6.3 [Simulation and Modeling]: Applications, J.3 [Life and Medical Sciences]: Biology.

**Keywords:** scientific visualization, realistic image synthesis, software design, L-system, modeling, simulation, ecosystem, plant development, clonal plant, root, tree.

## 1 INTRODUCTION

Computer modeling and visualization of plant development can be traced back to 1962, when Ulam applied cellular automata to simulate the development of branching patterns, thought of as an abstract representation of plants [53]. Subsequently, Cohen presented a more realistic model operating in continuous space [13], Linden-

<sup>1</sup>Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4 (mech|pwp@cpsc.ucalgary.ca)

mayer proposed the formalism of L-systems as a general framework for plant modeling [38, 39], and Honda introduced the first computer model of tree structures [32]. From these origins, plant modeling emerged as a vibrant area of interdisciplinary research, attracting the efforts of biologists, applied plant scientists, mathematicians, and computer scientists. Computer graphics, in particular, contributed a wide range of models and methods for synthesizing images of plants. See [18, 48, 54] for recent reviews of the main results.

One aspect of plant structure and behavior neglected by most models is the interaction between plants and their environment (including other plants). Indeed, the incorporation of interactions has been identified as one of the main outstanding problems in the domain of plant modeling [48] (see also [15, 18, 50]). Its solution is needed to construct predictive models suitable for applications ranging from computer-assisted landscape and garden design to the determination of crop and lumber yields in agriculture and forestry.

Using the information flow between a plant and its environment as the classification key, we can distinguish three forms of interaction and the associated models of plant-environment systems devised to date:

1. The plant is affected by global properties of the environment, such as day length controlling the initiation of flowering [23] and daily minimum and maximum temperatures modulating the growth rate [28].
2. The plant is affected by local properties of the environment, such as the presence of obstacles controlling the spread of grass [2] and directing the growth of tree roots [26], geometry of support for climbing plants [2, 25], soil resistance and temperature in various soil layers [16], and predefined geometry of surfaces to which plant branches are pruned [45].
3. The plant interacts with the environment in an information feedback loop, where the environment affects the plant and the plant reciprocally affects the environment. This type of interaction is related to *sighted* [4] or *exogenous* [42] mechanisms controlling plant development, in which parts of a plant influence the development of other parts of the same or a different plant through the space in which they grow. Specific models capture:
  - competition for *space* (including collision detection and access to light) between segments of essentially two-dimensional schematic branching structures [4, 13, 21, 22, 33, 34, 36];
  - competition between root tips for *nutrients* and *water* transported in soil [12, 37] (this mechanism is related to competition between growing branches of corals and sponges for nutrients diffusing in water [34]);

- competition for *light* between three-dimensional shoots of herbaceous plants [25] and branches of trees [9, 10, 11, 15, 33, 35, 52].

Models of exogenous phenomena require a comprehensive representation of both the developing plant and the environment. Consequently, they are the most difficult to formulate, implement, and document. Programs addressed to the biological audience are often limited to narrow groups of plants (for example, poplars [9] or trees in the pine family [21]), and present the results in a rudimentary graphical form. On the other hand, models addressed to the computer graphics audience use more advanced techniques for realistic image synthesis, but put little emphasis on the faithful reproduction of physiological mechanisms characteristic to specific plants.

In this paper we propose a general *framework* (defined as a modeling methodology supported by appropriate software) for modeling, simulating, and visualizing the development of plants that bi-directionally interact with their environment. The usefulness of modeling frameworks for simulation studies of models with complex (emergent) behavior is manifested by previous work in theoretical biology, artificial life, and computer graphics. Examples include cellular automata [51], systems for simulating behavior of cellular structures in discrete [1] and continuous [20] spaces, and L-system-based frameworks for modeling plants [36, 46]. Frameworks may have the form of a general-purpose simulation program that accepts models described in a suitable mini-language as input, *e.g.* [36, 46], or a set of library programs [27]. Compared to special-purpose programs, they offer the following benefits:

- At the conceptual level, they facilitate the design, specification, documentation, and comparison of models.
- At the level of model implementation, they make it possible to develop software that can be reused in various models. Specifically, graphical capabilities needed to visualize the models become a part of the modeling framework, and do not have to be reimplemented.
- Finally, flexible conceptual and software frameworks facilitate interactive experimentation with the models [46, Appendix A].

Our framework is intended both for purpose of image synthesis and as a research and visualization tool for model studies in plant morphogenesis and ecology. These goals are addressed at the levels of the simulation system and the modeling language design. The underlying paradigm of plant-environment interaction is described in Section 2. The resulting design of the simulation software is outlined in Section 3. The language for specifying plant models is presented in Section 4. It extends the concept of environmentally-sensitive L-systems [45] with constructs for bi-directional communication with the environment. The following sections illustrate the proposed framework with concrete models of plants interacting with their environment. The examples include: the development of planar branching systems controlled by the crowding of apices (Section 5), the development of clonal plants controlled by both the crowding of ramets and the quality of terrain (Section 6), the development of roots controlled by the concentration of water transported in the soil (Section 7), and the development of tree crowns affected by the local distribution of light (Section 8). The paper concludes with an evaluation of the results and a list of open problems (Section 9).

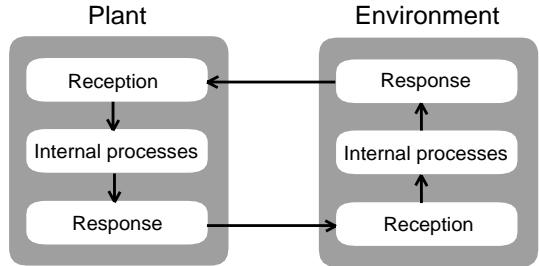


Figure 1: Conceptual model of plant and environment treated as communicating concurrent processes

## 2 CONCEPTUAL MODEL

As described by Hart [30], every environmentally controlled phenomenon can be considered as a chain of causally linked events. After a stimulus is perceived by the plant, information in some form is transported through the plant body (unless the site of stimulus perception coincides with the site of response), and the plant reacts. This reaction reciprocally affects the environment, causing its modification that in turn affects the plant. For example, roots growing in the soil can absorb or extract water (depending on the water concentration in their vicinity). This initiates a flow of water in the soil towards the depleted areas, which in turn affects further growth of the roots [12, 24].

According to this description, the interaction of a plant with the environment can be conceptualized as two concurrent processes that communicate with each other, thus forming a feedback loop of information flow (Figure 1). The plant process performs the following functions:

- reception of information about the environment in the form of scalar or vector values representing the stimuli perceived by specific organs;
- transport and processing of information inside the plant;
- generation of the response in the form of growth changes (*e.g.* development of new branches) and direct output of information to the environment (*e.g.* uptake and excretion of substances by a root tip).

Similarly, the environmental process includes mechanisms for the:

- perception of the plant’s actions;
- simulation of internal processes in the environment (*e.g.* the diffusion of substances or propagation of light);
- presentation of the modified environment in a form perceivable by the plant.

The design of a simulation system based on this conceptual model is presented next.

## 3 SYSTEM DESIGN

The goal is to create a framework, in which a wide range of plant structures and environments can be easily created, modified, and

used for experimentation. This requirement led us to the following design decisions:

- The plant and the environment should be modeled by separate programs and run as two communicating processes. This design is:
  - compatible with the assumed conceptual model of plant-environment interaction (Figure 1);
  - consistent with the principles of structured design (modules with clearly specified functions jointly contribute to the solution of a problem by communicating through a well defined interface; information local to each module is hidden from other modules);
  - appropriate for interactive experimentation with the models; in particular, changes in the plant program can be implemented without affecting the environmental program, and *vice versa*;
  - extensible to distributed computing environments, where different components of a large ecosystem may be simulated using separate computers.
- The user should have control over the type and amount of information exchanged between the processes representing the plant and the environment, so that all the needed but no superfluous information is transferred.
- Plant models should be specified in a language based on L-systems, equipped with constructs for bi-directional communication between the plant and the environment. This decision has the following rationale:
  - A succinct description of the models in an interpreted language facilitates experimentation involving modifications to the models;
  - L-systems capture two fundamental mechanisms that control development, namely flow of information from a mother module to its offspring (cellular descent) and flow of information between coexisting modules (endogenous interaction) [38]. The latter mechanism plays an essential role in transmitting information from the site of stimulus perception to the site of the response. Moreover, L-systems have been extended to allow for input of information from the environment (see Section 4);
  - Modeling of plants using L-systems has reached a relatively advanced state, manifested by models ranging from algae to herbaceous plants and trees [43, 46].
- Given the variety of processes that may take place in the environment, they should be modeled using special-purpose programs.
- Generic aspects of modeling, not specific to particular models, should be supported by the modeling system. This includes:
  - an L-system-based plant modeling program, which interprets L-systems supplied as its input and visualizes the results, and
  - the support for communication and synchronization of processes simulating the modeled plant and the environment.

A system architecture stemming from this design is shown in Figure 2. We will describe it from the perspective of extensions to the formalism of L-systems.

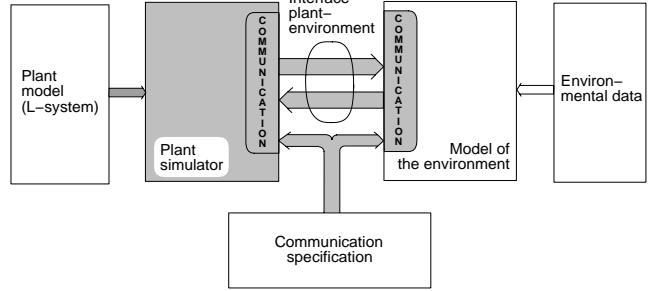


Figure 2: Organization of the software for modeling plants interacting with their environment. Shaded rectangles indicate components of the modeling framework, clear rectangles indicate programs and data that must be created by a user specifying a new model of a plant or environment. Shaded arrows indicate information exchanged in a standardized format.

## 4 OPEN L-SYSTEMS

Historically, L-systems were conceived as closed cybernetic systems, incapable of simulating any form of communication between the modeled plant and its environment. In the first step towards the inclusion of environmental factors, Rozenberg defined *table L-systems*, which allow for a change in the set of developmental rules (the production set of the L-system) in response to a change in the environment [31, 49]. Table L-systems were applied, for example, to capture the switch from the production of leaves to the production of flowers by the apex of a plant due to a change in day length [23]. *Parametric L-systems* [29, 46], introduced later, made it possible to implement a variant of this technique, with the environment affecting the model in a quantitative rather than qualitative manner. In a case study illustrating this possibility, weather data containing daily minimum and maximum temperatures were used to control the rate of growth in a bean model [28]. *Environmentally-sensitive L-systems* [45] represented the next step in the inclusion of environmental factors, in which local rather than global properties of the environment affected the model. The new concept was the introduction of query symbols, returning current position or orientation of the turtle in the underlying coordinate system. These parameters could be passed as arguments to user-defined functions, returning local properties of the environment at the queried location. Environmentally-sensitive L-systems were illustrated by models of topiary scenes. The environmental functions defined geometric shapes, to which trees were pruned.

*Open L-systems*, introduced in this paper, augment the functionality of environmentally-sensitive L-systems using a reserved symbol for bilateral communication with the environment. In short, parameters associated with an occurrence of the communication symbol can be set by the environment and transferred to the plant model, or set by the plant model and transferred to the environment. The environment is no longer represented by a simple function, but becomes an active process that may react to the information from the plant. Thus, plants are modeled as open cybernetic systems, sending information to and receiving information from the environment.

In order to describe open L-systems in more detail, we need to recall the rudiments of L-systems with turtle interpretation. Our presentation is reproduced from [45].

An L-system is a parallel rewriting system operating on branching structures represented as *bracketed strings* of symbols with associated numerical parameters, called *modules*. Matching pairs of square brackets enclose branches. Simulation begins with an initial string called the *axiom*, and proceeds in a sequence of discrete *derivation steps*. In each step, *rewriting rules* or *productions* replace all modules in the predecessor string by successor modules. The applicability of a production depends on a predecessor's context (in context-sensitive L-systems), values of parameters (in productions guarded by conditions), and on random factors (in stochastic L-systems). Typically, a production has the format:

$$id : lc < pred > rc : cond \rightarrow succ : prob$$

where *id* is the production identifier (label), *lc*, *pred*, and *rc* are the left context, the strict predecessor, and the right context, *cond* is the condition, *succ* is the successor, and *prob* is the probability of production application. The strict predecessor and the successor are the only mandatory fields. For example, the L-system given below consists of axiom  $\omega$  and three productions  $p_1$ ,  $p_2$ , and  $p_3$ .

$$\begin{aligned} \omega: & A(1)B(3)A(5) \\ p_1: & A(x) \rightarrow A(x+1) : 0.4 \\ p_2: & A(x) \rightarrow B(x-1) : 0.6 \\ p_3: & A(x) < B(y) > A(z) : y < 4 \rightarrow B(x+z)[A(y)] \end{aligned}$$

The stochastic productions  $p_1$  and  $p_2$  replace module  $A(x)$  by either  $A(x+1)$  or  $B(x-1)$ , with probabilities equal to 0.4 and 0.6, respectively. The context-sensitive production  $p_3$  replaces a module  $B(y)$  with left context  $A(x)$  and right context  $A(z)$  by module  $B(x+z)$  supporting branch  $A(y)$ . The application of this production is guarded by condition  $y < 4$ . Consequently, the first derivation step may have the form:

$$A(1)B(3)A(5) \Rightarrow A(2)B(6)[A(3)]B(4)$$

It was assumed that, as a result of random choice, production  $p_1$  was applied to the module  $A(1)$ , and production  $p_2$  to the module  $A(5)$ . Production  $p_3$  was applied to the module  $B(3)$ , because it occurred with the required left and right context, and the condition  $3 < 4$  was true.

In the L-systems presented as examples we also use several additional constructs (*cf.* [29, 44]):

- Productions may include statements assigning values to local variables. These statements are enclosed in curly braces and separated by semicolons.
- The L-systems may also include arrays. References to array elements follow the syntax of C; for example, *MaxLen[order]*.
- The list of productions is ordered. In the deterministic case, the first matching production applies. In the stochastic case, the set of all matching productions is established, and one of them is chosen according to the specified probabilities.

For details of the L-system syntax see [29, 43, 46].

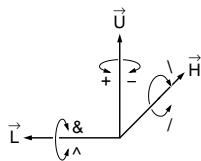


Figure 3: Controlling the turtle in three dimensions

In contrast to the parallel application of productions in each derivation step, the interpretation of the resulting strings proceeds sequentially, with reserved modules acting as commands to a LOGO-style turtle [46]. At any point of the string, the *turtle state* is characterized by a position vector  $\vec{P}$  and

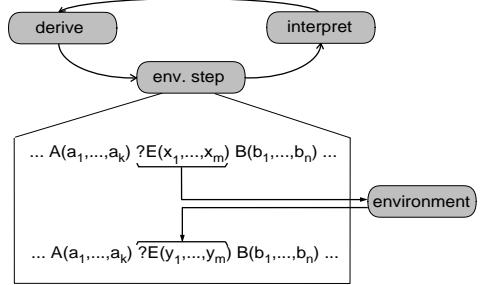


Figure 4: Information flow during the simulation of a plant interacting with the environment, implemented using an open L-system

three mutually perpendicular orientation vectors  $\vec{H}$ ,  $\vec{U}$ , and  $\vec{L}$ , indicating the turtle's heading, the up direction, and the direction to the left (Figure 3). Module  $F$  causes the turtle to draw a line in the current direction. Modules  $+$ ,  $-$ ,  $\&$ ,  $\wedge$ ,  $/$  and  $\backslash$  rotate the turtle around one of the vectors  $\vec{H}$ ,  $\vec{U}$ , or  $\vec{L}$ , as shown in Figure 3. The length of the line and the magnitude of the rotation angle can be given globally or specified as parameters of individual modules. During the interpretation of branches, the opening square bracket pushes the current position and orientation of the turtle on a stack, and the closing bracket restores the turtle to the position and orientation popped from the stack. A special interpretation is reserved for the module  $\%$ , which cuts a branch by erasing all symbols in the string from the point of its occurrence to the end of the branch [29]. The meaning of many symbols depends on the context in which they occur; for example,  $+$  and  $-$  denote arithmetic operators as well as modules that rotate the turtle.

The turtle interpretation of L-systems described above was designed to visualize models in a postprocessing step, with no effect on the L-system operation. Position and orientation of the turtle are important, however, while considering environmental phenomena, such as collisions with obstacles and exposure to light. The environmentally-sensitive extension of L-systems makes these attributes accessible during the rewriting process [45]. The generated string is interpreted after each derivation step, and turtle attributes found during the interpretation are returned as parameters to reserved *query modules*. Syntactically, the query modules have the form  $?X(x, y, z)$ , where  $X = P, H, U$ , or  $L$ . Depending on the actual symbol  $X$ , the values of parameters  $x$ ,  $y$ , and  $z$  represent a position or an orientation vector.

Open L-systems are a generalization of this concept. *Communication modules* of the form  $?E(x_1, \dots, x_m)$  are used both to send and receive environmental information represented by the values of parameters  $x_1, \dots, x_m$  (Figure 4). To this end, the string resulting from a derivation step is scanned from left to right to determine the state of the turtle associated with each symbol. This phase is similar to the graphical interpretation of the string, except that the results need not be visualized. Upon encountering a communication symbol, the plant process creates and sends a message to the environment including all or a part of the following information:

- the address (position in the string) of the communication module (mandatory field needed to identify this module when a reply comes from the environment),
- values of parameters  $x_i$ ,
- the state of the turtle (coordinates of the position and orientation

vector, as well as some other attributes, such as current line width),

- the type and parameters of the module following the communication module in the string (not used in the examples discussed in this paper).

The exact message format is defined in a *communication specification file*, shared between the programs modeling the plant and the environment (Figure 2). Consequently, it is possible to include only the information needed in a particular model in the messages sent to the environment. Transfer of the last message corresponding to the current scan of the string is signaled by a reserved end-of-transmission message, which may be used by the environmental process to start its operation.

The messages output by the plant modeling program are transferred to the process that simulates the environment using an interprocess communication mechanism provided by the underlying operating system (a pair of UNIX pipes or shared memory with access synchronized using semaphores, for example). The environment processes that information and returns the results to the plant model using messages in the following format:

- the address of the target communication module,
- values of parameters  $y_i$  carrying the output from the environment.

The plant process uses the received information to set parameter values in the communication modules (Figure 4). The use of addresses makes it possible to send replies only to selected communication modules. Details of the mapping of messages received by the plant process to the parameters of the communication modules are defined in the communication specification file.

After all replies generated by the environment have been received (a fact indicated by an end-of-transmission message sent by the environment), the resulting string may be interpreted and visualized, and the next derivation step may be performed, initiating another cycle of the simulation.

Note that, by preceding every symbol in the string with a communication module it is possible to pass complete information about the model to the environment. Usually, however, only partial information about the state of a plant is needed as input to the environment. Proper placement of communication modules in the model, combined with careful selection of the information to be exchanged, provide a means for keeping the amount of transferred information at a manageable level.

We will illustrate the operation of open L-systems within the context of complete models of plant-environment interactions, using examples motivated by actual biological problems.

## 5 A MODEL OF BRANCH TIERS

**Background.** Apical meristems, located at the endpoints of branches, are engines of plant development. The apices grow, contributing to the elongation of branch segments, and from time to time divide, spawning the development of new branches. If all apices divided periodically, the number of apices and branch segments would increase exponentially. Observations of real branching structures show, however, that the increase in the number of segments is less than exponential [8]. Honda and his collaborators modeled several hypothetical mechanisms that may control the extent of

branching in order to prevent overcrowding [7, 33] (see also [4]). One of the models [33], supported by measurements and earlier simulations of the tropical tree *Terminalia catappa* [19], assumes an exogenous interaction mechanism. *Terminalia* branches form horizontal tiers, and the model is limited to a single tier, treated as a two-dimensional structure. In this case, the competition for light effectively amounts to collision detection between the apices and leaf clusters. We reproduce this model as the simplest example illustrating the methodology proposed in this paper.

**Communication specification.** The plant communicates with the environment using communication modules of the form  $?E(x)$ . Messages sent to the environment include the turtle position and the value of parameter  $x$ , interpreted as the vigor of the corresponding apex. On this basis, the environmental process determines the fate of each apex. A parameter value of  $x = 0$  returned to the plant indicates that the development of the corresponding branch will be terminated. A value of  $x = 1$  allows for further branching.

**The model of the environment.** The environmental process considers each apex or non-terminal node of the developing tier as the center of a circular leaf cluster, and maintains a list of all clusters present. New clusters are added in response to messages received from the plant. All clusters have the same radius  $\rho$ , specified in the environmental data file (*cf.* Figure 2). In order to determine the fate of the apices, the environment compares apex positions with leaf cluster positions, and authorizes an apex to grow if it does not fall into an existing leaf cluster, or if it falls into a cluster surrounding an apex with a smaller vigor value.

**The plant model.** The plant model is expressed as an open L-system. The values of constants are taken from [33].

```
#define r1 0.94 /* contraction ratio and vigor 1 */
#define r2 0.87 /* contraction ratio and vigor 2 */
#define α1 24.4 /* branching angle 1 */
#define α2 36.9 /* branching angle 2 */
#define φ 138.5 /* divergence angle */
ω: -(90)[F(1)?E(1)A(1)]+(φ)[F(1)/?E(1)A(1)]
+(φ)[F(1)?E(1)A(1)]+(φ)[F(1)/?E(1)A(1)]
+(φ)[F(1)?E(1)A(1)]

p1: ?E(x) < A(v) : x == 1 →
[+(α2)F(v*r2)?E(r2)A(v*r2)]-(α1)F(v*r1)/?E(r1)A(v*r1)
p2: ?E(x) → ε
```

The axiom  $\omega$  specifies the initial structure as a whorl of five branch segments  $F$ . The divergence angle  $\varphi$  between consecutive segments is equal to  $138.5^\circ$ . Each segment is terminated by a communication symbol  $?E$  followed by an apex  $A$ . In addition, two branches include module  $/$ , which changes the directions at which subsequent branches will be issued (left vs. right) by rotating the apex  $180^\circ$  around the segment axis.

Production  $p_1$  describes the operation of the apices. If the value of parameter  $x$  returned by a communication module  $?E$  is not 1, the associated apex will remain inactive (do nothing). Otherwise the apex will produce a pair of new branch segments at angles  $\alpha_1$  and  $\alpha_2$  with respect to the mother segment. Constants  $r_1$  and  $r_2$  determine the lengths of the daughter segments as fractions of the length of their mother segment. The values  $r_1$  and  $r_2$  are also passed to the process simulating the environment using communication modules  $?E$ . Communication modules created in the previous derivation step are no longer needed and are removed by production  $p_2$ .

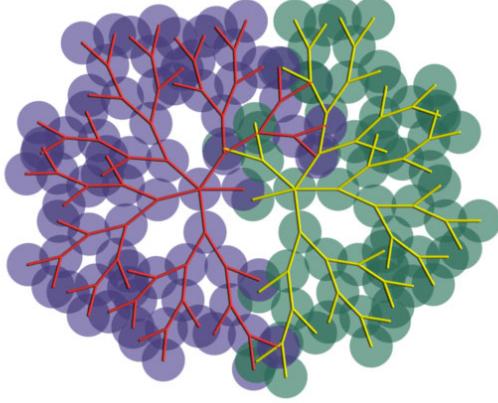


Figure 5: Competition for space between two tiers of branches simulated using the Honda model

**Simulation.** Figure 5 illustrates the competition for space between two tiers developing next to each other. The extent of branching in each tier is limited by collisions between its apices and its own or the neighbor’s leaf clusters. The limited growth of each structure in the direction of its neighbor illustrates the phenomenon of *morphological plasticity*, or adaptation of the form of plants to their environment [17].

## 6 A MODEL OF FORAGING IN CLONAL PLANTS

**Background.** Foraging (propagation) patterns in clonal plants provide another excellent example of response to crowding. A clonal plant spreads by means of horizontal stem segments (*spacers*), which form a branching structure that grows along the ground and connects individual plants (*ramets*) [3]. Each ramet consists of a leaf supported by an upright stem and one or more buds, which may give rise to further spacers and ramets. Their gradual death, after a certain amount of time, causes gradual separation of the whole structure (the *clone*) into independent parts.

Following the surface of the soil, clonal plants can be captured using models operating in two dimensions [5], and in that respect resemble *Terminalia* tiers. We propose a model of a hypothetical plant that responds to favorable environmental conditions (high local intensity of light) by more extensive branching and reduced size of leaves (allowing for more dense packing of ramets). It has been inspired by a computer model of clover outlined by Bell [4], the analysis of responses of clonal plants to the environment presented by Dong [17], and the computer models and descriptions of vegetative multiplication of plants involving the death of intervening connections by Room [47].

**Communication specification.** The plant sends messages to the environment that include turtle position and two parameters associated with the communications symbol,  $?E(type, x)$ . The first parameter is equal to 0, 1, or 2, and determines the type of exchanged information as follows:

- The message  $?E(0, x)$  represents a request for the light intensity (irradiance [14]) at the position of the communication module.

The environment responds by setting  $x$  to the intensity of incoming light, ranging from 0 (no light) to 1 (full light).

- The message  $?E(1, x)$  notifies the environment about the creation of a ramet with a leaf of radius  $x$  at the position of the communication module. No output is generated by the environment in response to this message.
- The message  $?E(2, x)$  notifies the environment about the death of a ramet with a leaf of radius  $x$  at the position of the communication module. Again, no output is generated by the environment.

**The model of the environment.** The purpose of the environment process is to determine light intensity at the locations requested by the plant. The ground is divided into patches (specified as a raster image using a paint program), with different light intensities assigned to each patch. In the absence of shading, these intensities are returned by the environmental process in response to messages of type 0. To consider shading, the environment keeps track of the set of ramets, adding new ramets in response to a messages of type 1, and deleting dead ramets in response to messages of type 2. If a sampling point falls in an area occupied by a ramet, the returned light intensity is equal to 0 (leaves are assumed to be opaque, and located above the sampling points).

**The plant model.** The essential features of the plant model are specified by the following open L-system.

```
#define α 45           /* branching angle */
#define MinLight 0.1    /* light intensity threshold */
#define MaxAge 20       /* lifetime of ramets and spacers */
#define Len 2.0          /* length of spacers */
#define Prob_B(x)        (0.12+x*0.42)
#define Prob_R(x)        (0.03+x*0.54)
#define Radius(x)        (sqrt(15-x*5)/π)

ω: A(1)?E(0,0)

p1: A(dir) > ?E(0,x) : x >= MinLight
      → R(x)B(x,dir)F(Len,0)A(-dir)?E(0,0)
p2: A(dir) > ?E(0,x) : x < MinLight → ε

p3: B(x,dir) → [+ (α*dir)F(Len,0)A(-dir)?E(0,0)] : Prob_B(x)
p4: B(x,dir) → ε : 1-Prob_B(x)

p5: R(x) → [@o(Radius(x),0)?E(1,Radius(x))] : Prob_R(x)
p6: R(x) → ε : 1-Prob_R(x)

p7: @o(radius,age): age < MaxAge → @o(radius,age+1)
p8: @o(radius,age): age == MaxAge → ?E(2,radius)

p9: F(len,age): age < MaxAge → F(len,age+1)
p10: F(len,age): age == MaxAge → f(len)

p11: ?E(type,x) → ε
```

The initial structure specified by the axiom  $\omega$  consists of an apex  $A$  followed by the communication module  $?E$ . If the intensity of light  $x$  reaching an apex is insufficient (below the threshold  $MinLight$ ), the apex dies (production  $p_2$ ). Otherwise, the apex creates a ramet initial  $R$  (i.e., a module that will yield a ramet), a branch initial  $B$ , a spacer  $F$ , and a new apex  $A$  terminated by communication module  $?E$  (production  $p_1$ ). The parameter  $dir$ , valued either 1 or -1, controls the direction of branching. Parameters of the spacer module specify its length and age.

A branch initial  $B$  may create a lateral branch with its own apex  $A$  and communication module  $?E$  (production  $p_3$ ), or it may die and disappear from the system (production  $p_4$ ). The probability of survival is an increasing linear function  $Prob_B$  of the light intensity  $x$  that has reached the mother apex  $A$  in the previous derivation step. A similar stochastic mechanism describes the production of a ramet by the ramet initial  $R$  (productions  $p_5$  and  $p_6$ ), with the probability of ramet formation controlled by an increasing linear function  $Prob_R$ . The ramet is represented as a circle  $@o$ ; its radius is a decreasing function  $Radius$  of the light intensity  $x$ . As in the case of spacers, the second parameter of a ramet indicates its age, initially set to 0. The environment is notified about the creation of the ramet using a communication module  $?E$ .

The subsequent productions describe the aging of spacers ( $p_7$ ) and ramets ( $p_9$ ). Upon reaching the maximum age  $MaxAge$ , a ramet is removed from the system and a message notifying the environment about this fact is sent by the plant ( $p_8$ ). The death of the spacers is simulated by replacing spacer modules  $F$  with invisible line segments  $f$  of the same length. This replacement maintains the relative position of the remaining elements of the structure. Finally, production  $p_{11}$  removes communication modules after they have performed their tasks.

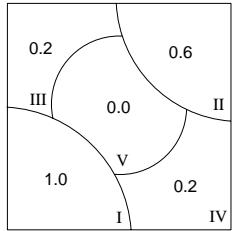


Figure 6: Division of the ground into patches

Figure 6: Division of the ground into patches varied using random functions with a normal distribution. Ramets have been represented as trifoliate leaves.

The development begins with a single ramet located in relatively good (light intensity 0.6) patch II at the top right corner of the growth area (Figure 7, step 9 of the simulation). The plant propagates through the unfavorable patch III without producing many branches and leaves (step 26), and reaches the best patch I at the bottom left corner (step 39). After quickly spreading over this patch (step 51), the plant searches for further favorable areas (step 62). The first attempt to reach patch II fails (step 82). The plant tries again, and this time succeeds (steps 101 and 116). Light conditions in patch II are not sufficient, however, to sustain the continuous presence of the plant (step 134). The colony disappears (step 153) until the patch is reached again by a new wave of propagation (steps 161 and 182).

The sustained occupation of patch I and the repetitive invasion of patch II represent an emerging behavior of the model, difficult to predict without running simulations. Variants of this model, including other branching architectures, responses to the environment, and layouts of patches in the environment, would make it possible to analyze different foraging strategies of clonal plants. A further extension could replace the empirical assumptions regarding plant responses with a more detailed simulation of plant physiology (for example, including production of photosynthates and their transport and partition between ramets). Such physiological models could provide insight into the extent to which the foraging patterns optimize plants' access to resources [17].

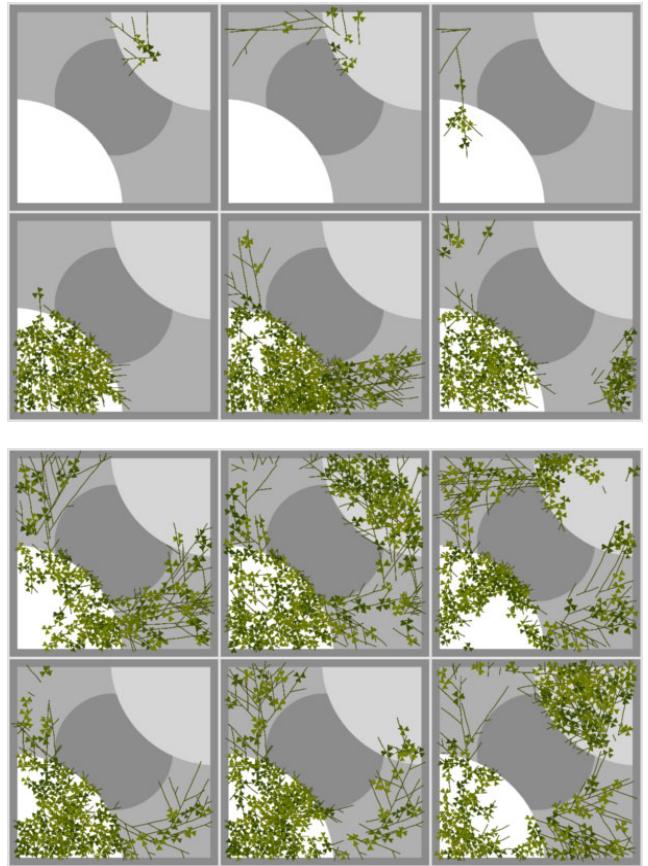


Figure 7: Development of a hypothetical clonal plant simulated using an extension of L-system 3. The individual images represent structures generated in 9, 26, 39, 51, 62, and 82 derivation steps (top), followed by structures generated in 101, 116, 134, 153, 161, and 182 steps (bottom).

## 7 A MODEL OF ROOT DEVELOPMENT

**Background.** The development of roots provides many examples of complex interactions with the environment, which involve mechanical properties, chemical reactions, and transport mechanisms in the soil. In particular, the main root and the rootlets absorb water from the soil, locally changing its concentration (volume of water per unit volume of soil) and causing water motion from water-rich to depleted regions [24]. The tips of the roots, in turn, follow the gradient of water concentration [12], thus adapting to the environment modified by their own activities.

Below we present a simplified implementation of the model of root development originally proposed by Clausnitzer and Hopmans [12]. We assume a more rudimentary mechanism of water transport, namely diffusion in a uniform medium, as suggested by Liddell and Hansen [37]. The underlying model of root architecture is similar to that proposed by Diggle [16]. For simplicity, we focus on model operation in two-dimensions.

**Communication specification.** The plant interacts with the environment using communication modules  $?E(c, \theta)$  located at the apices of the root system. A message sent to the environment includes the turtle position  $\vec{P}$ , the heading vector  $\vec{H}$ , the value of

parameter  $c$  representing the requested (optimal) water uptake, and the value of parameter  $\theta$  representing the tendency of the apex to follow the gradient of water concentration. A message returned to the plant specifies the amount of water actually received by the apex as the value of parameter  $c$ , and the angle biasing direction of further growth as the value of  $\theta$ .

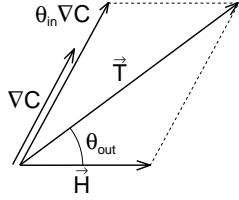


Figure 8: Definition of the biasing angle  $\theta_{out}$

### The model of the environment.

The environment maintains a field  $C$  of water concentrations, represented as an array of the amounts of water in square sampling areas. Water is transported by diffusion, simulated numerically using finite differencing [41]. The environment responds to a request for water from an apex located in an area  $(i, j)$  by granting the lesser of the

values requested and available at that location. The amount of water in the sampled area is then decreased by the amount received by the apex. The environment also calculates a linear combination  $\vec{T}$  of the turtle heading vector  $\vec{H}$  and the gradient of water concentration  $\nabla C$  (estimated numerically from the water concentrations in the sampled area and its neighbors), and returns an angle  $\theta$  between the vectors  $\vec{T}$  and  $\vec{H}$  (Figure 8). This angle is used by the plant model to bias turtle heading in the direction of high water concentration.

**The root model.** The open L-system representing the root model makes use of arrays that specify parameters for each branching order (main axis, its daughter axes, *etc.*). The parameter values are loosely based on those reported by Clausnitzer and Hopmans [12].

```
#define N 3                                /* max. branching order + 1 */
Define: { array
Req[N] = {0.1, 0.4, 0.05},                /* requested nutrient intake */
MinReq[N] = {0.01, 0.06, 0.01},            /* minimum nutrient intake */
ElRate[N] = {0.55, 0.25, 0.55},           /* maximum elongation rate */
MaxLen[N] = {200, 5, 0.8},                 /* maximum branch length */
Sens[N] = {10, 0, 0},                      /* sensitivity to gradient */
Dev[N] = {30, 75, 75},                     /* deviation in heading */
Del[N-1] = {30, 60},                       /* delay in branch growth */
BrAngle[N-1] = {90, 90},                   /* branching angle */
BrSpace[N-1] = {1, 0.5}                    /* distance between branches */
}
ω: A(0,0,0)?E(Req[0],Sens[0])
```

$p_1$ :  $A(n,s,b) > ?E(c,\theta) : (s > \text{MaxLen}[n]) \parallel (c < \text{MinReq}[n]) \rightarrow \varepsilon$   
 $p_2$ :  $A(n,s,b) > ?E(c,\theta) :$   
 $\quad (n \geq N-1) \parallel (b < \text{BrSpace}[n]) \{ h=c/\text{Req}[n]*\text{ElRate}[n]; \}$   
 $\quad \rightarrow +(nran(\theta,\text{Dev}[n]))F(h) A(n,s+h,b+h)?E(\text{Req}[n],\text{Sens}[n])$   
 $p_3$ :  $A(n,s,b) > ?E(c,\theta) :$   
 $\quad (n < N-1) \&& (b \geq \text{BrSpace}[n]) \{ h=c/\text{Req}[n]*\text{ElRate}[n]; \}$   
 $\quad \rightarrow +(nran(\theta,\text{Dev}[n]))B(n,0)F(h)$   
 $\quad /(180)A(n,s+h,h)?E(\text{Req}[n],\text{Sens}[n])$   
 $p_4$ :  $B(n,t) : t < \text{Del}[n] \rightarrow B(n,t+1)$   
 $p_5$ :  $B(n,t) : t \geq \text{Del}[n] \rightarrow [+(\text{BrAngle}[n])A(n+1,0,0)?E(\text{Req}[n+1],\text{Sens}[n+1])]$   
 $p_6$ :  $?E(c,\theta) \rightarrow \varepsilon$

The development starts with an apex  $A$  followed by a communication module  $?E$ . The parameters of the apex represent the branch order (0 for the main axis, 1 for its daughter axes, *etc.*), current axis length, and distance (along the axis) to the nearest branching point.

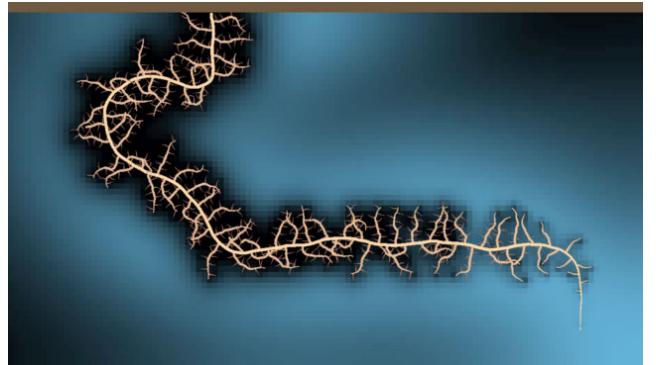


Figure 9: A two-dimensional model of a root interacting with water in soil. Background colors represent concentrations of water diffusing in soil (blue: high, black: low). The initial and boundary values have been set using a paint program.

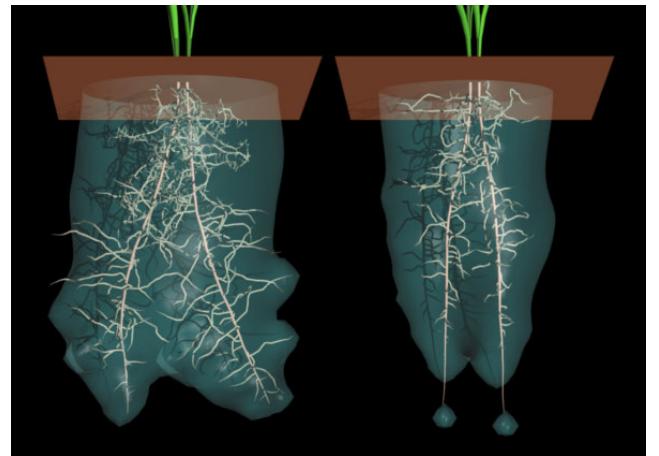


Figure 10: A three-dimensional extension of the root model. Water concentration is visualized by semi-transparent iso-surfaces [55] surrounding the roots. As a result of competition for water, the roots grow away from each other. The divergence between their main axes depends on the spread of the rootlets, which grow faster on the left than on the right.

Productions  $p_1$  to  $p_3$  describe possible fates of the apex as described below.

If the length  $s$  of a branch axis exceeds a predefined maximum value  $\text{MaxLen}[n]$  characteristic to the branch order  $n$ , or the amount of water  $c$  received by the apex is below the required minimum  $\text{MinReq}[n]$ , the apex dies, terminating the growth of the axis (production  $p_1$ ).

If the branch order  $n$  is equal to the maximum value assumed in the model ( $N - 1$ ), or the distance  $b$  to the closest branching point on the axis is less than the threshold value  $\text{BrSpace}[n]$ , the apex adds a new segment  $F$  to the axis (production  $p_2$ ). The length  $h$  of  $F$  is the product of the nominal growth increment  $\text{ElRate}[n]$  and the ratio of the amount of water received by the apex  $c$  to the amount requested  $\text{Req}[n]$ . The new segment is rotated with respect to its predecessor by an angle  $nran(\theta, \text{Dev}[n])$ , where  $nran$  is a random function with a normal distribution. The mean value  $\theta$ , returned by the environment, biases the direction of growth towards regions of

higher water concentration. The standard deviation  $Dev[n]$  characterizes the tendency of the root apex to change direction due to various factors not included explicitly in the model.

If the branch order  $n$  is less than the maximum value assumed in the model ( $N - 1$ ), and the distance  $b$  to the closest branching point on the axis is equal to or exceeds the threshold value  $BrSpace[n]$ , the apex creates a new branch initial  $B$  (production  $p_3$ ). Other aspects of apex behavior are the same as those described by production  $p_2$ .

After the delay of  $Del[n]$  steps (production  $p_4$ ), the branch initial  $B$  is transformed into an apex  $A$  followed by the communication module  $?E$  (production  $p_5$ ), giving rise to a new root branch. Production  $p_6$  removes communication modules that are no longer needed.

**Simulations.** A sample two-dimensional structure obtained using the described model is shown in Figure 9. The apex of the main axis follows the gradient of water concentration, with small deviations due to random factors. The apices of higher-order axes are not sensitive to the gradient and change direction at random, with a larger standard deviation. The absorption of water by the root and the rootlets decreases water concentration in their neighborhood; an effect that is not fully compensated by water diffusion from the water-rich areas. Low water concentration stops the development of some rootlets before they have reached their potential full length.

Figure 10 presents a three-dimensional extension of the previous model. As a result of competition for water, the main axes of the roots diverge from each other (left). If their rootlets grow more slowly, the area of influence of each root system is smaller and the main axes are closer to each other (right). This behavior is an emergent property of interactions between the root modules, mediated by the environment.

## 8 MODELS OF TREES CONTROLLED BY LIGHT

**Background.** Light is one of the most important factors affecting the development of plants. In the essentially two-dimensional structures discussed in Section 5, competition for light could be considered in a manner similar to collision detection between leaves and apices. In contrast, competition for light in three-dimensional structures must be viewed as long-range interaction. Specifically, shadows cast by one branch may affect other branches at significant distances.

The first simulations of plant development that take the local light environment into account are due to Greene [25]. He considered the entire sky hemisphere as a source of illumination and computed the amount of light reaching specific points of the structure by casting rays towards a number of points on the hemisphere. Another approach was implemented by Kanamaru *et al.* [35], who computed the amount of light reaching a given sampling point by considering it a center of projection, from which all leaf clusters in a tree were projected on a surrounding hemisphere. The degree to which the hemisphere was covered by the projected clusters indicated the amount of light received by the sampling point. In both cases, the models of plants responded to the amount and the direction of light by simulating heliotropism, which biased the direction of growth towards the vector of the highest intensity of incoming light. Subsequently, Chiba *et al.* extended the models by Kanamaru *et al.* using more involved tree models that included a mechanism simulating the flow of hypothetical endogenous information within the tree [10, 11]. A biologically better justified model, formulated

in terms of production and use of photosynthates by a tree, was proposed by Takenaka [52]. The amount of light reaching leaf clusters was calculated by sampling a sky hemisphere, as in the work by Greene. Below we reproduce the main features of the Takenaka's model using the formalism of open L-systems. Depending on the underlying tree architecture, it can be applied to synthesize images of deciduous and coniferous trees. We focus on a deciduous tree, which requires a slightly smaller number of productions.

**Communication specification.** The plant interacts with the environment using communication modules  $?E(r)$ . A message sent by the plant includes turtle position  $\vec{P}$ , which represents the center of a spherical leaf cluster, and the value of parameter  $r$ , which represents the cluster's radius. The environment responds by setting  $r$  to the flux [14] of light from the sky hemisphere, reaching the cluster.

**The model of the environment.** Once all messages describing the current distribution of leaves on a tree have been received, the environmental process computes the extent of the tree in the  $x$ ,  $y$ , and  $z$  directions, encompasses the tree in a tight grid ( $32 \times 32 \times 32$  voxels in our simulations), and allocates leaf clusters to voxels to speed up further computations. The environmental process then estimates the light flux  $\Phi$  from the sky hemisphere reaching each cluster (shadows cast by the branches are ignored). To this end, the hemisphere is represented by a set of directional light sources  $S$  (9 in the simulations). The flux densities (radiosities)  $B$  of the sources approximate the non-uniform distribution of light from the sky (cf. [52]). For each leaf cluster  $L_i$  and each light source  $S$ , the environment determines the set of leaf clusters  $L_j$  that may shade  $L_i$ . This is achieved by casting a ray from the center of  $L_i$  in the direction of  $S$  and testing for intersections with other clusters (the grid accelerates this process). In order not to miss any clusters that may partially occlude  $L_i$ , the radius of each cluster  $L_j$  is increased by the maximum value of cluster radius  $r_{max}$ .

To calculate the flux reaching cluster  $L_i$ , this cluster and all clusters  $L_j$  that may shade it according to the described tests are projected on a plane  $P$  perpendicular to the direction of light from the source  $S$ . The impact of a cluster  $L_j$  on the flux  $\Phi$  reaching cluster  $L_i$  is then computed according to the formula:

$$\Phi = (A_i - A_{ij})B + A_{ij}\tau B$$

where  $A_i$  is the area of the projection of  $L_i$  on  $P$ ,  $A_{ij}$  is the area of the intersection between projections of  $L_i$  and  $L_j$ , and  $\tau$  is the light transmittance through leaf cluster  $L_j$  (equal to 0.25 in the simulations). If several clusters  $L_j$  shade  $L_i$ , their influences are multiplied. The total flux reaching cluster  $L_i$  is calculated as the sum of the fluxes received from each light source  $S$ .

**The plant model.** In addition to the communication module  $?E$ , the plant model includes the following types of modules:

- Apex  $A(vig, del)$ . Parameter  $vig$  represents vigor, which determines the length of branch segments (internodes) and the diameter of leaf clusters produced by the apex. Parameter  $del$  is used to introduce a delay, needed for propagating products of photosynthesis through the tree structure between consecutive stages of development (years).
- Leaf  $L(vig, p, age, del)$ . Parameters denote the leaf radius  $vig$ , the amount of photosynthates produced in unit time according to the leaf's exposure to light  $p$ , the number of years for which a leaf has appeared at a given location  $age$ , and the delay  $del$ , which plays the same role as in the apices.
- Internode  $F(vig)$ . Consistent with the turtle interpretation, the

parameter  $vig$  indicates the internode length.

- Branch width symbol  $!(w, p, n)$ , also used to carry the endogenous information flow. The parameters determine: the width of the following internode  $w$ , the amount of photosynthates reaching the symbol's location  $p$ , and the number of terminal branch segments above this location  $n$ .

The corresponding L-system is given below.

```
#define φ 137.5 /* divergence angle */
#define α₀ 5 /* direction change - no branching */
#define α₁ 20 /* branching angle - main axis */
#define α₂ 32 /* branching angle - lateral axis */
#define W 0.02 /* initial branch width */
#define VD 0.95 /* apex vigor decrement */
#define Del 30 /* delay */
#define LS 5 /* how long a leaf stays */
#define LP 8 /* full photosynthate production */
#define LM 2 /* leaf maintenance */
#define PB 0.8 /* photosynthates needed for branching */
#define PG 0.4 /* photosynthates needed for growth */
#define BM 0.32 /* branch maintenance coefficient */
#define BE 1.5 /* branch maintenance exponent */
#define N_min 25 /* threshold for shedding */
Consider: ?E[]!L /* for context matching */
ω: !(W,1,1)F(2)L(1,LP,0,0)A(1,0)(!(0,0,0))!(W,0,1)

p₁: A(vig,del) : del<Del → A(vig,del+1)
p₂: L(vig,p,age,del) : (age<LS)&&(del<Del-1) → L(vig,p,age,del+1)
p₃: L(vig,p,age,del) : (age<LS)&&(del==Del-1)
→ L(vig,p,age,del+1)?E(vig*0.5)
p₄: L(vig,p,age,del) > ?E(r) : (age<LS) && (r*LP>=LM)
&& (del == Del) → L(vig,LP*r-LM,age+1,0)
p₅: L(vig,p,age,del) > ?E(r) : ((age == LS)||(r*LP<=LM))
&& (del == Del) → L(0,0,LS,0)
p₆: ?E(r) < A(vig,del) : r*LP-LM>PB {vig=vig*VD;}
→ /(\varphi)[+(α₂)!W,-PB,1]F(vig)L(vig,LP,0,0)A(vig,0)
[!(0,0,0)]!(W,0,1)
-(α₁)!W,0,1)F(vig)L(vig,LP,0,0)A(vig,0)
p₇: ?E(r) < A(vig,del) : r*LP-LM > PG {vig=vig*VD;}
→ /(\varphi)-(α₀)[!(0,0,0)]
(W,-PG,1)F(vig)L(vig,LP,0,0)A(vig,0)
p₈: ?E(r) < A(vig,del) : r*LP-LM <= PG → A(vig,0)
p₉: ?E(r) → ε
p₁₀: !(w₀,p₀,n₀) > L(vig,pₐ,age,del) [!(w₁,p₁,n₁)]!(w₂,p₂,n₂) :
{w=(w₁²+w₂²)⁰·⁵; p=p₁+p₂+pₗ-BM*(w/W)⁰·⁵BE;}
(p>0) || (n₁+n₂ >= N_min) → !(w,p,n₁+n₂)
p₁₁: !(w₀,p₀,n₀) > L(vig,pₐ,age,del) [!(w₁,p₁,n₁)]!(w₂,p₂,n₂)
→ !(w₀,0,0)L%
```

The simulation starts with a structure consisting of a branch segment  $F$ , supporting a leaf  $L$  and an apex  $A$  (axiom  $\omega$ ). The first branch width symbol  $!$  defines the segment width. Two additional symbols  $!$  following the apex create "virtual branches," needed to provide proper context for productions  $p_{10}$  and  $p_{11}$ . The tree grows in stages, with the delay of  $Del + 1$  derivation steps between consecutive stages introduced by production  $p_1$  for the apices and  $p_2$  for the leaves. Immediately before each new growth stage, communication symbols are introduced to inform the environment about the location and size of the leaf clusters ( $p_3$ ). If the flux  $r$  returned by the environment results in the production of photosynthates  $r * LP$

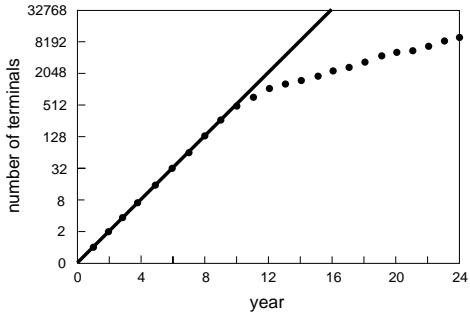


Figure 11: The number of terminal branch segments resulting from unrestricted bifurcation of apices (continuous line), compared to the number of segments generated in a simulation (isolated points)

exceeding the amount  $LM$  needed to maintain a cluster, it remains in the structure ( $p_4$ ). Otherwise it becomes a liability to the tree and dies ( $p_5$ ). Another condition to production  $p_5$  prevents a leaf from occupying the same location for more than  $LS$  years.

The flux  $r$  also determines the fate of the apex, captured by productions  $p_6$  to  $p_8$ . If the amount of photosynthates  $r * LP - LM$  transported from the nearby leaf exceeds a threshold value  $PB$ , the apex produces two new branches ( $p_6$ ). The second parameter in the first branch symbol  $!$  is set to  $-PB$ , to subtract the amount of photosynthates used for branching from the amount that will be transported further down. The length of branch segments  $vig$  is reduced with respect to the mother segment by a predefined factor  $VD$ , reflecting a gradual decrease in the vigor of apices with age. The branch width modules  $!$  following the first apex  $A$  are introduced to provide context required by productions  $p_{10}$  and  $p_{11}$ , as in the axiom.

If the amount of photosynthates  $r * LP - LM$  transported from the leaf is insufficient to produce new branches, but above the threshold  $PG$ , the apex adds a new segment  $F$  to the current branch axis without creating a lateral branch ( $p_7$ ). Again, a virtual branch containing the branch width symbol  $!$  is being added to provide context for productions  $p_{10}$  and  $p_{11}$ .

If the amount of photosynthates is below  $PG$ , the apex remains dormant ( $p_8$ ). Communication modules no longer needed are removed from the structure ( $p_9$ ).

Production  $p_{10}$  captures the endogenous information flow from leaves and terminal branch segments to the base of the tree. First, it determines the radius  $w$  of the mother branch segment as a function of the radii  $w_1$  and  $w_2$  of the supported branches:

$$w = \sqrt{w_1^2 + w_2^2}.$$

Thus, a cross section of the mother segment has an area equal to the sum of cross sections of the supported segments, as postulated in the literature [40, 46]. Next, production  $p_{10}$  calculates the flow  $p$  of photosynthates into the mother segment. It is defined as the sum of the flows  $p_L$ ,  $p_1$  and  $p_2$  received from the associated leaf  $L$  and from both daughter branches, decreased by the amount  $BM * (w/W)^{BE}$  representing the cost of maintaining the mother segment. Finally, production  $p_{10}$  calculates the number of terminal branch segments  $n$  supported by the mother segment as the sum of the numbers of terminal segments supported by the daughter branches,  $n_1$  and  $n_2$ .

Production  $p_{10}$  takes effect if the flow  $p$  is positive (the branch is not a liability to the tree), or if the number  $n$  of supported terminals



Figure 12: A tree model with branches competing for access to light, shown without the leaves



Figure 13: A climbing plant growing on the tree from the previous figure

is above a threshold  $N_{min}$ . If these conditions are not satisfied, production  $p_{11}$  removes (sheds) the branch from the tree using the cut symbol %.

**Simulations.** The competition for light between tree branches is manifested by two phenomena: reduced branching or dormancy

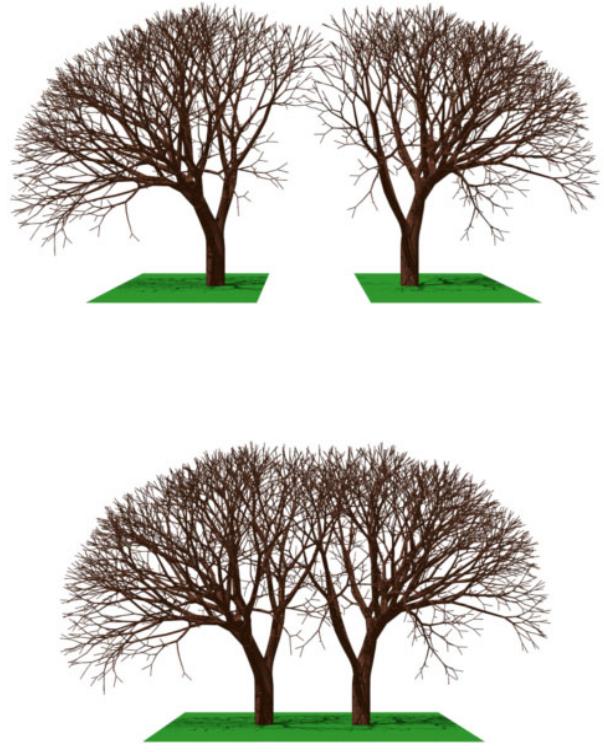


Figure 14: A model of deciduous trees competing for light. The trees are shown in the position of growth (top) and moved apart (bottom) to reveal the adaptation of crown geometry to the presence of the neighbor tree.

of apices in unfavorable local light conditions, and shedding of branches which do not receive enough light to contribute to the whole tree. Both phenomena limit the extent of branching, thus controlling the density of the crown. This property of the model is supported by the simulation results shown in Figure 11. If the growth was unlimited (production  $p_6$  was always chosen over  $p_7$  and  $p_8$ ), the number of terminal branch segments would double every year. Due to the competition for light, however, the number of terminal segments observed in an actual simulation increases more slowly. For related statistics using a different tree architecture see [52].

A tree image synthesized using an extension of the presented model is shown in Figure 12. The key additional feature is a gradual reduction of the branching angle of a young branch whose sister branch has been shed. As the result, the remaining branch assumes the role of the leading shoot, following the general growth direction of its supporting segment. Branch segments are represented as texture-mapped generalized cylinders, smoothly connected at the branching points (*cf.* [6]). The bark texture was created using a paint program.

As an illustration of the flexibility of the modeling framework presented in this paper, Figure 13 shows the effect of seeding a hypothetical climbing plant near the same tree. The plant follows the surface of the tree trunk and branches, and avoids excessively dense

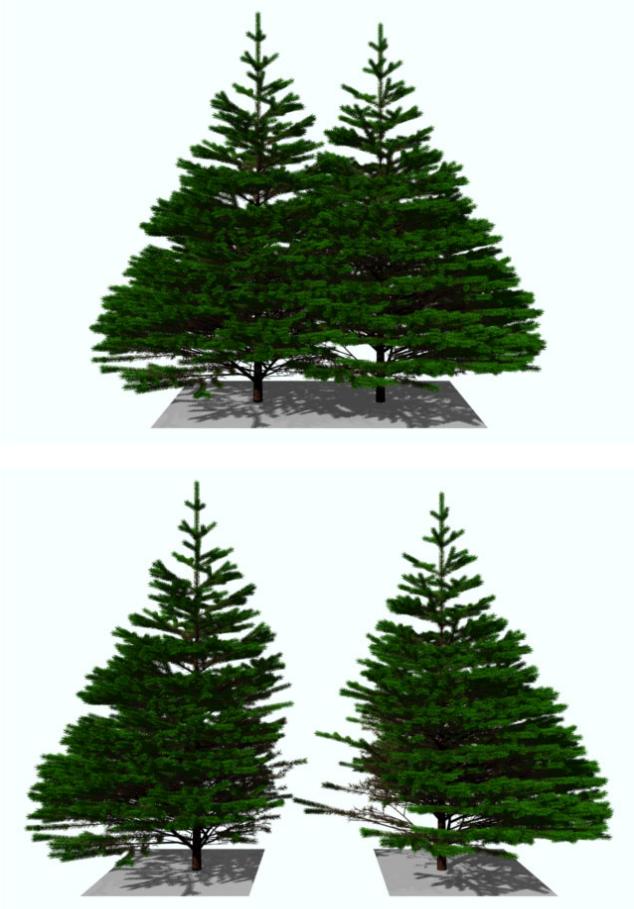


Figure 15: A model of coniferous trees competing for light. The trees are shown in the position of growth (top) and moved apart (bottom).

colonization of any particular area. Thus, the model integrates several environmentally-controlled phenomena: the competition of tree branches for light, the following of surfaces by a climbing plant, and the prevention of crowding as discussed in Section 6. Leaves were modeled using cubic patches (*cf.* [46]).

In the simulations shown in Figure 14 two trees described by the same set of rules (younger specimens of the tree from Figure 12) compete for light from the sky hemisphere. Moving the trees apart after they have grown reveals the adaptation of their crowns to the presence of the neighbor tree. This simulation illustrates both the necessity and the possibility of incorporating the adaptive behavior into tree models used for landscape design purposes.

The same phenomenon applies to coniferous trees, as illustrated in Figure 15. The tree model is similar to the original model by Takenaka [52] and can be viewed as consisting of approximately horizontal tiers (as discussed in Section 5) produced in sequence by the apex of the tree stem. The lower tiers are created first and therefore potentially can spread more widely than the younger tiers higher up (the *phase effect* [46]). This pattern of development is affected by the presence of the neighboring tree: the competition for light prevents the crowns from growing into each other.

The trees in Figure 15 retain branches that do not receive enough light. In contrast, the trees in the stand presented in Figure 16 shed

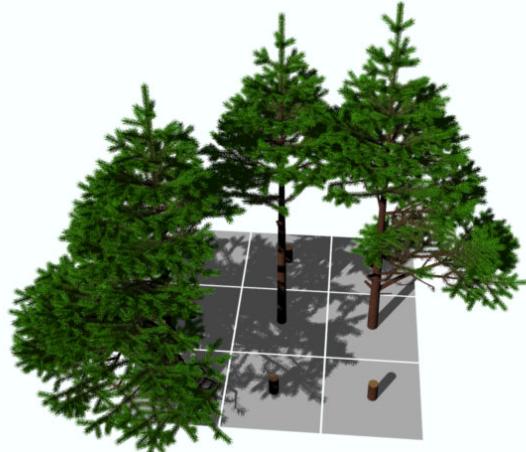


Figure 16: Relationship between tree form and its position in a stand.

branches that do not contribute photosynthates to the entire tree, using the same mechanism as described for the deciduous trees. The resulting simulation reveals essential differences between the shape of the tree crown in the middle of a stand, at the edge, or at the corner. In particular, the tree in the middle retains only the upper part of its crown. In lumber industry, the loss of lower branches is usually a desirable phenomenon, as it reduces knots in the wood and the amount of cleaning that trees require before transport. Simulations may assist in choosing an optimal distance for planting trees, where self-pruning is maximized, yet there is sufficient space between trees to allow for unimpeded growth of trunks in height and diameter.

## 9 CONCLUSIONS

In this paper, we introduced a framework for the modeling and visualization of plants interacting with their environment. The essential elements of this framework are:

- a system design, in which the plant and the environment are treated as two separate processes, communicating using a standard interface, and
- the language of open L-systems, used to specify plant models that can exchange information with the environment.

We demonstrated the operation of this framework by implementing models that capture collisions between branches, the propagation of clonal plants, the development of roots in soil, and the development of tree crowns competing for light. We found that the proposed framework makes it possible to easily create and modify models spanning a wide range of plant structures and environmental processes. Simulations of the presented phenomena were fast enough to allow interactive experimentation with the models (Table 1).

There are many research topics that may be addressed using the simulation and visualization capabilities of the proposed framework. They include, for instance:

- Fundamental analysis of the role of different forms of information flow in plant morphogenesis (in particular, the relationship

Fig.	Number of		Derivation		Time <sup>a</sup>	
	branch segments	leaf clusters	steps	yrs	sim.	render.
5	138	140	5	5	1 s	1 s
7	786	229	182	NA	50 s	2 s
9	4194	34 <sup>b</sup>	186	NA	67 s	3 s
10	37228	448 <sup>b</sup>	301	NA	15 min	70 s
12	22462	19195	744	24	22 min	13 s <sup>c</sup>
15	13502	3448	194	15	4 min	8 s <sup>d</sup>

<sup>a</sup> Simulation and rendering using OpenGL on a 200MHz/64MB Indigo<sup>2</sup> Extreme

<sup>b</sup> active apices

<sup>c</sup> without generalized cylinders and texture mapping

<sup>d</sup> branching structure without needles

Table 1: Numbers of primitives and simulation/rendering times for generating and visualizing selected models

between endogenous and exogenous flow). This is a continuation of the research pioneered by Bell [4] and Honda *et al.* [7, 33].

- Development of a comprehensive plant model describing the cycling of nutrients from the soil through the roots and branches to the leaves, then back to the soil in the form of substances released by fallen leaves.
- Development of models of specific plants for research, crop and forest management, and for landscape design purposes. The models may include environmental phenomena not discussed in this paper, such as the global distribution of radiative energy in the tree crowns, which affects the amount of light reaching the leaves and the local temperature of plant organs.

The presented framework itself is also open to further research. To begin, the precise functional specification of the environment, implied by the design of the modeling framework, is suitable for a formal analysis of algorithms that capture various environmental processes. This analysis may highlight tradeoffs between time, memory, and communication complexity, and lead to programs matching the needs of the model to available system resources in an optimal manner.

A deeper understanding of the spectrum of processes taking place in the environment may lead to the design of a mini-language for environment specification. Analogous to the language of L-systems for plant specification, this mini-language would simplify the modeling of various environments, relieving the modeler from the burden of low-level programming in a general-purpose language. Fleischer and Barr's work on the specification of environments supporting collisions and reaction-diffusion processes [20] is an inspiring step in this direction.

Complexity issues are not limited to the environment, but also arise in plant models. They become particularly relevant as the scope of modeling increases from individual plants to groups of plants and, eventually, entire plant communities. This raises the problem of selecting the proper level of abstraction for designing plant models, including careful selection of physiological processes incorporated into the model and the spatial resolution of the resulting structures.

The complexity of the modeling task can be also addressed at the level of system design, by assigning various components of the model (individual plants and aspects of the environment) to different components of a distributed computing system. The communication

structure should then be redesigned to accommodate information transfers between numerous processes within the system.

In summary, we believe that the proposed modeling methodology and its extensions will prove useful in many applications of plant modeling, from research in plant development and ecology to landscape design and realistic image synthesis.

## Acknowledgements

We would like to thank Johannes Battjes, Campbell Davidson, Art Diggle, Heinjo During, Michael Guzy, Naoyoshi Kanamaru, Bruno Moulia, Zbigniew Prusinkiewicz, Bill Remphrey, David Reid, and Peter Room for discussions and pointers to the literature relevant to this paper. We would also like to thank Bruno Andrieu, Mark Hammel, Jim Hanan, Lynn Mercer, Chris Prusinkiewicz, Peter Room, and the anonymous referees for helpful comments on the manuscript. Most images were rendered using the ray tracer *rayshade* by Craig Kolb. This research was sponsored by grants from the Natural Sciences and Engineering Research Council of Canada.

## REFERENCES

- [1] AGRAWAL, P. The cell programming language. *Artificial Life* 2, 1 (1995), 37–77.
- [2] ARVO, J., AND KIRK, D. Modeling plants with environment-sensitive automata. In *Proceedings of Ausgraph'88* (1988), pp. 27 – 33.
- [3] BELL, A. *Plantform: An illustrated guide to flowering plants*. Oxford University Press, Oxford, 1991.
- [4] BELL, A. D. The simulation of branching patterns in modular organisms. *Philos. Trans. Royal Society London, Ser. B* 313 (1986), 143–169.
- [5] BELL, A. D., ROBERTS, D., AND SMITH, A. Branching patterns: the simulation of plant architecture. *Journal of Theoretical Biology* 81 (1979), 351–375.
- [6] BLOOMENTHAL, J. Modeling the Mighty Maple. Proceedings of SIGGRAPH '85 (San Francisco, California, July 22–26, 1985), in *Computer Graphics*, 19, 3 (July 1985), pages 305–311, ACM SIGGRAPH, New York, 1985.
- [7] BORCHERT, R., AND HONDA, H. Control of development in the bifurcating branch system of *Tabebuia rosea*: A computer simulation. *Botanical Gazette* 145, 2 (1984), 184–195.
- [8] BORCHERT, R., AND SLADE, N. Bifurcation ratios and the adaptive geometry of trees. *Botanical Gazette* 142, 3 (1981), 394–401.
- [9] CHEN, S. G., CEULEMANS, R., AND IMPENS, I. A fractal based *Populus* canopy structure model for the calculation of light interception. *Forest Ecology and Management* (1993).
- [10] CHIBA, N., OHKAWA, S., MURAOKA, K., AND MIURA, M. Visual simulation of botanical trees based on virtual heliotropism and dormancy break. *The Journal of Visualization and Computer Animation* 5 (1994), 3–15.
- [11] CHIBA, N., OHSHIDA, K., MURAOKA, K., MIURA, M., AND SAITO, N. A growth model having the abilities of growth-regulations for simulating visual nature of botanical trees. *Computers and Graphics* 18, 4 (1994), 469–479.
- [12] CLAUSNITZER, V., AND HOPMANS, J. Simultaneous modeling of transient three-dimensional root growth and soil water flow. *Plant and Soil* 164 (1994), 299–314.
- [13] COHEN, D. Computer simulation of biological pattern generation processes. *Nature* 216 (October 1967), 246–248.

- [14] COHEN, M., AND WALLACE, J. *Radiosity and realistic image synthesis*. Academic Press Professional, Boston, 1993. With a chapter by P. Hanrahan and a foreword by D. Greenberg.
- [15] DE REFFYE, P., HOULLIER, F., BLAISE, F., BARTHELEMY, D., DAUZAT, J., AND AUCLAIR, D. A model simulating above- and below-ground tree architecture with agroforestry applications. *Agroforestry Systems* 30 (1995), 175–197.
- [16] DIGGLE, A. J. ROOTMAP - a model in three-dimensional coordinates of the structure and growth of fibrous root systems. *Plant and Soil* 105 (1988), 169–178.
- [17] DONG, M. *Foraging through morphological response in clonal herbs*. PhD thesis, University of Utrecht, October 1994.
- [18] FISHER, J. B. How predictive are computer simulations of tree architecture. *International Journal of Plant Sciences* 153 (Suppl.) (1992), 137–146.
- [19] FISHER, J. B., AND HONDA, H. Computer simulation of branching pattern and geometry in *Terminalia* (Combretaceae), a tropical tree. *Botanical Gazette* 138, 4 (1977), 377–384.
- [20] FLEISCHER, K. W., AND BARR, A. H. A simulation testbed for the study of multicellular development: The multiple mechanisms of morphogenesis. In *Artificial Life III*, C. G. Langton, Ed. Addison-Wesley, Redwood City, 1994, pp. 389–416.
- [21] FORD, E. D., AVERY, A., AND FORD, R. Simulation of branch growth in the *Pinaceae*: Interactions of morphology, phenology, foliage productivity, and the requirement for structural support, on the export of carbon. *Journal of Theoretical Biology* 146 (1990), 15–36.
- [22] FORD, H. Investigating the ecological and evolutionary significance of plant growth form using stochastic simulation. *Annals of Botany* 59 (1987), 487–494.
- [23] FRIJTERS, D., AND LINDENMAYER, A. A model for the growth and flowering of *Aster novae-angliae* on the basis of table (1,0)L-systems. In *L Systems*, G. Rozenberg and A. Salomaa, Eds., Lecture Notes in Computer Science 15. Springer-Verlag, Berlin, 1974, pp. 24–52.
- [24] GARDNER, W. R. Dynamic aspects of water availability to plants. *Soil Science* 89, 2 (1960), 63–73.
- [25] GREENE, N. Voxel space automata: Modeling with stochastic growth processes in voxel space. Proceedings of SIGGRAPH '89 (Boston, Mass., July 31–August 4, 1989), in *Computer Graphics* 23, 4 (August 1989), pages 175–184, ACM SIGGRAPH, New York, 1989.
- [26] GREENE, N. Detailing tree skeletons with voxel automata. SIGGRAPH '91 Course Notes on Photorealistic Volume Modeling and Rendering Techniques, 1991.
- [27] GUZY, M. R. A morphological-mechanistic plant model formalized in an object-oriented parametric L-system. Manuscript, USDA-ARS Salinity Laboratory, Riverside, 1995.
- [28] HANAN, J. Virtual plants — Integrating architectural and physiological plant models. In *Proceedings of ModSim 95* (Perth, 1995), P. Binning, H. Bridgman, and B. Williams, Eds., vol. 1, The Modelling and Simulation Society of Australia, pp. 44–50.
- [29] HANAN, J. S. *Parametric L-systems and their application to the modelling and visualization of plants*. PhD thesis, University of Regina, June 1992.
- [30] HART, J. W. *Plant tropisms and other growth movements*. Unwin Hyman, London, 1990.
- [31] HERMAN, G. T., AND ROZENBERG, G. *Developmental systems and languages*. North-Holland, Amsterdam, 1975.
- [32] HONDA, H. Description of the form of trees by the parameters of the tree-like body: Effects of the branching angle and the branch length on the shape of the tree-like body. *Journal of Theoretical Biology* 31 (1971), 331–338.
- [33] HONDA, H., TOMLINSON, P. B., AND FISHER, J. B. Computer simulation of branch interaction and regulation by unequal flow rates in botanical trees. *American Journal of Botany* 68 (1981), 569–585.
- [34] KAANDORP, J. *Fractal modelling: Growth and form in biology*. Springer-Verlag, Berlin, 1994.
- [35] KANAMARU, N., CHIBA, N., TAKAHASHI, K., AND SAITO, N. CG simulation of natural shapes of botanical trees based on heliotropism. *The Transactions of the Institute of Electronics, Information, and Communication Engineers J75-D-II*, 1 (1992), 76–85. In Japanese.
- [36] KURTH, W. *Growth grammar interpreter GROGRA 2.4: A software tool for the 3-dimensional interpretation of stochastic, sensitive growth grammars in the context of plant modeling. Introduction and reference manual*. Forschungszentrum Waldökosysteme der Universität Göttingen, Göttingen, 1994.
- [37] LIDDELL, C. M., AND HANSEN, D. Visualizing complex biological interactions in the soil ecosystem. *The Journal of Visualization and Computer Animation* 4 (1993), 3–12.
- [38] LINDENMAYER, A. Mathematical models for cellular interaction in development, Parts I and II. *Journal of Theoretical Biology* 18 (1968), 280–315.
- [39] LINDENMAYER, A. Developmental systems without cellular interaction, their languages and grammars. *Journal of Theoretical Biology* 30 (1971), 455–484.
- [40] MACDONALD, N. *Trees and networks in biological models*. J. Wiley & Sons, New York, 1983.
- [41] PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. *Numerical recipes in C: The art of scientific computing. Second edition*. Cambridge University Press, Cambridge, 1992.
- [42] PRUSINKIEWICZ, P. Visual models of morphogenesis. *Artificial Life* 1, 1/2 (1994), 61–74.
- [43] PRUSINKIEWICZ, P., HAMMEL, M., HANAN, J., AND MĚCH, R. Visual models of plant development. In *Handbook of formal languages*, G. Rozenberg and A. Salomaa, Eds. Springer-Verlag, Berlin, 1996. To appear.
- [44] PRUSINKIEWICZ, P., AND HANAN, J. L-systems: From formalism to programming languages. In *Lindenmayer systems: Impacts on theoretical computer science, computer graphics, and developmental biology*, G. Rozenberg and A. Salomaa, Eds. Springer-Verlag, Berlin, 1992, pp. 193–211.
- [45] PRUSINKIEWICZ, P., JAMES, M., AND MĚCH, R. Synthetic topiary. Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994), pages 351–358, ACM SIGGRAPH, New York, 1994.
- [46] PRUSINKIEWICZ, P., AND LINDENMAYER, A. *The algorithmic beauty of plants*. Springer-Verlag, New York, 1990. With J. S. Hanan, F. D. Fracchia, D. R. Fowler, M. J. M. de Boer, and L. Mercer.
- [47] ROOM, P. M. ‘Falling apart’ as a lifestyle: the rhizome architecture and population growth of *Salvinia molesta*. *Journal of Ecology* 71 (1983), 349–365.
- [48] ROOM, P. M., MAILLETTE, L., AND HANAN, J. Module and metamer dynamics and virtual plants. *Advances in Ecological Research* 25 (1994), 105–157.
- [49] ROZENBERG, G. T0L systems and languages. *Information and Control* 23 (1973), 357–381.
- [50] SACHS, T., AND NOVOPLANSKY, A. Tree from: Architectural models do not suffice. *Israel Journal of Plant Sciences* 43 (1995), 203–212.
- [51] SIPPER, M. Studying artificial life using a simple, general cellular model. *Artificial Life* 2, 1 (1995), 1–35.
- [52] TAKENAKA, A. A simulation model of tree architecture development based on growth response to local light environment. *Journal of Plant Research* 107 (1994), 321–330.
- [53] ULAM, S. On some mathematical properties connected with patterns of growth of figures. In *Proceedings of Symposia on Applied Mathematics* (1962), vol. 14, American Mathematical Society, pp. 215–224.
- [54] WEBER, J., AND PENN, J. Creation and rendering of realistic trees. Proceedings of SIGGRAPH '95 (Los Angeles, California, August 6–11, 1995), pages 119–128, ACM SIGGRAPH, New York, 1995.
- [55] WYVILL, G., MCPHEETERS, C., AND WYVILL, B. Data structure for soft objects. *The Visual Computer* 2, 4 (February 1986), 227–234.

# Realistic modeling and rendering of plant ecosystems

Oliver Deussen<sup>1</sup>

Pat Hanrahan<sup>2</sup>

Bernd Lintermann<sup>3</sup>

Radomír Měch<sup>4</sup>

Matt Pharr<sup>2</sup>

Przemysław Prusinkiewicz<sup>4</sup>

<sup>1</sup> Otto-von-Guericke University of Magdeburg

<sup>2</sup> Stanford University

<sup>3</sup> The ZKM Center for Art and Media Karlsruhe

<sup>4</sup> The University of Calgary\*

## Abstract

Modeling and rendering of natural scenes with thousands of plants poses a number of problems. The terrain must be modeled and plants must be distributed throughout it in a realistic manner, reflecting the interactions of plants with each other and with their environment. Geometric models of individual plants, consistent with their positions within the ecosystem, must be synthesized to populate the scene. The scene, which may consist of billions of primitives, must be rendered efficiently while incorporating the subtleties of lighting in a natural environment.

We have developed a system built around a pipeline of tools that address these tasks. The terrain is designed using an interactive graphical editor. Plant distribution is determined by hand (as one would do when designing a garden), by ecosystem simulation, or by a combination of both techniques. Given parametrized procedural models of individual plants, the geometric complexity of the scene is reduced by *approximate instancing*, in which similar plants, groups of plants, or plant organs are replaced by instances of representative objects before the scene is rendered. The paper includes examples of visually rich scenes synthesized using the system.

**CR categories:** I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism, I.6.3 [Simulation and Modeling]: Applications, J.3 [Life and Medical Sciences]: Biology.

**Keywords:** realistic image synthesis, modeling of natural phenomena, ecosystem simulation, self-thinning, plant model, vector quantization, approximate instancing.

## 1 INTRODUCTION

Synthesis of realistic images of terrains covered with vegetation is a challenging and important problem in computer graphics. The challenge stems from the visual complexity and diversity of the modeled scenes. They include natural ecosystems such as forests or

grasslands, human-made environments, for instance parks and gardens, and intermediate environments, such as lands recolonized by vegetation after forest fires or logging. Models of these ecosystems have a wide range of existing and potential applications, including computer-assisted landscape and garden design, prediction and visualization of the effects of logging on the landscape, visualization of models of ecosystems for research and educational purposes, and synthesis of scenes for computer animations, drive and flight simulators, games, and computer art.

Beautiful images of forests and meadows were created as early as 1985 by Reeves and Blau [50] and featured in the computer animation *The Adventures of André and Wally B.* [34]. Reeves and Blau organized scene modeling as a sequence of steps: specification of a terrain map that provides the elevation of points in the scene, interactive or procedural placement of vegetation in this terrain, modeling of individual plants (grass and trees), and rendering of the models. This general scheme was recently followed by Chiba *et al.* [8] in their work on forest rendering, and provides the basis for commercial programs devoted to the synthesis of landscapes [2, 49].

The complexity of nature makes it necessary to carefully allot computing resources — CPU time, memory, and disk space — when recreating natural scenes with computer graphics. The size of the database representing a scene during the rendering is a particularly critical item, since the amount of geometric data needed to represent a detailed outdoor scene is more than can be represented on modern computers. Consequently, a survey of previous approaches to the synthesis of natural scenes reflects the quest for a good tradeoff between the realism of the images and the amount of resources needed to generate them.

The scenes synthesized by Reeves and Blau were obtained using (structured) particle systems, with the order of one million particles per tree [50]. To handle large numbers of primitive elements contributing to the scene, the particle models of individual trees were generated procedurally and rendered sequentially, each model discarded as soon as a tree has been rendered. Consequently, the size of memory needed to generate the scene was proportional to the number of particles in a single tree, rather than the total number of particles in the scene. This approach required approximate shading calculations, since the detailed information about the neighborhood trees was not available. Approximate shading also reduced the time needed to render the scenes.

Another approach to controlling the size of scene representation is the reduction of visually unimportant detail. General methods for achieving this reduction have been the subject of intense research (for a recent example and further references see [24]), but the results do not easily apply to highly branching plant structures. Consequently, Weber and Penn [63] introduced a heuristic multiresolution representation specific to trees, which allows for reducing

\* Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4 ([pwp@cpsc.ucalgary.ca](mailto:pwp@cpsc.ucalgary.ca))

the number of geometric primitives in the models that occupy only a small portion on the screen. A multiresolution representation of botanical scenes was also explored by Marshall *et al.* [35], who integrated polygonal representations of larger objects with tetrahedral approximations of the less relevant parts of a scene.

A different strategy for creating visually complex natural scenes was proposed by Gardner [17]. In this case, the terrain and the trees were modeled using a relatively small number of geometric primitives (quadric surfaces). Their perceived complexity resulted from procedural textures controlling the color and the transparency of tree crowns. In a related approach, trees and other plants were represented as texture-mapped flat polygons (for example, see [49]). This approach produced visible artifacts when the position of the viewpoint was changed. A more accurate image-based representation was introduced by Max [37], who developed an algorithm for interpolating between precomputed views of trees. A multiresolution extension of this method, taking advantage of the hierarchical structure of the modeled trees, was presented in [36]. Shade *et al.* described a hybrid system for walkthroughs that uses a combination of geometry and textured polygons [53].

Kajiya and Kay [26] introduced volumetric textures as an alternative paradigm for overcoming the limitations of texture-mapped polygons. A method for generating terrains with volumetric textures representing grass and trees was proposed by Neyret [40, 41]. Chiba *et al.* [8] removed the deformations of plants caused by curvatures of the underlying terrain by allowing texels to intersect.

The use of volumetric textures limits the memory or disk space needed to represent a scene, because the same texel can be re-applied to multiple areas. The same idea underlies the oldest approach to harnessing visually complex scenes, object instancing [59]. According to the paradigm of instancing, an object that appears several times in a scene (possibly resized or deformed by an affine transformation) is defined only once, and its different occurrences (instances) are specified by affine transformations of the prototype. Since the space needed to represent the transformations is small, the space needed to represent an entire scene depends primarily on the number and complexity of *different* objects, rather than the number of their instances. Plants are particularly appealing objects of instancing, because repetitive occurrences can be found not only at the level of plant species, but also at the level of plant organs and branching structures. This leads to compact hierarchical data structures conducive to fast ray tracing, as discussed by Kay and Kajiya [27], and Snyder and Barr [56]. Hart and DeFanti [20, 21] further extended the paradigm of instancing from hierarchical to recursive (self-similar) structures. All the above papers contain examples of botanical scenes generated using instancing.

The complexity of natural scenes makes them not only difficult to render, but also to specify. Interactive modeling techniques, available in commercial packages such as Alias/Wavefront Studio 8 [1], focus on the direct manipulation of a relatively small number of surfaces. In contrast, a landscape with plants may include many millions of individual surfaces — representing stems, leaves, flowers, and fruits — arranged into complex branching structures, and further organized in an ecosystem. In order to model and render such scenes, we employ the techniques summarized below.

**Multilevel modeling and rendering pipeline.** Following the approach initiated by Reeves and Blau [50], we decompose the process of image synthesis into stages: modeling of the terrain, specification of plant distribution, modeling of the individual plants, and rendering of the entire scene. Each of these stages operates at a different level of abstraction, and provides a relatively high-level input for the next stage. Thus, the modeler is not concerned with

plant distribution when specifying the terrain, and plant distribution is determined (interactively or algorithmically) without considering details of the individual plants. This is reminiscent of the simulations of flocks of birds [51], models of flower heads with phyllotactic patterns [16], and models of organic structures based on morphogenetic processes [14], where simulations were performed using geometrically simpler objects than those used for visualization. Blumberg and Galyean extended this paradigm to *multi-level* direction of autonomous animated agents [5]. In an analogous way, we apply it to multi-level modeling.

**Open system architecture.** By clearly specifying the formats of inputs and outputs for each stage of the pipeline, we provide a framework for incorporating independently developed modules into our system. This open architecture makes it possible to augment the complexity of the modeled scenes by increasing the range of the available modules, and facilitates experimentation with various approaches and algorithms.

**Procedural models.** As observed by Smith [55], procedural models are often characterized by a significant *data-base amplification*, which means that they can generate complex geometric structures from small input data sets. We benefit from this phenomenon by employing procedural models in all stages of the modeling pipeline.

**Approximate instancing.** We use object instancing as the primary paradigm for reducing the size of the geometric representation of the rendered scenes. To increase the degree of instancing, we cluster scene components (plants and their parts) in their parameter spaces, and approximate all objects within a given cluster with instances of a single representative object. This idea was initially investigated by Brownbill [7]; we extend it further by applying vector quantization (*c.f.* [18]) to find the representative objects in multidimensional parameter spaces.

**Efficient rendering.** We use memory- and time-efficient rendering techniques: decomposition of the scenes into subscenes that are later composited [12], ray tracing with instancing and a support for rendering many polygons [56], and memory-coherent ray tracing [43] with instancing.

By employing these techniques, we have generated scenes with up to 100,000 detailed plant models. This number could be increased even further, since none of the scenes required more than 150MB to store. However, with 100,000 plants, each plant is visible on average only in 10 pixels of a 1K × 1K image. Consequently, we seem to have reached the limits of useful scene complexity, because the level of visible detail is curbed by the size and resolution of the output device.

## 2 SYSTEM ARCHITECTURE

The considerations presented in the previous section led us to the modular design of our modeling and rendering system *EcoSys*, shown schematically in Figure 1.

The modeling process begins with the specification of a terrain. For this purpose, we developed an interactive editor *TerrEdit*, which integrates a number of terrain modeling techniques (Section 3). Its output, a *terrain data* file, includes the altitudes of a grid of points superimposed on the terrain, normal vectors indicating the local slope of the terrain, and optional information describing environmental conditions, such as soil humidity.

The next task is to determine plant distribution on a given terrain. We developed two techniques for this purpose: visual editing of plant densities and simulation of plant interactions within an ecosystem

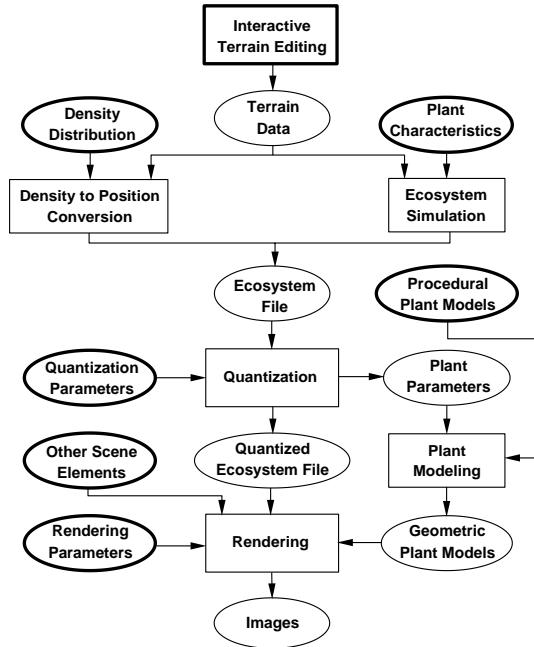


Figure 1: Architecture of the scene synthesis system. Bold frames indicate interactive programs and input files specified by the user.

(Section 4). The editing approach is particularly well suited to model environments designed by people, for example orchards, gardens, or parks. The user specifies the distribution of plant densities using a paint program. To convert this information into positions of individual plants, we developed the program *densedis* based on a half-toning algorithm: each dot becomes a plant. We can also specify positions of individual plants explicitly; this is particularly important in the synthesis of scenes that include detailed views of individual plants in the foreground.

To define plant distribution in natural environments, such as forests or meadows, we apply an ecosystem simulation model. Its input consists of terrain data, ecological characteristics of plant species (for example, annual or perennial growth and reproduction cycle, preference for wet or dry soil, and shade tolerance) and, optionally, the initial distribution of plants. The growth of plants is simulated accounting for competition for space, sunlight, resources in the soil, aging and death, seed distribution patterns, *etc.* We perform these simulations using the L-system-based plant modeling program *cpfg* [47], extended with capabilities for simulating interactions between plants and their environments [39]. To allow for simulations involving thousands of plants, we use their simplified geometrical representations, which are subsequently replaced by detailed plant models for visualization purposes.

Specification of a plant distribution may involve a combination of interactive and simulation techniques. For example, a model of an orchard may consist of trees with explicitly specified positions and weeds with positions determined by a simulation. Conversely, the designer of a scene may wish to change its aspects after an ecological simulation for aesthetic reasons. To allow for these operations, both *densedis* and *cpfg* can take a given plant distribution as input for further processing.

Plant distribution, whether determined interactively or by ecosystem simulation, is represented in an *ecosystem* file. It contains the information about the type, position and orientation of each plant

(which is needed to assemble the final scene), and parameters of individual plants (which are needed to synthesize their geometric models).

Since we wish to render scenes that may include thousands of plants, each possibly with many thousands of polygons, the creation and storage of a separate geometric plant model for each plant listed in the ecosystem file is not practical. Consequently, we developed a program *quantv* that clusters plants in their parameter space and determines a representative plant for each cluster (Section 6). The algorithm performs quantization adaptively, thus the result depends on the characteristics of plants in the ecosystem. The quantization process produces two outputs: a *plant parameter* file, needed to create geometric models of the representative plants, and a *quantized ecosystem* file, which specifies positions and orientations of the instances of representative plants throughout the scene.

We employ two modeling programs to create the representative plants: the interactive plant modeler *xfrog* [10, 32, 33] and the L-system-based simulator *cpfg* [39, 47]. These programs input parametrized *procedural plant models* and generate specific *geometric plant models* according to the values in the plant parameter file (Section 5). To reduce the amount of geometric data, we extended the concept of instancing and quantization to components of plants. Thus, if a particular plant or group of plants has several parts (such as branches, leaves, or flowers) that are similar in their respective parameter spaces, we replace all occurrences of these parts with instances of a representative part.

Finally, the ecosystem is rendered. The input for rendering consists of the quantized ecosystem file and the representative plant models. Additional information may include geometry of the terrain and human-made objects, such as houses or fences. In spite of the quantization and instancing, the resulting scene descriptions may still be large. We experimented with three renderers to handle this complexity (Section 7). One renderer, called *fshade*, decomposes the scene into sub-scenes that are rendered individually and composited to form final images. Unfortunately, separating the scene into sub-scenes makes it impossible to properly capture global illumination effects. To alleviate this problem, we use the ray-tracer *rayshade* [29], which offers support for instancing and time-efficient rendering of scenes with many polygons, as long as the scene description fits in memory. When the scene description exceeds the available memory, we employ the memory-efficient ray-tracer *toro* [43], extended with a support for instancing.

In the following sections we describe the components of the EcoSys modeling pipeline in more detail. In Section 8, we present examples that illustrate the operation of the system as a whole.

### 3 TERRAIN SPECIFICATION

We begin the modeling of a scene with the specification of a terrain. The goal of this step is to determine elevation data, local orientations of the terrain, and additional characteristics, such as the water content in the soil, which may affect the type and vigor of plants at different locations.

Terrain data may have several sources. Representations of real terrains are available, for example, from the U.S. Geological Survey [30]. Several techniques have also been developed for creating synthetic terrains. They include: hand-painted height maps [65], methods for generating fractal terrains (reviewed in [38]), and models based on the simulation of soil erosion [28, 38].

In order to provide detailed control over the modeled terrain while taking advantage of the data amplification of fractal methods [55],

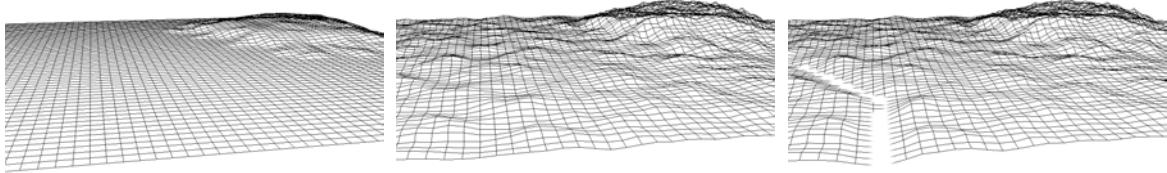


Figure 2: Three stages in creating a terrain: after loading a height map painted by hand (left), with hills added using noise synthesis (middle), and with a stream cut using the stream mask (right).

we designed and implemented an interactive terrain editing system *TerEdit*, which combines various techniques in a procedural manner. Terrain editing consists of *operations*, which modify the terrain geometry and have the spatial scope limited by *masks*. A similar paradigm is used in Adobe Photoshop [9], where *selections* can be used to choose an arbitrary subset of an image to edit.

We assume that masks have values between zero and one, allowing for smooth blending of the effects of operations. Both masks and operations can depend on the horizontal coordinates and the altitude of the points computed so far. Thus, it is possible to have masks that select terrain above some altitude or operations that are functions of the current altitude. The user's editing actions create a pipeline of operations with associated masks; to compute the terrain altitude at a point, the stages of this pipeline are evaluated in order. Undo and redo operations are easily supported by removing and re-adding operations from the pipeline and re-evaluating the terrain.

Examples of editing operations include translation, scaling, non-linear scaling, and algorithmic synthesis of the terrain. The synthesis algorithm is based on noise synthesis [38], which generates realistic terrains by adding multiple scales of Perlin's noise function [42]. The user can adjust a small number of parameters that control the overall roughness of the terrain, the rate of change in roughness across the surface of the terrain, and the frequency of the noise functions used. Noise synthesis allows terrain to be easily evaluated at a single point, without considering the neighboring points; this makes it possible to have operations that act locally. Another advantage of noise synthesis is efficiency of evaluation; updating the wireframe terrain view (based on  $256 \times 256$  samples of the region of interest) after applying an operation typically takes under a second. On a multiprocessor machine, where terrain evaluation is multi-threaded, the update time is not noticeable.

The editor provides a variety of masks, including ones that select rectangular regions of terrain from a top view, masks that select regions based on their altitude, and masks defined by image files. One of the most useful masks is designed for cutting streams through terrain. The user draws a set of connected line segments over the terrain, and the influence of the mask is based on the minimum distance from a sample point to any of these segments. A spline is applied to smoothly increase the influence of the mask close to the segments. When used with a scaling operation, the terrain inside and near the stream is scaled towards the water level, and nearby terrain is ramped down, while the rest of the terrain is unchanged.

The specification of a terrain using *TerEdit* is illustrated in Figure 2. In the first snapshot, the hill in the far corner was defined by loading in a height map that had been painted by hand. Next, small hills were added to the entire terrain using noise synthesis. The last image shows the final terrain, after the stream mask was used to cut the path of a stream. A total of five operators were applied to make this terrain, and the total time to model it was approximately fifteen minutes.

Once the elevations have been created, additional parameters of the terrain can be computed as input for ecosystem simulations or a direct source of parameters for plant models. Although the user can interactively paint parameters on the terrain, simulation provides a more sound basis for the modeling of natural ecosystems. Consequently, *TerEdit* incorporates a simulator of rain water flow and distribution in the soil, related to both the erosion algorithm by Musgrave *et al.* [38] and the particle system simulation of water on building facades by Dorsey *et al.* [11]. Water is dropped onto the terrain from above; some is absorbed immediately while the rest flows downhill and is absorbed by the soil that it passes through. A sample terrain with the water distribution generated using this approach is shown in Figure 3.

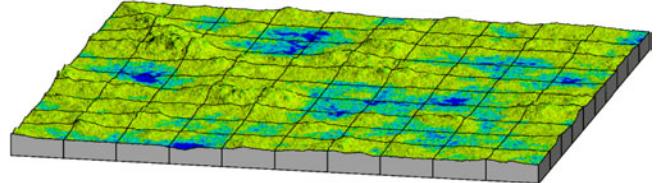


Figure 3: A sample terrain with the water concentration ranging from high (blue) to low (yellow)

## 4 SPECIFICATION OF PLANT POPULATIONS

The task of populating a terrain with plants can be addressed using methods that offer different tradeoffs between the degree of user control, time needed to specify plant distribution, and biological validity of the results. The underlying techniques can be divided into *space-occupancy* or *individual-based* techniques. This classification is related to paradigms of spatially-explicit modeling in ecology [3, 19], and parallels the distinction between space-based and structure-based models of morphogenesis [44].

The space-occupancy techniques describe the distribution of the *densities* of given plant species over a terrain. In the image synthesis context, this distribution can be obtained using two approaches:

**Explicit specification.** The distribution of plant densities is measured in the field (by counting plants that occupy sample plots) or created interactively, for example using a paint program.

**Procedural generation.** The distributions of plant densities is obtained by simulating interactions between plant populations using an ecological model. The models described in the literature are commonly expressed in terms of *cellular automata* [19] or *reaction-diffusion* processes [23].

The individual-based techniques provide the location and attributes of *individual plants*. Again, we distinguish two approaches:

**Explicit specification.** Plant positions and attributes represent field data, for example obtained by surveying a real forest [25], or specified interactively by the user.

**Procedural generation.** Plant positions and attributes are obtained using a *point pattern generation model*, which creates a distribution of points with desired statistical properties [66], or using an *individual-based population model* [13, 58], which is applied to simulate interactions between plants within an ecosystem.

Below we describe two methods for specifying plant distribution that we have developed and implemented as components of EcoSys. The first method combines interactive editing of plant densities with a point pattern generation of the distribution of individual plants. The second method employs individual-based ecological simulations.

#### 4.1 Interactive specification of plant populations

To specify a plant population in a terrain, the user creates a set of gray-level images with a standard paint program. These images define the spatial distributions of plant densities and of plant characteristics such as the age and vigor.

Given an image that specifies the distribution of densities of a plant species, positions of individual plants are generated using a half-toning algorithm. We have used the Floyd-Steinberg algorithm [15] for this purpose. Each black pixel describes the position of a plant in the raster representing the terrain. We also have implemented a relaxation method that iteratively moves each plant position towards the center of mass of its Voronoi polygon [6]. This reduces the variance of distances between neighboring plants, which sometimes produces visually pleasing effects.

Once the position of a plant has been determined, its parameters are obtained by referring to the values of appropriate raster images at the same point. These values may control the plant model directly or provide arguments to user-specified mathematical expressions, which in turn control the models. This provides the user with an additional means for flexibly manipulating the plant population.

Operations on raster images make it possible to capture some interactions between plants. For example, if the radius of a tree crown is known, the image representing the projection of the crown on the ground may be combined with user-specified raster images to decrease the density or vigor of plants growing underneath.

#### 4.2 Simulation of ecosystems

Individual-based models of plant ecosystems operate at various levels of abstraction, depending on the accuracy of the representation of individual plants [58]. Since our goal is to simulate complex scenes with thousands of plants, we follow the approach of Firbank and Watkinson [13], and represent plants coarsely as circles positioned in a continuous landscape. Each circle defines the *ecological neighborhood* of the plant in its center, that is the area within which the plant interacts with its neighbors. Biologically motivated rules govern the outcomes of interactions between the intersecting circles. Global behavior of the ecosystem model is an emergent property of a system of many circles.

We implemented the individual-based ecosystem models using the framework of *open L-systems* [39]. Since L-systems operate on branching structures, we represent each plant as a circle located at the end of an invisible positioning line. All lines are connected into a branching structure that spreads over the terrain.

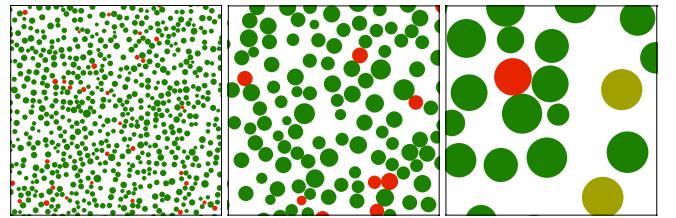


Figure 4: Steps 99, 134, and 164 of a sample simulation of the self-thinning process. Colors represent states of the plants: not dominated (green), dominated (red), and old (yellow). The simulation began with 62,500 plants, placed at random in a square field. Due to the large number of plants, only a part of the field is shown.

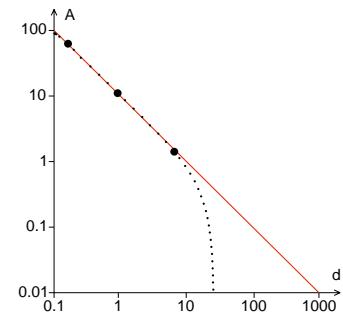


Figure 5: The average area of plants as a function of their density. Small dots represent the results of every tenth simulation step. Large dots correspond to the states of simulation shown in Figure 4.

For example, let us consider a model of plant distribution due to a fundamental phenomenon in plant population dynamics, *self-thinning*. This phenomenon is described by Ricklefs as follows [52, page 339]: “If one plots the logarithm of average plant weight as a function of the logarithm of density, data points fall on a line with a slope of approximately  $-\frac{3}{2}$  [called the self-thinning curve]. [...] When seeds are planted at a moderate density, so that the beginning combination of density and average dry weight lies below the self-thinning curve, plants grow without appreciable mortality until the population reaches its self-thinning curve. After that point, the intense crowding associated with further increase in average plant size causes the death of smaller individuals.”

Our model of self-thinning is a simplified version of that by Firbank and Watkinson [13]. The simulation starts with an initial set of circles, distributed at random in a square field, and assigned random initial radii from a given interval. If the circles representing two plants intersect, the smaller plant dies and its corresponding circle is removed from the scene. Plants that have reached a limit size are considered old and die as well.

Figure 4 shows three snapshots of the simulation. The corresponding plot shows the average area of the circles as a function of their density (Figure 5). The slope of the self-thinning curve is equal to  $-1$ ; assuming that mass is proportional to volume, which in turn is proportional to area raised to the power of  $-\frac{3}{2}$ , the self-thinning curve in the density-mass coordinates would have the slope of  $-\frac{3}{2}$ . Thus, in spite of its simplicity, our model captures the essential characteristic of plant distribution before and after it has reached the self-thinning curve.

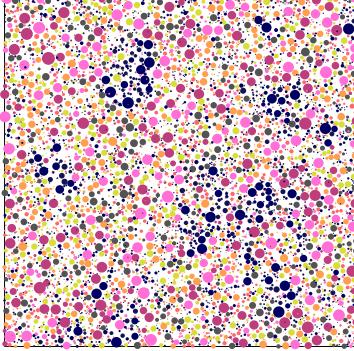


Figure 6: Simulated distribution of eight plant species in a terrain from Figure 3. Colors indicate plant species. Plants with a preference for wet areas are shown in blue.

A slightly more complicated model describes plant distribution in a population of different plant species. Each species is defined by a set of values that determine: (i) the number of new plants added to the field per simulation step, (ii) the maximum size of the plants, (iii) their average growth rate, (iv) the probability of surviving the domination by a plant with a competitive advantage, and (v) a preference for wet or dry areas. An individual plant is characterized by: (i) the species to which it belongs, (ii) its size, and (iii) its vigor. The vigor is a number in the range from 0 to 1, assigned to each plant as a randomized function of water concentration at the plant’s location and the plant’s preference for wet or dry areas. The competitive ability of a plant is determined as a product of its vigor and its relative size (the ratio between the actual and maximum size). When the circles representing two plants intersect, their competitive abilities are compared. The plant with a smaller competitive ability is dominated by the other plant and may die with the defined probability.

Figure 6 presents the result of a simulation involving a mix of eight plant species growing in a terrain shown in Figure 3. Plants with a preference for wet areas are represented by blue circles. Plants with a preference for dry areas have been assigned other colors. Through the competition between the species, a segregation of plants between the wet and dry areas has emerged.

Similar models can be developed to capture other phenomena that govern the development of plant populations.

## 5 MODELING OF PLANTS

Interactive editing of plant populations and the simulation of ecosystems determine positions and high-level characteristics of all plants in the modeled scene. On this basis, geometric models of individual plants must now be found.

Recent advances in plant measuring techniques have made it possible to construct a geometric model of a specific plant according to detailed measurements of its structure [54]. Nevertheless, for the purpose of visualizing plants differing by age, vigor, and possibly other parameters, it is preferable to treat geometric models as a product of the underlying procedural models. Construction of such models for computer graphics and biological purposes has been a field of active study, recently reviewed in [45]. Consequently, below we discuss only the issue of model parametrization, that is the incorporation of high-level parameters returned by the population model

into the plant models. We consider two different approaches, which reflect different predictive values of *mechanistic* and *descriptive* models [60].

Mechanistic models operate by simulating the processes that control the development of plants over time. They inherently capture how the resulting structure changes with age [46, 47]. If a mechanistic model incorporates environmental inputs, the dependence of the resulting structure on the corresponding environmental parameters is an emergent feature of the model [39]. The model predicts the effect of various combinations of environmental parameters on the structure, and no explicit parametrization is needed. L-systems [47] and their extensions [39] provide a convenient tool for expressing mechanistic models. Within EcoSys, mechanistic models have been generated using `cpfg`.

Descriptive models capture plant architecture without simulating the underlying developmental processes. Consequently, they do not have an inherent predictive value. Nevertheless, if a family of geometric models is constructed to capture the key “postures” of a plant at different ages and with different high-level characteristics, we can obtain the in-between geometries by interpolation. This is equivalent to fitting functions that map the set of high-level parameters to the set of lower-level variables present in the model, and can be accomplished by regression [57] (see [48] for an application example). In the special case of plant postures characterized by a single parameter, the interpolation between key postures is analogous to key-framing [62], and can be accomplished using similar techniques. We applied interpolation to parametrize models created using both `xfrog` and `cpfg`.

## 6 APPROXIMATE INSTANCING

Geometric plant models are often large. A detailed polygonal representation of a herbaceous plant may require over 10MB to store; a scene with one thousand plants (a relatively small number in ecosystem simulations) would require 10GB. One approach for reducing such space requirements is to simplify geometric representations of objects that have a small size on the screen. We incorporated a version of this technique into our system by parameterizing the procedural plant models so that they can produce geometries with different polygonizations of surfaces. However, this technique alone was not sufficient to reduce the amount of data to manageable levels.

Instancing was used successfully in the past for compactly representing complex outdoor scenes (*e.g.* [56]). According to the paradigm of instancing [59], geometric objects that are identical up to affine transformations become instances of one object. To achieve a further reduction in the size of geometric descriptions, we extended the paradigm of instancing to objects that resemble each other, but are not exactly the same. Thus, sets of similar plants are represented by instances of a single representative plant. Furthermore, the hierarchical structure of plant scenes, which may be decomposed into groups of plants, individual plants, branches of different order, plant organs such as leaves and flowers, *etc.*, lends itself to instancing at different levels of the hierarchy. We create hierarchies of instances by quantizing model components in their respective parameter spaces, and reusing them.

Automatic generation of instance hierarchies for plant models expressed using a limited class of L-systems was considered by Hart [20, 21]. His approach dealt only with exact instancing. Brownbill [7] considered special cases of approximate instancing of plants, and analyzed tradeoffs between the size of the geometric models and their perceived distortion (departure from the original

geometry caused by the reduction of diversity between the components). He achieved reductions of the database size ranging from 5:1 to 50:1 with a negligible visual impact on the generated images (a tree and a field of grass). This result is reminiscent of the observation by Smith [55] that the set of random numbers used in stochastic algorithms for generating fractal mountains and particle-system trees can be reduced to a few representative values without significantly affecting the perceived complexity of the resulting images.

We generalize Brownbill’s approach by relating it to clustering. Assuming that the characteristics of each plant are described by a vector of real numbers, we apply a clustering algorithm to the set of these vectors in order to find representative vectors. Thus, we reduce the problem of finding representative plants and instancing them to the problem of finding a set of representative points in the parameter space and mapping each point to its representative. We assume that plants with a similar appearance are described by close points in their parameter space; if this is not the case (for example, because the size of a plant is a nonlinear function of its age), we transform the parameter space to become perceptually more linear. We cluster and map plant parts in the same manner as the entire plants.

This clustering and remapping can be stated also in terms of vector quantization [18]: we store a code book of plants and plant parts and, for each input plant, we store a mapping to an object in the code book rather than the plant geometry itself. In computer graphics, vector quantization has been widely used for color image quantization [22]; more recent applications include reduction of memory needs for texture mapping [4] and representing light fields [31].

We use a multi-dimensional clustering algorithm developed by Wan *et al.* [61], which subdivides the hyperbox containing data points by choosing splitting planes to minimize the variance of the resulting clusters of data. We extended this algorithm to include an “importance weight” with each input vector. The weights make it possible to further optimize the plant quantization process, for example by allocating more representative vectors to the plants that occupy a large area of the screen.

## 7 RENDERING

Rendering natural scenes raises two important questions: (i) dealing with scene complexity, and (ii) simulating illumination, materials and atmospheric effects. Within **EcoSys**, we addressed these questions using two different strategies.

The first strategy is to split the scene into sub-scenes of manageable complexity, render each of them independently using ray-casting, and composite the resulting RGB $\alpha$ Z images into the final image [12]. The separation of the scene into sub-scenes is a byproduct of the modeling process: both `densedis` and `cpfg` can output the distribution of a single plant species to form a sub-scene. The ray-casting algorithm is implemented in `fshade`, which creates the scene geometry procedurally by invoking the `xfrog` plant modeler at run time. This reduces file I/O and saves disk space compared to storing all of the geometric information for the scene on disk and reading it in while rendering. For example, the poplar tree shown in Figure 16 is 16 KB as a procedural model (plant template), but 6.7 MB in a standard text geometry format.

A number of operations can be applied to the RGB $\alpha$ Z images before they are combined. Image processing operations, such as saturation and brightness adjustments, are often useful. Atmospheric effects can be introduced in a post process, by modifying colors according to the pixel depth. Shadows are computed using shadow maps [64].

The scene separation makes it possible to render the scene quickly and re-render its individual sub-scenes as needed to improve the image. However, complex lighting effects cannot be easily included, since the renderer doesn’t have access to the entire scene description at once.

The second rendering strategy is ray tracing. It lacks the capability to easily re-render parts of scenes that have been changed, but makes it possible to include more complex illumination effects. In both ray-tracers that we have used, `rayshade` [29] and `toro` [43], procedural geometry descriptions are expanded into triangle meshes, complemented with a hierarchy of grids and bounding boxes needed to speed up rendering [56]. `Rayshade` requires the entire scene description (object prototypes with their bounding boxes and a hierarchy of instancing transformations) to be kept in memory, otherwise page swapping significantly decreases the efficiency of rendering. In the case of `toro`, meshes are stored on disk; these are read in parts to a memory cache as needed for rendering computations and removed from memory when a prescribed limit amount of memory has been used. Consequently, the decrease in performance when the memory size has been reached is much slower [43]. We have made the straightforward extension of memory-coherent ray-tracing algorithms to manage instanced geometry: along with non-instanced geometry, the instances in the scene are also managed by the geometry cache.

Because rays can be traced that access the entire scene, more complex lighting effects can be included. For example, we have found that attenuating shadow rays as they pass through translucent leaves of tree crowns greatly improves their realism and visual richness.

## 8 EXAMPLES

We evaluated our system by applying it to create a number of scenes. In the examples presented below, we used two combinations of the modules: (i) ecosystem simulation and plant modeling using `cpfg` followed by rendering using `rayshade` or `toro`, and (ii) interactive specification of plant distribution using `densedis` in conjunction with plant generation using `xfrog` and rendering using `fshade`.

Figure 7 presents visualizations of two stages of the self-thinning process, based on distributions shown in Figure 4. The plants represent hypothetical varieties of *Lychnis coronaria* [47] with red, blue, and white flowers. Plant size values returned by the ecosystem simulation were quantized to seven representative values for each plant variety. The quantized values were mapped to the age of the modeled plants. The scene obtained after 99 simulation steps had 16,354 plants. The `rayshade` file representing this scene without instancing would be 3.5 GB (estimated); with instancing it was 6.7 MB, resulting in the compression ratio of approximately 500:1. For the scene after 164 steps, the corresponding values were: 441 plants, 125 MB, 5.8 MB, compression 21:1.

The mountain meadow (Figure 8 top) was generated by simulating an ecosystem of eight species of herbaceous plants, as discussed in Section 5. The distribution of plants is qualitatively similar to that shown schematically in Figure 6, but it includes a larger number of smaller plants. The individual plants were modeled with a high level of detail, which made it possible to zoom in on this scene and view individual plants. The complete scene has approximately 102,522 plants, comprising approximately  $2 \cdot 10^9$  primitives (polygons and cylinders). The `rayshade` file representing this scene without instancing would be 200 GB (estimated), with the instancing it was 151 MB, resulting in a compression ratio of approximately 1,300:1.

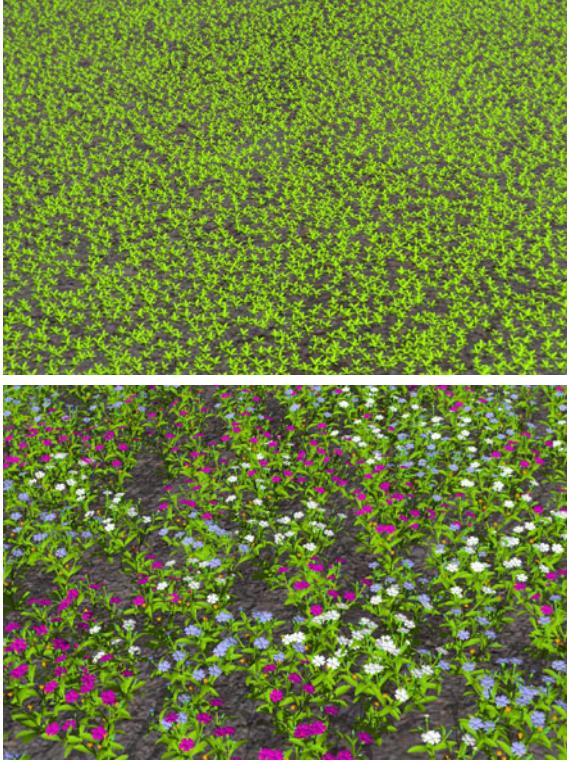


Figure 7: A *Lychnis coronaria* field after 99 and 164 simulation steps

The time needed to model this scene on a 150 MHz R5000 Silicon Graphics Indy with 96 MB RAM was divided as follows: simulation of the ecosystem (25 steps): 35 min, quantization (two-dimensional parameter space, each of the 8 plant species quantized to 7 levels): 5 min, generation of the 56 representative plants using `cpfg`: 9 min. The rendering time using `rayshade` on a 180 MHz R10000 Silicon Graphics Origin 200 with 256 MB RAM ( $1024 \times 756$  pixels, 4 samples per pixel) was approximately 8 hours. (It was longer using `toro`, but in that case the rendering time did not depend critically on the amount of RAM.)

In the next example, the paradigm of parameterizing, quantizing, and instancing was applied to entire groups of plants: tufts of grass with daisies. The number of daisies was controlled by a parameter (Figure 9). The resulting lawn is shown in Figure 10. For this image, ten different sets of grass tufts were generated, each instanced twenty times on average. The total reduction in geometry due to quantization and instancing (including instancing of grass blades and daisies within the tufts) was by a factor of 130:1. In Figure 11, a model parameter was used to control the size of the heaps of leaves. The heap around the stick and the stick itself were modeled manually.

Interactive creation of complex scenes requires the proper use of techniques to achieve an aesthetically pleasing result. To illustrate the process that we followed, we retrace the steps that resulted in the stream scene shown in Figure 15.

We began with the definition of a hilly terrain crossed by a little stream (Figure 2). To cover it with plants, we first created procedural models of plant species fitting this environment (Figure 12). Next, we extracted images representing the terrain altitudes and the stream position (Figures 13a and 13b) from the original terrain data. This

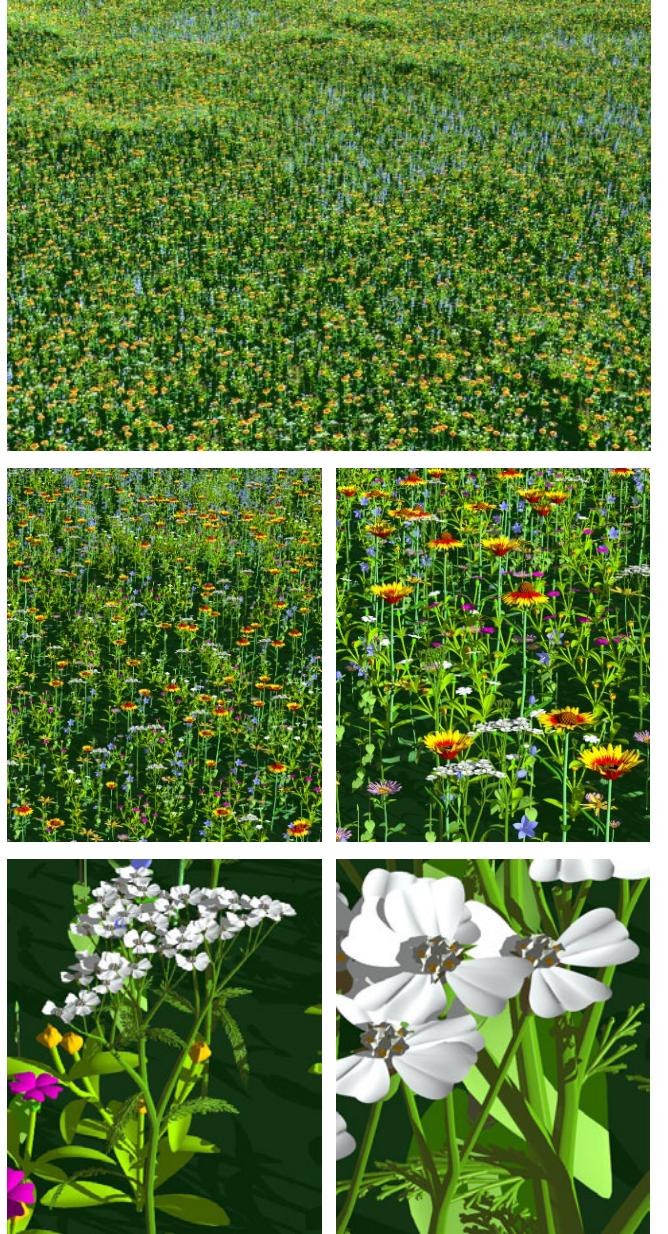


Figure 8: Zooming in on a mountain meadow

provided visual cues needed while painting plant distributions, for example, to prevent plants from being placed in the stream.

After that, we interactively chose a viewpoint, approximately at human height. With the resulting perspective view of the terrain as a reference, we painted a gray scale image for each plant species to define its distribution. We placed vegetation only in the areas visible from the selected viewpoint to speed up the rendering later on. For example, Figure 13c shows the image that defines the density distribution of stinging nettles. Since the stinging nettles grow on wet ground, we specified high density values along the stream. The density image served as input to `densedis`, which determined positions of individual plants. The dot diagram produced by `densedis` (Figure 13d) provided visual feedback that was used to refine the density image step by step until the desired distribution was found.



Figure 9: Grass tufts with varying daisy concentrations



Figure 10: Lawn with daisies

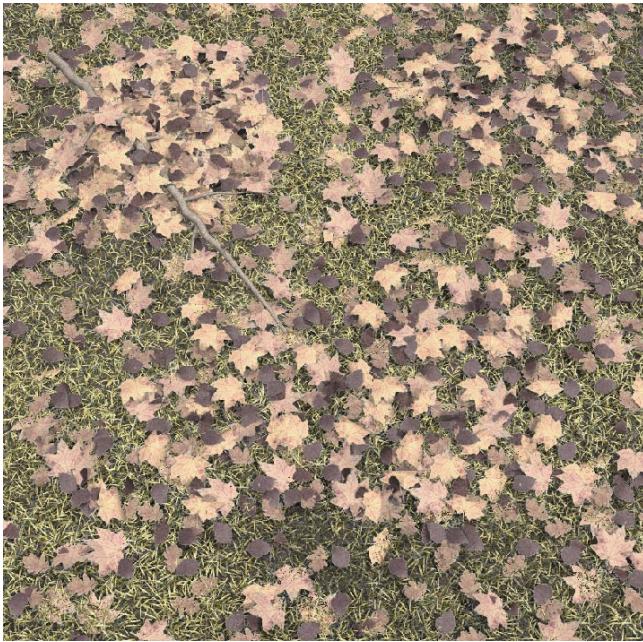


Figure 11: Leaves on grass

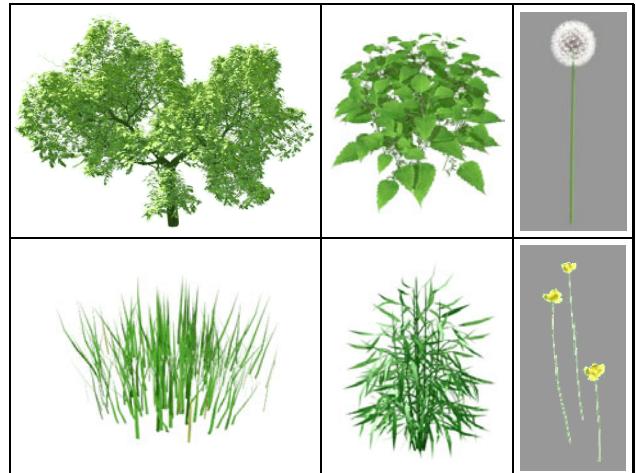


Figure 12: Sample plant models used in the stream scene. Top row: apple, stinging nettle, dandelion; bottom row: grass tuft, reed, yellow flower.

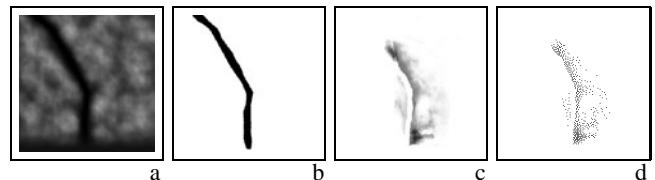


Figure 13: Creating distribution of stinging nettle: the heightmap of the covered area (a), the river image (b), the plant density distribution painted by the user (c), and the resulting plant positions (d).

Once the position of plants was established, we employed additional parameters to control the appearance of the plants. The vigor of stinging nettle plants, which affects the length of their stems and the number of leaves, was controlled using the density image for the nettles. To control the vigor of grass we used the height map: as a result, grass tufts have a slightly less saturated color on top of the hill than in the lower areas. Each tuft was oriented along a weighted sum of the terrain's surface normal vector and the up vector.

At this point, the scene was previewed and further changes in the density image were made until a satisfying result was obtained.

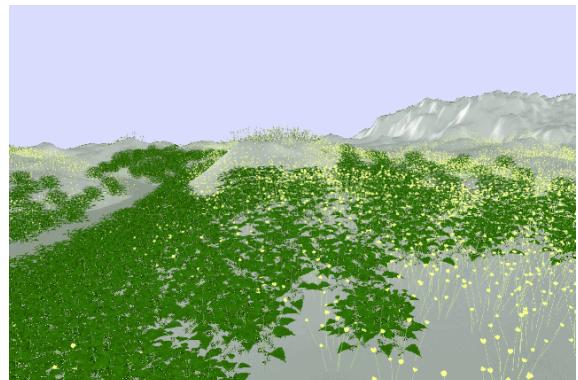


Figure 14: OpenGL preview of the stream scene including stinging nettle and yellow flowers

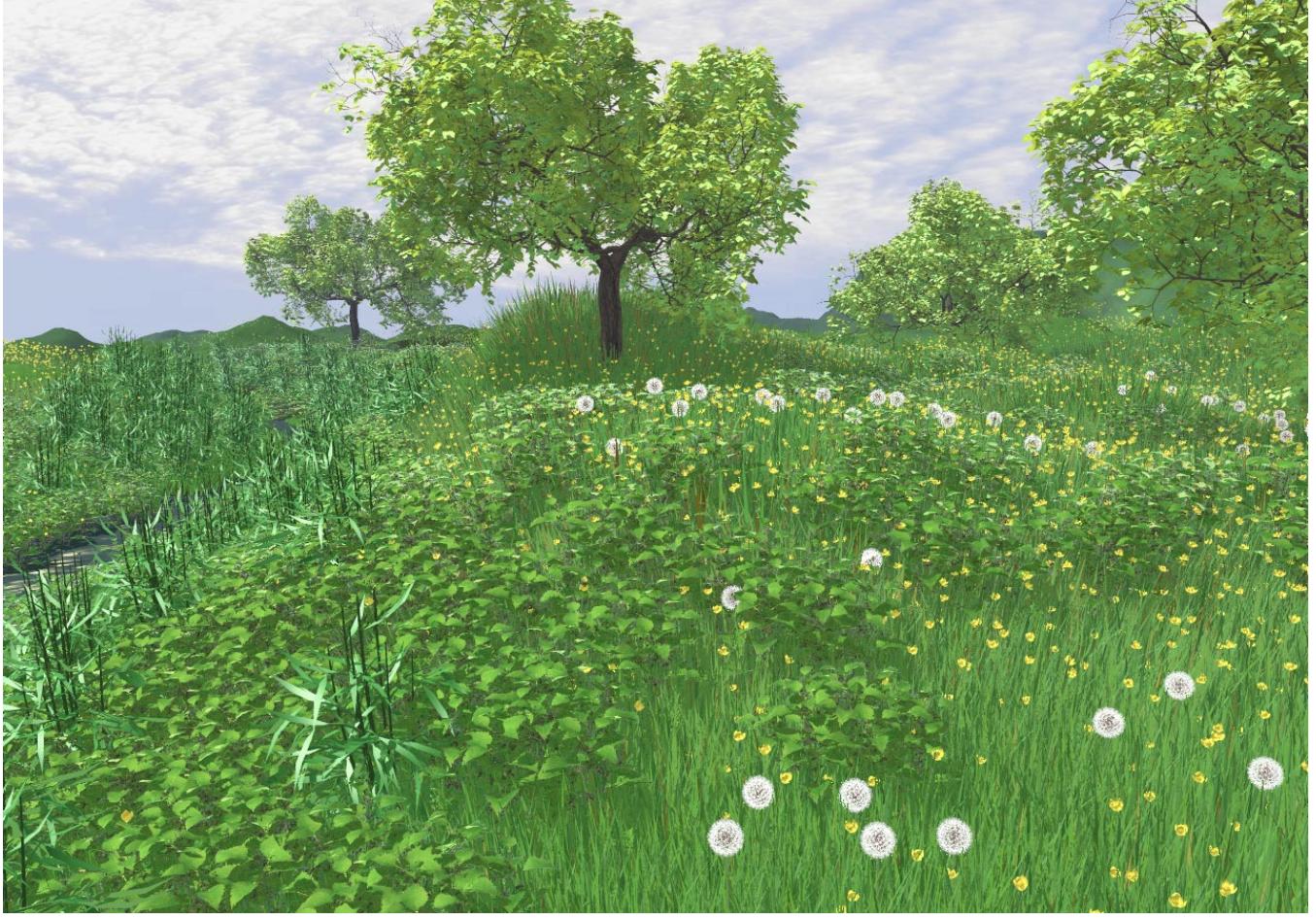


Figure 15: Stream scene

Figure 14 shows the preview of the distribution of stinging nettles and yellow flowers. To prevent intersections between these plants, the painted density image for the yellow flowers was multiplied by the inverted density image for the nettles.

The apple trees were placed by painting black dots on a white image. The final scene (Figure 15) was rendered using `fshade`. Images representing each species were rendered separately, and the resulting sub-scenes were composited as described in Section 7. The clouds were then added using a real photograph as a texture map. To increase the impression of depth in the scene, color saturation and contrast were decreased with increasing depth in a postprocessing step, and colors were shifted towards blue. Table 1 provides statistics about the instancing and geometric compression for this scene. The creation of this image took two days plus one day for defining the plant models. The actual compute time needed to synthesize this scene on a 195 MHz R10000 8-processor Silicon Graphics Onyx with 768MB RAM ( $1024 \times 756$  pixels, 9 samples per pixel) was 75 min.

Figures 16 and 17 present further examples of scenes with interactively created plant distributions. To simulate the effect of shadowing on the distribution of the yellow flowers in Figure 16, we rendered a top view of the spheres that approximate the shape of the apple trees, and multiplied the resulting image (further modified interactively) with the initial density image for the yellow flowers. We followed a similar strategy in creating Figure 17: the most impor-

tant trees were positioned first, then rendered from above to provide visual cues for the further placements. Table 2 contains statistics about the geometry quantization in Figure 17.

plant	obj.	inst.	plant	obj.	inst.
apple	1	4	grass tuft	15	2577
reed	140	140	stinging nettle	10	430
dandelion	10	55	yellow flower	10	2751

Table 1: Number of prototype objects and their instances in the stream scene (Figure 15). Number of polygons without instancing: 16,547,728, with instancing: 992,216. Compression rate: 16.7:1.

plant	obj.	inst.	plant	obj.	inst.
weeping willow	16	16	reed	15	35
birch	43	43	poppy	20	128
distant tree	20	119	cornflower	72	20
St. John's wort	20	226	dandelion	20	75
grass tuft	15	824			

Table 2: Number of prototype objects and their instances in the Dutch scene (Figure 17). Number of polygons without instancing: 40,553,029, with instancing: 6,737,036. Compression rate: 6.0:1



Figure 16: Forest scene



Figure 17: Dutch landscape

## 9 CONCLUSIONS

We presented the design and experimental implementation of a system for modeling and rendering scenes with many plants. The central issue of managing the complexity of these scenes was addressed with a combination of techniques: the use of different levels of abstraction at different stages of the modeling and rendering pipeline, procedural modeling, approximate instancing, and the employment of space- and time-efficient rendering methods. We tested our system by generating a number of visually complex scenes. Consequently, we are confident that the presented approach is operational and can be found useful in many practical applications.

Our work is but an early step in the development of techniques for creating and visualizing complex scenes with plants, and the presented concepts require further research. A fundamental problem is the evaluation of the impact of quantization and approximate instancing on the generated scenes. The difficulty in studying this problem stems from: (i) the difficulty in generating non-instanced reference images for visual comparison purposes (the scenes are too large), (ii) the lack of a formally defined error metric needed to evaluate the artifacts of approximate instancing in an objective manner, and (iii) the difficulty in generalizing results that were obtained by the analysis of specific scenes. A (partial) solution to this problem would set the stage for the design and analysis of methods that may be more suitable for quantizing plants than the general-purpose variance-based algorithm used in our implementation.

Other research problems exposed by our experience with EcoSys include: (i) improvement of the terrain model through its coupling with the plant population model (in nature vegetation affects terrain, for example by preventing erosion); (ii) design of algorithms for converting plant densities to positions, taking into account statistical properties of plant distributions found in natural ecosystems [66]; (iii) incorporation of morphogenetic plasticity (dependence of the plant shape on its neighbors [58]) into the multi-level modeling framework; this requires transfer of information about plant shapes between the population model and the procedural plant models; (iv) extension of the modeling method presented in this paper to animated scenes (with growing plants and plants moving in the wind); (v) design of methods for conveniently previewing scenes with billions of geometric primitives (for example, to select close views of details); and (vi) application of more faithful local and global illumination models to the rendering of plant scenes (in particular, consideration of the distribution of diffuse light in the canopy).

## Acknowledgements

We would like to acknowledge Craig Kolb for his implementation of the variance-based quantization algorithm, which we adapted to the needs of our system, and Christain Jacob for his experimental implementations and discussions pertinent to the individual-based ecosystem modeling. We also thank: Stefania Bertazzon, Jim Hanan, Richard Levy, and Peter Room for discussions and pointers to the relevant literature, the referees for helpful comments on the manuscript, Chris Prusinkiewicz for editorial help, and Darcy Grant for system support in Calgary. This research was sponsored in part by the National Science Foundation grant CCR-9508579-001 to Pat Hanrahan, and by the Natural Sciences and Engineering Research Council of Canada grant OGP0130084 to Przemyslaw Prusinkiewicz.

## REFERENCES

- [1] Alias/Wavefront; a division of Silicon Graphics Ltd. Studio V8. SGI program, 1996.
- [2] AnimaTek, Inc. AnimatTek's World Builder. PC program, 1996.
- [3] R. A. Armstrong. A comparison of index-based and pixel-based neighborhood simulations of forest growth. *Ecology*, 74(6):1707–1712, 1993.
- [4] A. C. Beers, M. Agrawala, and N. Chaddha. Rendering from compressed textures. In *SIGGRAPH 96 Conference Proceedings*, pages 373–378, August 1996.
- [5] B. M. Blumberg and T. A. Galyean. Multi-level direction of autonomous creatures for real-time virtual environments. In *SIGGRAPH 95 Conference Proceedings*, pages 47–54, August 1995.
- [6] B.N. Boots. *Spatial tessellations: concepts and applications of Voronoi diagrams*. John Wiley, 1992.
- [7] A. Brownbill. Reducing the storage required to render L-system based models. Master's thesis, University of Calgary, October 1996.
- [8] N. Chiba, K. Muraoka, A. Doi, and J. Hosokawa. Rendering of forest scenery using 3D textures. *The Journal of Visualization and Computer Animation*, 8:191–199, 1997.
- [9] Adobe Corporation. Adobe Photoshop.
- [10] O. Deussen and B. Lintemann. A modelling method and user interface for creating plants. In *Proceedings of Graphics Interface 97*, pages 189–197, May 1997.
- [11] J. Dorsey, H. Køhling Pedersen, and P. Hanrahan. Flow and changes in appearance. In *SIGGRAPH 96 Conference Proceedings*, pages 411–420, August 1996.
- [12] T. Duff. Compositing 3-D rendered images. *Computer Graphics (SIGGRAPH 85 Proceedings)*, 19(3):41–44, 1985.

- [13] F. G. Firbank and A. R. Watkinson. A model of interference within plant monocultures. *Journal of Theoretical Biology*, 116:291–311, 1985.
- [14] K. W. Fleischer, D. H. Laidlaw, B. L. Currin, and A. H. Barr. Cellular texture generation. In *SIGGRAPH 95 Conference Proceedings*, pages 239–248, August 1995.
- [15] R. W. Floyd and L. Steinberg. An adaptive algorithm for spatial gray scale. In *SID 75, Int. Symp. Dig. Tech. Papers*, pages 36–37, 1975.
- [16] D. R. Fowler, P. Prusinkiewicz, and J. Battjes. A collision-based model of spiral phyllotaxis. *Computer Graphics (SIGGRAPH 92 Proceedings)*, 26(2):361–368, 1992.
- [17] G. Y. Gardner. Simulation of natural scenes using textured quadric surfaces. *Computer Graphics (SIGGRAPH 84 Proceedings)*, 18(3):11–20, 1984.
- [18] A. Gersho and R. M. Gray. *Vector quantization and signal compression*. Kluwer Academic Publishers, 1991.
- [19] D. G. Green. Modelling plants in landscapes. In M. T. Michalewicz, editor, *Plants to ecosystems. Advances in computational life sciences I*, pages 85–96. CSIRO Publishing, Melbourne, 1997.
- [20] J. C. Hart and T. A. DeFanti. Efficient anti-aliased rendering of 3D linear fractals. *Computer Graphics (SIGGRAPH 91 Proceedings)*, 25:91–100, 1991.
- [21] J. C. Hart. The object instancing paradigm for linear fractal modeling. In *Proceedings of Graphics Interface 92*, pages 224–231, 1992.
- [22] P. Heckbert. Color image quantization for frame buffer display. *Computer Graphics (SIGGRAPH 82 Proceedings)*, 16:297–307, 1982.
- [23] S. I. Higgins and D. M. Richardson. A review of models of alien plant spread. *Ecological Modelling*, 87:249–265, 1996.
- [24] H. Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH 97 Conference Proceedings*, pages 189–198, August 1997.
- [25] D. H. House, G. S. Schmidt, S. A. Arvin, and M. Kitagawa-DeLeon. Visualizing a real forest. *IEEE Computer Graphics and Applications*, 18(1):12–15, 1998.
- [26] J. T. Kajiya and T. L. Kay. Rendering fur with three dimensional textures. *Computer Graphics (SIGGRAPH 89 Proceedings)*, 23(3):271–289, 1989.
- [27] T. L. Kay and J. T. Kajiya. Ray tracing complex scenes. *Computer Graphics (SIGGRAPH 86 Proceedings)*, 20(4):269–278, 1986.
- [28] A. D. Kelley, M. C. Malin, and G. M. Nielson. Terrain simulation using a model of stream erosion. *Computer Graphics (SIGGRAPH 88 Proceedings)*, 22(4):263–268, 1988.
- [29] C. Kolb. Rayshade. <http://graphics.stanford.edu/~cek/rayshade>.
- [30] M. P. Kumler. An intensive comparison of triangulated irregular networks (TINs) and digital elevation models (DEMs). *Cartographica*, 31(2), 1994.
- [31] M. Levoy and P. Hanrahan. Light field rendering. In *SIGGRAPH 96 Conference Proceedings*, pages 31–42, August 1996.
- [32] B. Lintemann and O. Deussen. Interactive structural and geometrical modeling of plants. To appear in the *IEEE Computer Graphics and Applications*.
- [33] B. Lintemann and O. Deussen. Interactive modelling and animation of natural branching structures. In R. Boulic and G. Hégron, editors, *Computer Animation and Simulation 96*. Springer, 1996.
- [34] Lucasfilm Ltd. The Adventures of André and Wally B. Film, 1984.
- [35] D. Marshall, D. S. Fussel, and A. T. Campbell. Multiresolution rendering of complex botanical scenes. In *Proceedings of Graphics Interface 97*, pages 97–104, May 1997.
- [36] N. Max. Hierarchical rendering of trees from precomputed multi-layer Z-buffers. In X. Pueyo and P. Schröder, editors, *Rendering Techniques 96*, pages 165–174 and 288. Springer Wien, 1996.
- [37] N. Max and K. Ohnsaki. Rendering trees from precomputed Z-buffer views. In P. M. Hanrahan and W. Purgathofer, editors, *Rendering Techniques 95*, pages 74–81 and 359–360. Springer Wien, 1995.
- [38] F. K. Musgrave, C. E. Kolb, and R. S. Mace. The synthesis and rendering of eroded fractal terrains. *Computer Graphics (SIGGRAPH 89 Proceedings)*, 23(3):41–50, 1989.
- [39] R. Měčík and P. Prusinkiewicz. Visual models of plants interacting with their environment. In *SIGGRAPH 96 Conference Proceedings*, pages 397–410, August 1996.
- [40] F. Neyret. A general and multiscale model for volumetric textures. In *Proceedings of Graphics Interface 95*, pages 83–91, 1995.
- [41] F. Neyret. Synthesizing verdant landscapes using volumetric textures. In X. Pueyo and P. Schröder, editors, *Rendering Techniques 96*, pages 215–224 and 291, Wien, 1996. Springer-Verlag.
- [42] K. Perlin. An image synthesizer. *Computer Graphics (SIGGRAPH 85 Proceedings)*, 19(3):287–296, 1985.
- [43] M. Pharr, C. Kolb, R. Gershbein, and P. Hanrahan. Rendering complex scenes with memory-coherent ray tracing. In *SIGGRAPH 97 Conference Proceedings*, pages 101–108, August 1997.
- [44] P. Prusinkiewicz. Visual models of morphogenesis. *Artificial Life*, 1(1/2):61–74, 1994.
- [45] P. Prusinkiewicz. Modeling spatial structure and development of plants: a review. *Scientia Horticulturae*, 74(1/2), 1998.
- [46] P. Prusinkiewicz, M. Hammel, and E. Mjolsness. Animation of plant development. In *SIGGRAPH 93 Conference Proceedings*, pages 351–360, August 1993.
- [47] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag, New York, 1990. With J. S. Hanan, F. D. Fracchia, D. R. Fowler, M. J. M. de Boer, and L. Mercer.
- [48] P. Prusinkiewicz, W. Remphrey, C. Davidson, and M. Hammel. Modeling the architecture of expanding *Fraxinus pennsylvanica* shoots using L-systems. *Canadian Journal of Botany*, 72:701–714, 1994.
- [49] Questar Productions, LLC. World Construction Set Version 2. PC program, 1997.
- [50] W. T. Reeves and R. Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. *Computer Graphics (SIGGRAPH 85 Proceedings)*, 19(3):313–322, 1985.
- [51] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics (SIGGRAPH 87 Proceedings)*, 21(4):25–34, 1987.
- [52] R. E. Ricklefs. *Ecology. Third Edition*. W. H. Freeman, New York, 1990.
- [53] J. Shade, D. Lischinski, D. Salesin, T. DeRose, and J. Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *SIGGRAPH 96 Conference Proceedings*, pages 75–82, August 1996.
- [54] H. Sinoquet and R. Rivet. Measurement and visualization of the architecture of an adult tree based on a three-dimensional digitising device. *Trees*, 11:265–270, 1997.
- [55] A. R. Smith. Plants, fractals, and formal languages. *Computer Graphics (SIGGRAPH 84 Proceedings)*, 18(3):1–10, 1984.
- [56] J. M. Snyder and A. H. Barr. Ray tracing complex models containing surface tessellations. *Computer Graphics (SIGGRAPH 87 Proceedings)*, 21(4):119–128, 1987.
- [57] R. R. Sokal and F. J. Rohlf. *Biometry. Third Edition*. W. H. Freeman, New York, 1995.
- [58] K. A. Sorrensen-Cothen, E. D. Ford, and D. G. Sprugel. A model of competition incorporating plasticity through modular foliage and crown development. *Ecological Monographs*, 63(3):277–304, 1993.
- [59] I. E. Sutherland. Sketchpad: A man-machine graphical communication system. Proceedings of the Spring Joint Computer Conference, 1963.
- [60] J. H. M. Thornley and I. R. Johnson. *Plant and crop modeling: A mathematical approach to plant and crop physiology*. Oxford University Press, New York, 1990.
- [61] S. J. Wan, S. K. M. Wong, and P. Prusinkiewicz. An algorithm for multidimensional data clustering. *ACM Trans. Math. Software*, 14(2):135–162, 1988.
- [62] A. Watt and M. Watt. *Advanced animation and rendering techniques: Theory and practice*. Addison-Wesley, Reading, 1992.
- [63] J. Weber and J. Penn. Creation and rendering of realistic trees. In *SIGGRAPH 95 Conference Proceedings*, pages 119–128, August 1995.
- [64] L. Williams. Casting curved shadows on curved surfaces. *Computer Graphics (SIGGRAPH 78 Proceedings)*, 12(3):270–274, 1978.
- [65] L. Williams. Shading in two dimensions. In *Proceedings of Graphics Interface 91*, pages 143–151, June 1991.
- [66] H. Wu, K. W. Malafant, L. K. Pendridge, P. J. Sharpe, and J. Walker. Simulation of two-dimensional point patterns: application of a lattice framework approach. *Ecological Modelling*, 38:299–308, 1997.

# Artificial Evolution for Computer Graphics

Karl Sims

Thinking Machines Corporation  
245 First Street, Cambridge, MA 02142

## 1 ABSTRACT

This paper describes how evolutionary techniques of variation and selection can be used to create complex simulated structures, textures, and motions for use in computer graphics and animation. Interactive selection, based on visual perception of procedurally generated results, allows the user to direct simulated evolutions in preferred directions. Several examples using these methods have been implemented and are described. 3D plant structures are grown using fixed sets of genetic parameters. Images, solid textures, and animations are created using mutating symbolic lisp expressions. Genotypes consisting of symbolic expressions are presented as an attempt to surpass the limitations of fixed-length genotypes with predefined expression rules. It is proposed that artificial evolution has potential as a powerful tool for achieving flexible complexity with a minimum of user input and knowledge of details.

## 2 INTRODUCTION

Procedural models are increasingly employed in computer graphics to create scenes and animations having high degrees of complexity. A price paid for this complexity is that the user often loses the ability to maintain sufficient control over the results. Procedural models can also have limitations because the details of the procedure must be conceived, understood, and designed by a human. The techniques presented here contribute towards solutions to these problems by enabling “evolution” of procedural models using interactive “perceptual selection.” Although they do not give complete control over every detail of the results, they do permit the creation of a large variety of complex entities which are still user directed, and the user is not required to understand the underlying creation process involved.

Many years ago Charles Darwin proposed the theory that all species came about via the process of evolution [2]. Evolution is now considered not only powerful enough to bring about biological entities as complex as humans and consciousness, but also useful in simulation to create algorithms and structures of higher levels of complexity than could easily be built by design. Genetic algorithms have shown to be a useful method of searching large spaces using simulated systems of variation and selection [23][7][6][5]. In *The Blind Watchmaker*, Dawkins has demonstrated the power of Darwinism with a simulated evolution of 2D branching structures made from sets of genetic parameters. The user selects the “biomorphs” that survive and reproduce to create each new generation [4][3]. Latham and Todd have applied these concepts to help generate computer sculptures made with constructive solid geometry techniques [28][9].

Variations on these techniques are used here with the emphasis on the potential of creating forms, textures, and motions that are useful in the production of computer graphics and animation, and also on the potential of using representations that are not bounded by a fixed space of possible results.

## 2.1 Evolution

Both biological and simulated evolutions involve the basic concepts of genotype and phenotype, and the processes of expression, selection, and reproduction with variation.

The *genotype* is the genetic information that codes for the creation of an individual. In biological systems, genotypes are normally composed of DNA. In simulated evolutions there are many possible representations of genotypes, such as strings of binary digits, sets of procedural parameters, or symbolic expressions. The *phenotype* is the individual itself, or the form that results from the developmental rules and the genotype. *Expression* is the process by which the phenotype is generated from the genotype. For example, expression can be a biological developmental process that reads and executes the information from DNA strands, or a set of procedural rules that utilize a set of genetic parameters to create a simulated structure. Usually, there is a significant amplification of information between the genotype and phenotype.

*Selection* is the process by which the fitness of phenotypes is determined. The likelihood of survival and the number of new offspring an individual generates is proportional to its fitness measure. *Fitness* is simply the ability of an organism to survive and reproduce. In simulation, it can be calculated by an explicitly defined fitness evaluation function, or it can be provided by a human observer as it is in this work.

*Reproduction* is the process by which new genotypes are generated from an existing genotype or genotypes. For evolution to progress there must be *variation* or mutations in new genotypes with some frequency. Mutations are usually probabilistic as opposed to deterministic. Note that selection is, in general, non-random and is performed on phenotypes; variation is usually random and is performed on the corresponding genotypes [See figure 1].

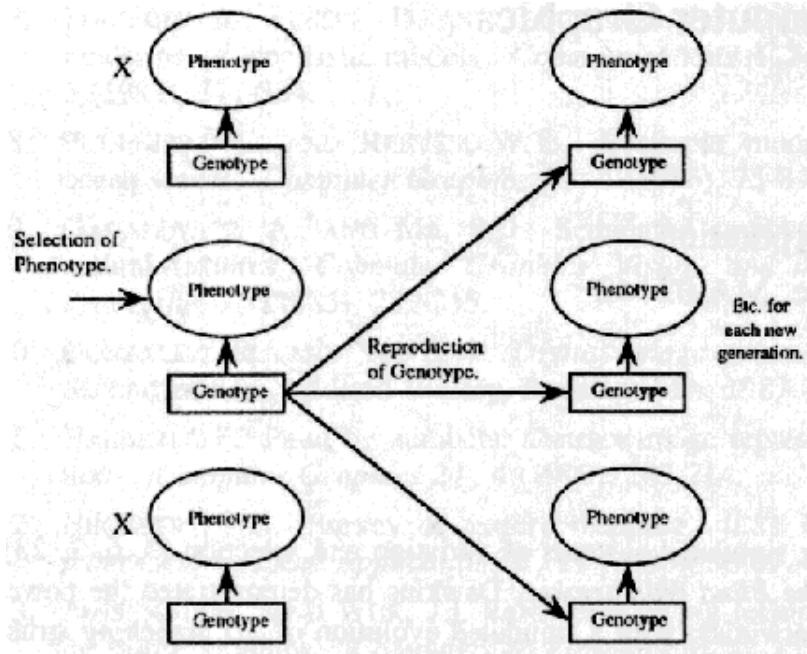


Figure 1: Phenotype selection, genotype reproduction.

The repeated cycle of reproduction with variation and selection of the most fit individuals drives the evolution of a population towards higher and higher levels of fitness.

*Sexual combination* can allow genetic material of more than one parent to be mixed together in some way to create new genotypes. This permits features to evolve independently and later be combined into a single individual. Although it is not necessary for evolution to occur, it is a valuable practice that can enhance progress in both biological and simulated evolutions.

## 2.2 Genetic Algorithms

Genetic algorithms were first developed by Holland [11] as robust searching techniques in which populations of test points are evolved by random variation and selection. They have become widely used in a number of applications to find optima in very large search spaces [23][7][6].

Genetic algorithms differ from the examples presented in this paper in that they usually utilize an explicit analytic function to measure the fitness of phenotypes. Since it is difficult to automatically measure the aesthetic visual success of simulated objects or images, here the fitness is provided by a human user based on visual perception. Some combinations of automatic selection and interactive selection are also utilized.

Population sizes used for genetic algorithms are usually fairly large (100 to 1000 or more) to allow searching of many test points and avoiding only local optima. At each generation, many individuals survive and reproduce to create the next generation. For the examples presented in this paper, the success of a solution is dependent on human opinion, therefore there is no single global optimum. Many local optima are potentially interesting solutions. For this reason, and also because of user interface practicality, a smaller population size has been used (20 - 40), and only one or two individuals are

chosen to reproduce for each new generation.

Genotypes used in genetic algorithms traditionally consist of fixed-length character strings used by fixed expression rules. This is appropriate for searching predefined dimensional spaces for optimum solutions, but these restrictions are sometimes limiting. Koza [13][12] has used hierarchical lisp expressions as genotypes such that the dimensionality of the search space itself can be extended to successfully solve problems such as artificial ant navigation and game strategies. Discovery systems, such as AM, Eurisko, and Cyrano, also utilize a form of mutating lisp programs [14][8]. The examples of evolving images, volume textures, and animations presented here also use genotypic representations composed of lisp expressions, although the set of functions used includes various vector transformations, noise generators, and image processing operations, as well as standard numerical functions.

In the next section, techniques for using artificial evolution to explore samples in parameter spaces are discussed. In section 4, examples of evolving images, volume textures, and animations which utilize mutating symbolic expressions as genotypes are presented. Finally, results, suggestions for future work, and conclusions are given in the last three sections.

### 3 EXPLORING PARAMETER SPACES

Procedural models such as fractals, graftals, and procedural texturing allow a user to create a high degree of complexity with relatively simple input information [25][21][19][18]. One method of procedural structure creation involves a set of  $N$  input parameters each of which has an effect on a developmental process which assembles the structure. The set of possible structures corresponds to the  $N$ -dimensional space of possible parameter values. Consider an array of knobs, each controlling one parameter, that can be experimentally turned to adjust the results. As more options are added to the procedure for more variation of results, the number of input parameters grows and it can become increasingly difficult for a user to predict the effects of adjusting particular parameters and combinations of parameters, and to adjust the knobs effectively by hand.

An alternative approach is to sample randomly in the neighborhood of a currently existing parameter set by making random alterations to a parameter or several parameters, then inspect and select the best sample or samples of those presented. This allows exploration through the parameter space in incremental arbitrary directions without requiring knowledge of the specific effects of each parameter. This is artificial evolution in which the genotype is the parameter set, and the phenotype is the resulting structure. Selection is performed by the user picking preferred phenotypes from groups of samples, and as long as the samples can be generated and displayed quickly enough, it can be a useful technique.

#### 3.1 Evolving 3D Plant Structures

The first example of artificial evolution involves 3D plant structures which can be grown using a set of “genetic” parameters. Plant generation algorithms of various types have been shown to be useful examples of procedurally generated structures [29][25][22][21][16][1]. The model used in this work is described briefly below, but details have been omitted as the emphasis is on the evolutionary process.

Parameters describing fractal limits, branching factors, scaling, stochastic contributions, etc., are used to generate 3-dimensional tree structures consisting of connected segments. Growth rules use 21 genetic parameters and the hierarchy location of each segment in the tree to determine how fast that segment should grow, when it should generate new buds, and in which directions. The tree structures are grown

in arbitrarily small increments for smooth simulation of development.

After a desirable tree structure has been evolved using interactive selection and the mutation methods described below, its phenotype can be saved for further manipulation. Solid polygonal branches can be generated with connected cylinders and cone shapes, and leaves can be generated by connecting sets of peripheral nodes with polygonal surfaces. Shading parameters, color, and bump textures can be assigned to make bark and leaf surfaces. These additional properties could also be selected and adjusted using artificial evolution, but due to the longer computation times involved to test samples, these parameters were adjusted by hand. In some cases, leaf shapes were evolved independently and then explicitly added to the tip segments of other evolved plant structures. A forest of plant structures created using these methods is shown in figure 3.

### 3.2 Mutating Parameter Sets

For artificial evolution of parameter sets to occur, they must be reproduced with some probability of mutation. There are many possible methods for mutating parameter sets. The technique used here involves normalizing each parameter for a genetic value between .0 and 1.0, and then copying each genetic value or gene,  $g_i$ , from the parent to the child with a certain probability of mutation,  $m$ . A mutation is achieved by adding a random amount,  $+d$ , to the gene. So, a new genotype,  $G'$ , is created using each gene,  $g'_i$ , of a parent genotype,  $G$ , as follows:

```
For each  $g_i$ 
  If  $\text{rand}(.0, 1.0) < m$ 
    then  $g'_i = g_i + \text{rand}(-d, d)$ 
        clamp or wrap  $g'_i$  to legal bounds.
    else  $g'_i = g_i$ 
```

The normalized values are scaled, offset, and optionally squared to give the parameter values actually used. This allows the mutation distances,  $+d$ , to be proportional to the scale of the range of valid parameter values. Squaring or raising some values to even higher powers can be useful because it causes more sensitivity in the lower region of the range of parameter values. The mutation rate and amount are easily adjusted, but are commonly useful at much higher values than in natural systems ( $m=0.2$ ,  $d=0.4$ ). The random value between  $-d$  and  $d$  might preferably be found using a Gaussian distribution instead of this simple linear distribution, giving smaller mutations more likelihood than larger ones.

### 3.3 Mating Parameter Sets

When two parameter sets are found that both create structures with different successful features, it is sometimes desirable to combine these features into a single structure. This can be accomplished by mating them. Reproducing two parameter sets with sexual combination can be performed in many ways. Four possible methods are listed below with some of their resulting effects:

1. *Crossovers* can be performed by sequentially copying genes from one parent, but with some frequency the source genotype is switched to the other parent. This causes adjacent genes to be more likely to stick together than genes at opposite ends of the sequence. Each pair of genes has a *linkage* probability depending on their distance from each other.
2. Each gene can be independently copied from one parent or the other with equal probability. If the

parent genes each correspond to a point in N-dimensional genetic space, then the genes of the possible children using this method correspond to the  $2^N$  corners of the N-dimensional rectangular solid connecting the two parent points. This method is the most commonly used in this work and is demonstrated in figure 2. Two parent plant structures are shown in the upper left boxes, and the remaining forms are their children.

3. Each gene can receive a random percentage,  $p$ , of one parent's genes, and a  $1 - p$  percentage of the other parent's genes. If the percentage is the same for each gene, linear *interpolation* between the parent genotypes results, and the children will fall randomly on the line between the N-dimensional points of the parents. If evenly spaced samples along this line were generated, a *genetic dissolve* could be made that would cause a smooth transition between the parent phenotypes if the changing parameters had continuous effects on the phenotypes. This is an example of utilizing the underlying genetic representation for specific manipulation of the results. Interpolation could also be performed with three parents to create children that fall on a triangular region of a plane in the N-dimensional genetic space.
4. Finally, each new gene can receive a random value between the two parent values of that gene. This is like the interpolation scheme above, except each gene is independently interpolated between the parent genes. This method results in possible children anywhere within the N-dimensional rectangular solid connecting the parent points.

Mutating and mating parameter sets allow a user to explore and combine samples in a given parameter space. In the next section, methods are presented that allow mutations to add new parameters and extend the space, instead of simply adjusting existing parameter values.

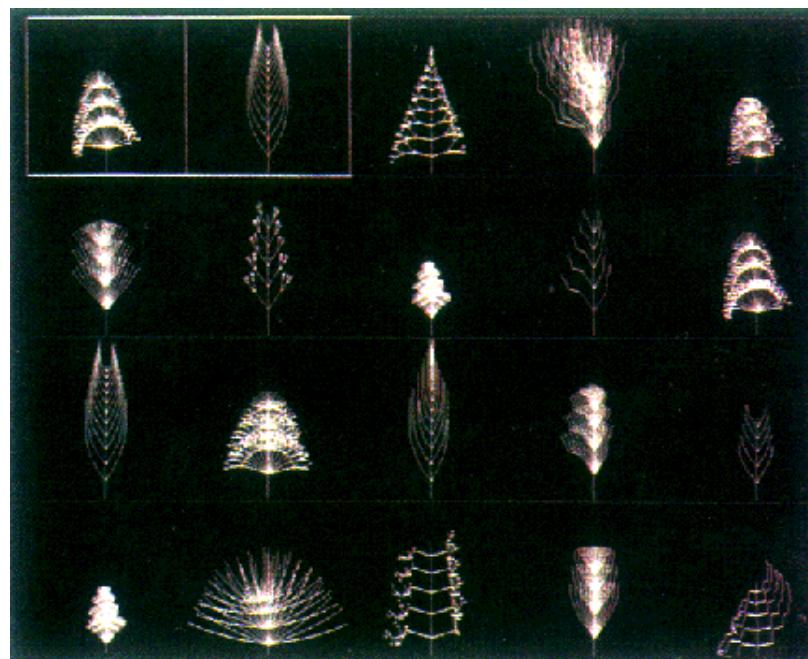


Figure 2: Mating plant structures.



Figure 3: Forest of “evolved” plants.

## 4 SYMBOLIC EXPRESSIONS AS GENOTYPES

A limitation of genotypes consisting of a fixed number of parameters and fixed expression rules as described above is that there are solid boundaries on the set of possible phenotypes. There is no possibility for the evolution of a new developmental rule or a new parameter. There is no way for the genetic space to be extended beyond its original definition - the N-dimensional genetic space will remain only N-dimensional.

To surpass this limitation, it is desirable to include procedural information in the genotype instead of just parameter data, and the procedural and data elements of the genotype should not be restricted to a specific structure or size.

Symbolic lisp expressions are used as genotypes in an attempt to meet these needs. A set of lisp functions and a set of argument generators are used to create arbitrary expressions which can be mutated, evolved, and evaluated to generate phenotypes. Some mutations can create larger expressions with new parameters and extend the space of possible phenotypes, while others just adjust existing parts of the expression. Details of this process are best described by the examples below.

### 4.1 Evolving Images

The second example of artificial evolution involves the generation of textures by mutating symbolic expressions. Equations that calculate a color for each pixel coordinate  $(x,y)$  are evolved using a *function set* containing some standard common lisp functions [26], vector transformations, procedural noise generators, and image processing operations:

`+, -, *, /, mod, round, min, max, abs, expt, log, and,`

*or, xor, sin, cos, atan, if, dissolve, hsv-to-rgb, vector, transform-vector, bw-noise, color-noise, warped-bw-noise, warped-color-noise, blur, band-pass, grad-mag, grad-dir, bump, ifs, warped-ifs, warp-abs, warp-rel, warp-by-gra*.

Each function takes a specified number of arguments and calculates and returns an image of scalar (b/w) or vector (color) values.

Noise generators can create solid 2D scalar and vector noise at various frequencies with random seeds passed as arguments so specific patterns can be preserved between generations [figure 4f, and 4i]. The warped versions of functions take  $(U, V)$  coordinates as arguments instead of using global  $(X, Y)$  pixel coordinates, allowing the result to be distorted by an arbitrary inverse mapping function [figure 4i]. Boolean operations (*and*, *or*, and *xor*) operate on each bit of floating-point numbers and can cause fractal-like grid patterns [figure 4e]. Versions of *sin* and *cos* which normalize their results between .0 and 1.0 instead of -1.0 and 1.0 can be useful. Some functions such as blurs, convolutions, and those that use gradients also use neighboring pixel values to calculate their result [figure 4h]. *Band-pass* convolutions can be performed using a difference of Gaussians filter which can enhance edges. Iterative function systems (*ifs*) can generate fractal patterns and shapes.

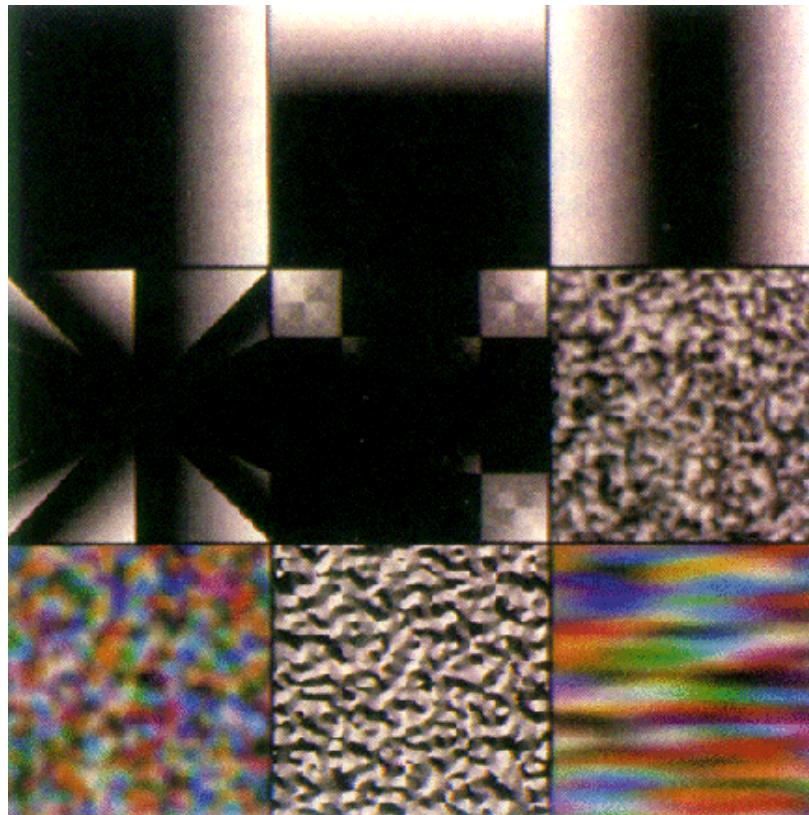


Figure 4: Simple expression examples.

(reading left to right, top to bottom)

- a.  $X$
- b.  $Y$
- c.  $(\text{abs } X)$

```

d. (mod X (abs Y))
e. (and X Y)
f. (bw-noise .2 2)
g. (color-noise .1 2)
h. (grad-direction (bw-noise .15 2) .0 .0)
i. (warped-color-noise (* X .2) Y .1 2)

```

Details of the specific implementations of these functions are not given here because they are not as important as the methods used for combining them into longer expressions. Many other functions would be interesting to include in this *function set*, but these have provided for a fairly wide variety of resulting images.

Simple random expressions are generated by choosing a function at random from the *function set* above, and then generating as many random arguments as that function requires. Arguments to these functions can be either scalars or vectors, and either constant values or images of values. Random arguments can be generated from the following forms:

- A random scalar value such as .4
- A random 3-element vector such as #(42 23 69)
- A variable such as the *X* or *Y* pixel coordinates.
- Another lisp expression which returns a b/w or color image.

Most of the functions have been adapted to either coerce the arguments into the required types, or perform differently according to the argument types given to them. Arguments to certain functions can optionally be restricted to some subset of the available types. For the most part these functions receive and return images, and can be considered as image processing operations. Expressions are simply evaluated to produce images. Figure 4 shows examples of some simple expressions and their resulting images.

Artificial evolution of these expressions is performed by first generating and displaying a population of simple random expressions in a grid for interactive selection. The expressions of images selected by the user are reproduced with mutations for each new generation such that more and more complex expressions and more perceptually successful images can evolve. Some images evolved with this process are shown in figures 9 to 13.

## 4.2 Mutating Symbolic Expressions

Symbolic expressions must be reproduced with mutations for evolution of them to occur. There are several properties of symbolic expression mutation that are desirable. Expressions should often be only slightly modified, but sometimes significantly adjusted in structure and size. Large random changes in genotype usually result in large jumps in phenotype which are less likely to be improvements, but are necessary for extending the expression to more complex forms.

A recursive mutation scheme is used to mutate expressions. Lisp expressions are traversed as tree structures and each node is in turn subject to possible mutations. Each type of mutation occurs at different frequencies depending on the type of node:

1. Any node can mutate into a new random expression. This allows for large changes, and usually results in a fairly significant alteration of the phenotype.

2. If the node is a scalar value, it can be adjusted by the addition of some random amount.
3. If the node is a vector, it can be adjusted by adding random amounts to each element.
4. If the node is a function, it can mutate into a different function. For example  $(abs X)$  might become  $(cos X)$ . If this mutation occurs, the arguments of the function are also adjusted if necessary to the correct number and types.
5. An expression can become the argument to a new random function. Other arguments are generated at random if necessary. For example  $X$  might become  $(* X .3)$ .
6. An argument to a function can jump out and become the new value for that node. For example  $(* X .3)$  might become  $X$ . This is the inverse of the previous type of mutation.
7. Finally, a node can become a copy of another node from the parent expression. For example  $(+ (abs X) (* Y .6))$  might become  $(+ (abs (* Y .6)) (* Y .6))$ . This causes effects similar to those caused by mating an expression with itself. It allows for sub-expressions to duplicate themselves within the overall expression.

Other types of mutations could certainly be implemented, but these are sufficient for a reasonable balance of slight modifications and potential for changes in complexity.

It is preferable to adjust the mutation frequencies such that a decrease in complexity is slightly more probable than an increase. This prevents the expressions from drifting towards large and slow forms without necessarily improving the results. They should still easily evolve towards larger sizes, but a larger size should be due to selection of improvements instead of random mutations with no effect.

The relative frequencies for each type of mutation above can be adjusted and experimented with. The overall mutation frequency is scaled inversely in proportion to the length of the parent expression. This decreases the probability of mutation at each node when the parent expression is large so that some stability of the phenotypes is maintained.

The evaluation of expressions and display of the resulting images can require significant calculation times as expressions increase in complexity. To keep image evolution at interactive speeds, estimates of compute speeds are calculated for each expression by summing pre-computed runtime averages for each function. Slow expressions are eliminated before ever being displayed to the user. New offspring with random mutations are generated and tested until fast enough expressions result. In this way automatic selection is combined with interactive selection. If necessary, this technique could also be performed to keep memory usage to a minimum.

### **4.3 Mating Symbolic Expressions**

Symbolic expressions can be reproduced with sexual combinations to allow sub-expressions from separately evolved individuals to be mixed into a single individual. Two methods for mating symbolic expressions are described.

The first method requires the two parents to be somewhat similar in structure. The nodes in the expression trees of both parents are simultaneously traversed and copied to make the new expression.

When a difference is encountered between the parents, one of the two versions is copied with equal probability. For example, the following two parents can be mated to generate four different expressions, two of which are equal to the parents, and two of which have some portions from each parent:

```

parent1: (* (abs X) (mod X Y))
parent2: (* (/ Y X) (* X -.7))

child1: (* (abs X) (mod X Y))
child2: (* (abs X) (* X -.7))
child3: (* (/ Y X) (mod X Y))
child4: (* (/ Y X) (* X -.7))

```

This method is often useful for combining similar expressions that each have some desired property. It usually generates offspring without very large variations from the parents. Two expressions with different root nodes will not form any new combinations. This might be compared to the inability of two different species to mate and create viable offspring.

The second method for mating expressions combines the parents in a less constrained way. A node in the expression tree of one parent is chosen at random and replaced by a node chosen at random from the other parent. This *crossing over* technique allows any part of the structure of one parent to be inserted into any part of the other parent and permits parts of even dissimilar expressions to be combined. With this method, the parent expressions above can generate 61 different child expressions - many more than the 4 of the first method.

#### 4.4 Evolving Volume Textures

A third variable,  $Z$ , is added to the list of available arguments to enable functions to be evolved that calculate colors for each point in  $(X,Y,Z)$  space. The *function set* shown in section 4.1 is adjusted for better results: 2D functions that require neighboring pixel values such as convolutions and warps are removed, and 3D solid noise generating functions are added.

These expressions are more difficult to visualize because they encompass all of 3D space. They are evaluated on the surfaces of spheres and planes for fast previewing and selection as shown in figure 5. Evolved volume expressions can then be incorporated into procedural shading functions to texture arbitrary objects. This process allows complex volume textures such as those described in [18] and [19] to be evolved without requiring specific equations to be understood and carefully adjusted by hand. Figure 6 was generated by evolving three volume texture expressions and then evaluating them at the surfaces positions of three objects during the rendering process.

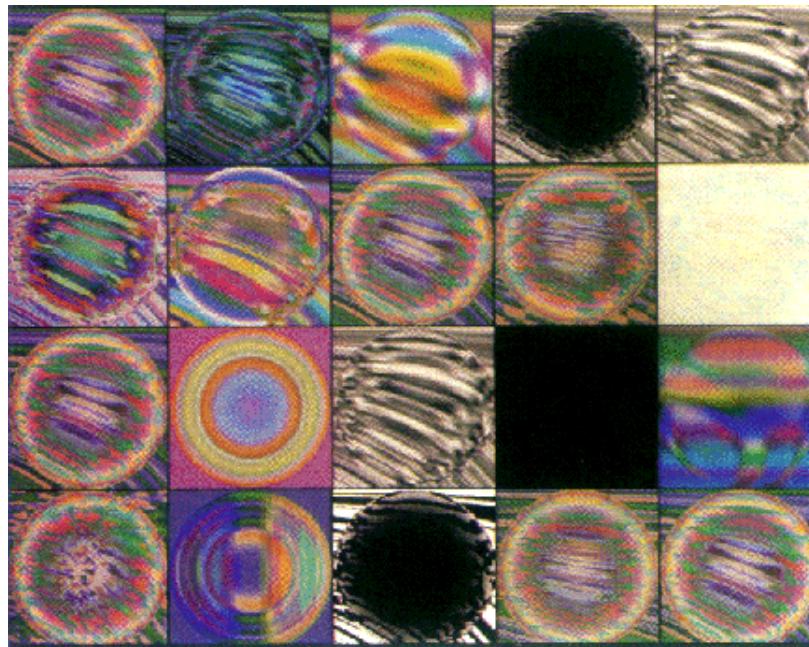


Figure 5: Parent with 19 random mutations.



Figure 6: Marble and wooden tori.

## 4.5 Evolving Animations

Several extensions to the image evolution system described above can be used to evolve moving images. Five methods for incorporating a temporal dimension in symbolic expressions are proposed:

1. Another input variable, *Time*, can be added to the list of available arguments. Expressions can be evolved that are functions of *X*, *Y*, and *Time* such that different images are produced as the value of *Time* is smoothly animated. More computation is required to generate, display and select samples because a sequence of images must be calculated. An alternate method of display involves displaying various slices of the (*X*, *Y*, *Time*) space (although operations requiring neighboring pixel values might not receive the correct information if the values of *Time* vary between them).
2. *Genetic cross dissolves* can be performed between two expressions of similar structure. Interpolation between two expressions is performed by matching the expressions where they are identical and interpolating between the results where they are different. Results of differing expression branches are first calculated and dissolved, and then used by the remaining parts of the expression. If the two expressions have different root nodes, a conventional image dissolve will result. If only parts within their structures are different, interesting motions can occur. This technique utilizes the existing genetic representation of evolved still images to generate in-betweens for a smooth transition from one to another. It is an example of the usefulness of the alternate level of control given by the underlying genetic information. A series of frames from a genetic cross dissolve are shown in figure 7.

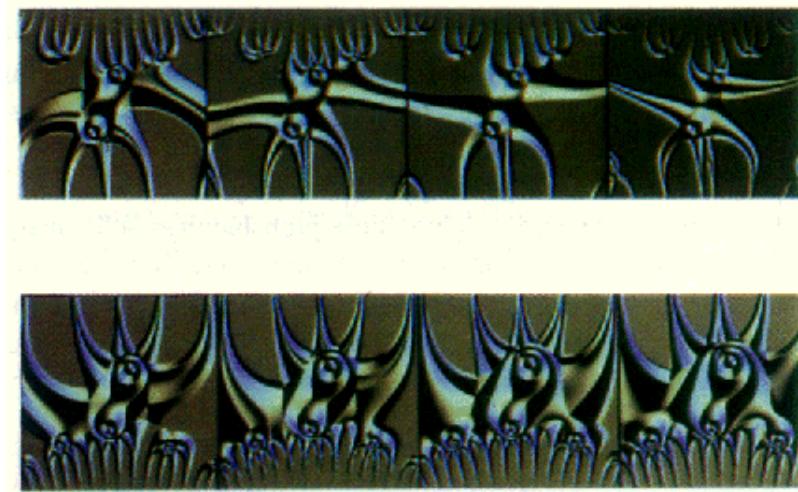


Figure 7: Frames from a “genetic cross dissolve.”

3. An input image can be added to the list of available arguments to make functions of *X*, *Y*, and *Image*. The input image can then be animated and processed by evaluating the expression multiple times for values of *Image* corresponding to frames of another source of animation such as hand drawn or traditional 3D computer graphics. This is effectively a technique for evolving complex image processing and warping functions that compute new images from given input images. Figure 8 was created in this way with an input image of a human face.



Figure 8: Fire of Faces.

4. The images that use the pixel coordinates  $(X, Y)$  to determine the colors at each pixel can be animated by altering the mappings of  $X$  and  $Y$  before the expression is evaluated. Simple zooming and panning can be performed as well as 3D perspective transformations and arbitrary patterns of distortion.
5. Evolved expressions can be adjusted and experimented with by hand. If parameters in expressions are smoothly interpolated to new values, the corresponding image will change in potentially interesting ways. For example, solid noise can be made to change frequency, colors can be dissolved into new shades, and angles can be rotated. This is another example of utilizing the underlying genetic information to manipulate images. A small change in the expression can result in a powerful alteration of the resulting image.

Finally, the techniques above can be used together in various combinations to make an even wider range of possibilities for evolving animations.

## 5 RESULTS

Evolution of 3D plant structures, images, solid textures, and animations have been implemented on the Connection Machine<sup>(R)</sup> system CM-2, a data parallel supercomputer [27][10]. The parallel implementation details will not be discussed in this paper, but each application is reasonably suited for highly parallel representation and computation. Lisp expression mutations and combinations are performed on a *front-end* computer and the Connection Machine system is used to evaluate the expression for all pixels in parallel using *Starlisp* and display the resulting image.

3D Plant structures have been evolved and used in the animated short *Panspermia* [24]. A frame from this sequence is shown in figure 3 which contains a variety of species created using these techniques. An interactive system for quickly growing, displaying, and selecting sample structures allows a wide range

of plant shapes to be efficiently created by artificial evolution. Populations of samples can be displayed for selection in wire frame in a grid format as shown in figure 2, or displayed as separate higher-resolution images which can be interactively flipped through by scrolling with a mouse. Typically between 5 and 20 generations are necessary for acceptable structures to emerge.

Images, volume textures, and various animations have been created using mutating symbolic expressions. These sometimes require more generations to evolve complex expressions that give interesting images - often at least 10 to 40 generations. Again, an interactive tool for quickly displaying grids of sample images to be selected amongst makes the evolution process reasonably efficient. [See figure 5.] The number of possible symbolic expressions of acceptable length is extremely large, and a wide variety of textures and patterns can occur. Completely unexpected kinds of images have emerged. Figure 9 was created from the following evolved expression:

```
(round (log (+ y (color-grad (round (+ (abs (round  
(log (+ y (color-grad (round (+ y (log (invert y) 15.5))  
x) 3.1 1.86 #(0.95 0.7 0.59) 1.35)) 0.19) x)) (log (invert  
y) 15.5)) x) 3.1 1.9 #(0.95 0.7 0.35) 1.35)) 0.19) x)
```

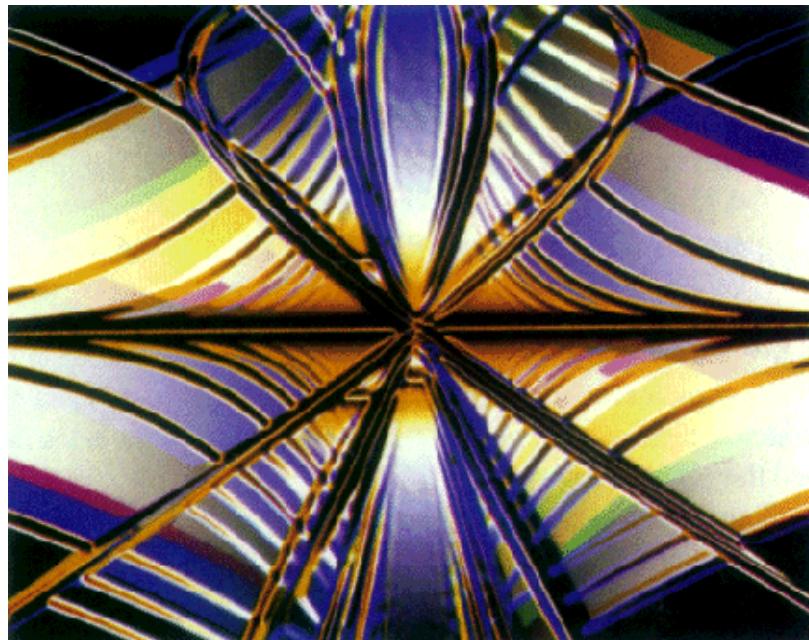


Figure 9

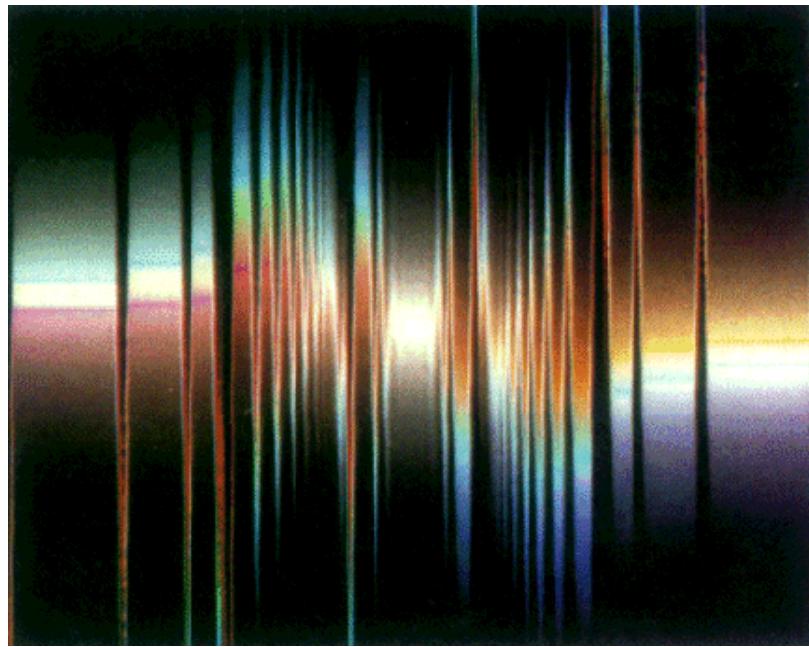


Figure 10



Figure 11

Figure 13 was created from this expression:

```
(sin (+ (- (grad-direction (blur (if (hsv-to-rgb (warped-
color-noise #(0.57 0.73 0.92) (/ 1.85 (warped-color-
noise x y 0.02 3.08)) 0.11 2.4)) #(0.54 0.73 0.59) #(1.06
0.82 0.06)) 3.1) 1.46 5.9) (hsv-to-rgb (warped-color-
```

```
noise y (/ 4.5 (warped-color-noise y (/ x y) 2.4 2.4))  
0.02 2.4))) x))
```

Note that expressions only five or six lines long can generate images of fair complexity. Equations such as these can be evolved from scratch in timescales of only several minutes - probably much faster than they could be designed.

Figures 10, 11, and 12 were also created from expressions of similar lengths. Fortunately, analysis of expressions is not required when using these methods to create them. Users usually stop attempting to understand why each expression generates each image. However, for those interested, expressions for other figures are listed in the appendix.

Two different approaches of user selection behavior are possible. The user can have a goal in mind and select samples that are closer to that goal until it is hopefully reached. Alternatively, the user can follow the more interesting samples as they occur without attempting to reach any specific goal.

The results of these various types of evolved expressions can be saved in the very concise form of the final genotypic expression itself. This facilitates keeping large libraries of evolved forms which can then be used to contribute to further evolutions by mating them with other forms or further evolving them in new directions.

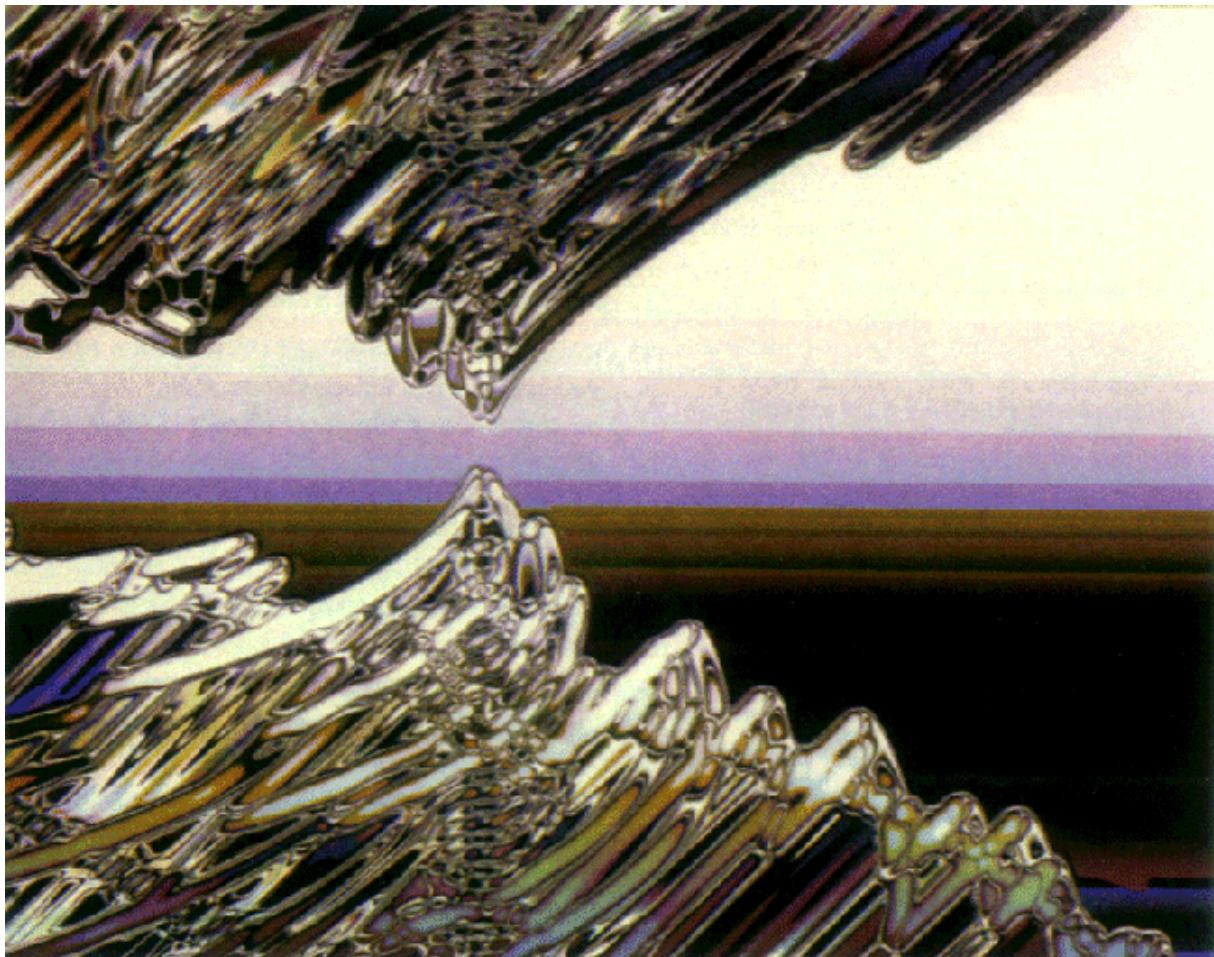


Figure 12



Figure 13

## 6 FUTURE WORK

Artificial evolution has many other possible applications for computer graphics and animation. Procedures that use various other forms of solid noise could be explored, such as those that create objects, create density functions, or warp objects [15][20]. Procedures could be evolved that generate motion from a set of rules (possibly cellular automata, or particle systems), or that control distributions and characteristics of 2D objects such as lines, solid shapes, or brush strokes. Algorithms that use procedural construction rules to create 3D objects from polygons, or functions that generate, manipulate, and combine geometric primitives could also be explored.

These techniques might also make valuable tools in domains beyond computer simulations. New possibilities for shapes and textures could be explored for use in product design or the fashion industry.

Several variations on the methods for artificial evolution described above might make interesting experiments. Mutation frequencies could be included in the genotype itself so that they also can be

mutated. This might allow for the evolution of evolvability [4]. Frequencies from the most successful evolutions could be kept as the defaults.

It might be interesting to attempt to automatically evolve a symbolic expression that could generate a simple specific goal image. An image differencing function could be used to calculate a *fitness* based on how close a test image was to the goal, and an expression could be searched for by automatic selection. Then, interactive selection could be used to evolve further images starting with that expression.

Large amounts of information of all the human selection choices of many evolutions could be saved and analyzed. A difficult challenge would be to create a system that could generalize and “understand” what makes an image visually successful, and even generate other images that meet these learned criteria.

Combinations of random variations and non-random variations using learned information might be helpful. If a user picks phenotypes in a certain direction from the parent, mutations for the next generation might have a tendency to continue in that same direction, causing evolution to have “momentum.”

Also, combinations of evolution and the ability to apply specific adjustments to the genotype might allow more user control over evolved results. Automatic “genetic engineering” could permit a user to request an evolved image to be more blue, or a texture more grainy.

## 7 CONCLUSION

Artificial evolution has been demonstrated to be a potentially powerful tool for the creation of procedurally generated structures, textures, and motions. Reproduction with random variations and survival of the visually interesting can lead to useful results. Representations for genotypes which are not limited to fixed spaces and can grow in complexity have shown to be worthwhile.

Evolution is a method for creating and exploring complexity that does not require human understanding of the specific process involved. This process of artificial evolution could be considered as a system for helping the user with creative explorations, or it might be considered as a system which attempts to “learn” about human aesthetics from the user. In either case, it allows the user and computer to interactively work together in a new way to produce results that neither could easily produce alone.

An important limiting factor in the usefulness of artificial evolution is that samples need to be generated quickly enough such that it is advantageous for the user to choose from random samples instead of carefully adjusting new samples by hand. The computer needs to generate and display samples fast enough to keep the user interested while selecting amongst them. As computation becomes more powerful and available, artificial evolution will hopefully become advantageous in more and more domains.

## 8 Acknowledgments

Thanks to Lew Tucker, Jim Salem, Gary Oberbrunner, Matt Fitzgibbon, Dave Sheppard, and Zotje Maes for help and support. Thanks to Peter Schröder for being a helpful and successful user of these tools.

Thanks to Luis Ortiz and Katy Smith for help with document preparation. And thanks to Danny Hillis, Larry Yaeger, and Richard Dawkins for discussions and inspiration.

## APPENDIX

Figure 5, Parent expression:

```
(warped-color-noise (warped-bw-noise (dissolve x 2.53 y) z 0.09 12.0) (invert z) 0.05 -2.06)
```

Figure 6, Marble torus:

```
(dissolve (cos (and 0.25 #(0.43 0.73 0.74))) (log (+ (warped-bw-noise (min z 11.1) (log (rotate-vector (+ (warped-bw-noise (cos x) (dissolve (cos (and 0.25 #(0.43 0.73 0.74))) (log (+ (warped-bw-noise (max (min z 8.26) (/ -0.5 #(0.82 0.39 0.19))) (log (+ (warped-bw-noise (cos x) z -0.04 0.89) #(0.82 0.39 0.19)) #(0.15 0.34 0.50)) -0.04 -3.0) y) #(0.15 0.34 0.50) y) -0.04 -3.0) x) z y) #(0.15 0.34 0.5)) -0.02 -1.79) -0.4) #(-0.09 0.34 0.55)) -0.7)
```

Figure 7, Cross dissolve:

```
(hsv-to-rgb (bump (hsv-to-rgb (ifs 2.29 0.003 (dissolve 1.77 3.67 time) 2.6 0.1 (dissolve 5.2 3.2 time) -31.0 (dissolve 23.9 -7.4 time) (dissolve 1.13 9.5 time) (dissolve 4.8 0.16 time) 20.7 4.05 (dissolve 0.48 0.46 time) (dissolve 2.94 -0.68 time) (dissolve 0.42 0.54 time) (dissolve 0.09 0.54 time))) (atan 2.25 (dissolve 0.1 0.11 time) 0.15) (dissolve 4.09 8.23 time) (dissolve #(0.41 0.36 0.08) #(0.68 0.22 0.31) time) #(0.36 0.31 0.91) (dissolve 6.2 4.3 time) (dissolve 0.16 0.40 time) (dissolve 2.08 0.23 time)))
```

Figure 8, Fire of Faces:

```
(+ (min 10.8 (warp-rel image image (bump image x 9.6 #(0.57 0.02 0.15) #(0.52 0.03 0.38) 3.21 2.49 10.8))) (dissolve #(0.81 0.4 0.16) x (dissolve y #(0.88 0.99 0.66) image)))
```

Figure 10:

```
(rotate-vector (log (+ y (color-grad (round (+ (abs (round (log #(0.01 0.67 0.86) 0.19) x)) (hsv-to-rgb (bump (if x 10.7 y) #(0.94 0.01 0.4) 0.78 #(0.18 0.28 0.58) #(0.4 0.92 0.58) 10.6 0.23 0.91))) x) 3.1 1.93 #(0.95 0.7 0.35) 3.03)) -0.03) x #(0.76 0.08 0.24))
```

Figure 11 is unfortunately “extinct” because it was created before the genome saving utility was complete.

Figure 12:

```
(cos (round (atan (log (invert y) (+ (bump (+ (round x y) y) #(0.46 0.82 0.65) 0.02 #(0.1 0.06 0.1) #(0.99 0.06 0.41) 1.47 8.7 3.7) (color-grad (round (+ y y) (log (invert x) (+ (invert y) (round (+ y x) (bump (warped-ifs (round y y) y 0.08 0.06 7.4 1.65 6.1 0.54 3.1 0.26 0.73 15.8 5.7 8.9 0.49 7.2 15.6 0.98) #(0.46 0.82 0.65) 0.02 #(0.1 0.06 0.1) #(0.99 0.06 0.41) 0.83 8.7 2.6)))) 3.1 6.8 #(0.95 0.7 0.59) 0.57)) #(0.17 0.08 0.75) 0.37) (vector y 0.09 (cos (round y y)))))
```

## BIBLIOGRAPHY

- [1] Aono, M., and Kunii, T. L., “Botanical Tree Image Generation,” *IEEE Computer Graphics and*

- Applications*, Vol.4, No.5, May 1982.
- [2] Darwin, Charles, *The Origin of Species*, New American Library, Mentor paperback, 1859.
  - [3] Dawkins, Richard, *The Blind Watchmaker*, Harlow Logman, 1986.
  - [4] Dawkins, Richard, “The Evolution of Evolvability,” *Artificial Life Proceedings*, 1987, pp.201-220.
  - [5] Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1989, Addison-Wesley Publishing Co.
  - [6] Grenfenstette, J. J., *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, Hillsdale, New Jersey, Lawrence Erlbaum Associates, 1985.
  - [7] Grenfenstette, J. J., *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, 1987, (Hillsdale, New Jersey: Lawrence Erlbaum Associates.)
  - [8] Haase, K., “Automated Discovery,” *Machine Learning: Principles and Techniques*, by Richard Forsyth, Chapman & Hall 1989, pp.127-155.
  - [9] Haggerty, M., “Evolution by Esthetics, an Interview with W. Latham and S. Todd,” *IEEE Computer Graphics*, Vol.11, No.2, March 1991, pp.5-9.
  - [10] Hillis, W. D., “The Connection Machine,” *Scientific American*, Vol. 255, No. 6, June 1987.
  - [11] Holland, J. H., *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI: University of Michigan Press, 1975.
  - [12] Koza, J. R. “Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems,” Stanford University Computer Science Department Technical Report STAN-CS-90-1314, June 1990.
  - [13] Koza, J. R. “Evolution and Co-Evolution of Computer Programs to Control Independently Acting Agents,” *Conference on Simulation of Adaptive Behavior* (SAB-90) Paris, Sept.24-28, 1990.
  - [14] Lenat, D. B. and Brown,J.S. “Why AM and EURISKO appear to work,” *Artificial intelligence*, Vol.23, 1984, pp.269-294.
  - [15] Lewis, J. P., “Algorithms for Solid Noise Synthesis,” *Computer Graphics*, Vol.23, No.3, July 1989, pp.263-270.
  - [16] Oppenheimer, P. “Real time design and animation of fractal plants and trees.” *Computer Graphics*, Vol.20, No.4, 1986, pp.55-64.
  - [17] Oppenheimer, P. “The Artificial Menagerie” *Artificial Life Proceedings*, 1987, pp.251-274.

- [18] Peachy, D., "Solid Texturing of Complex Surfaces," *Computer Graphics* Vol.19, No.3, July 1985, pp.279-286.
  - [19] Perlin, K., "An Image Synthesizer," *Computer Graphics*, Vol.19, No.3, July 1985, pp.287-296.
  - [20] Perlin, K., "Hypertexture," *Computer Graphics*, Vol.23, No.3, July 1989, pp.253-262.
  - [21] Prusinkiewicz, P., Lindenmayer, A., and Hanan, J., "Developmental Models of Herbaceous Plants for Computer Imagery Purposes," *Computer Graphics*, Vol.22 No.4, 1988, pp.141-150.
  - [22] Reffye, P., Edelin, C., Francon, J., Jaeger, M., Puech, C. "Plant Models Faithful to Botanical Structure and Development," *Computer Graphics* Vol.22, No.4, 1988, pp.151-158.
  - [23] Schaffer, J. D., "Proceedings of the Third international Conference on Genetic Algorithms," June 1989, Morgan Kaufmann Publishers, Inc.
  - [24] Sims, K., *Panspermia*, Siggraph Video Review 1990.
  - [25] Smith, A. R., "Plants, Fractals, and Formal Languages," *Computer Graphics*, Vol.18, No.3, July 1984, pp.1-10.
  - [26] Steele, G., *Common Lisp, The Language*, Digital Press, 1984.
  - [27] Thinking Machines Corporation, *Connection Machine Model CM-2 Technical Summary*, technical report, May 1989.
  - [28] Todd, S. J. P., and Latham, W., "Mutator, a Subjective Human Interface for Evolution of Computer Sculptures," IBM United Kingdom Scientific Centre Report 248, 1991.
  - [29] Viennot, X., Eyrolles, G., Janey, N., and Arques, D., "Combinatorial Analysis of Ramified Patterns and Computer Imagery of Trees," *Computer Graphics*, Vol.23, No.3, July 1989, pp.31-40.
-

# Evolving 3D Morphology and Behavior by Competition

Karl Sims

Thinking Machines Corporation  
245 First Street, Cambridge, MA 02142

## Abstract

This paper describes a system for the evolution and co-evolution of virtual creatures that compete in physically simulated three-dimensional worlds. Pairs of individuals enter one-on-one contests in which they contend to gain control of a common resource. The winners receive higher relative fitness scores allowing them to survive and reproduce. Realistic dynamics simulation including gravity, collisions, and friction, restricts the actions to physically plausible behaviors.

The morphology of these creatures and the neural systems for controlling their muscle forces are both genetically determined, and the morphology and behavior can adapt to each other as they evolve simultaneously. The genotypes are structured as directed graphs of nodes and connections, and they can efficiently but flexibly describe instructions for the development of creatures' bodies and control systems with repeating or recursive components. When simulated evolutions are performed with populations of competing creatures, interesting and diverse strategies and counter-strategies emerge.

## 1 Introduction

Interactions between evolving organisms are generally believed to have a strong influence on their resulting complexity and diversity. In natural evolutionary systems the measure of fitness is not constant: the reproducibility of an organism depends on many environmental factors including other evolving organisms, and is continuously in flux. Competition between organisms is thought to play a significant role in preventing static fitness landscapes and sustaining evolutionary change.

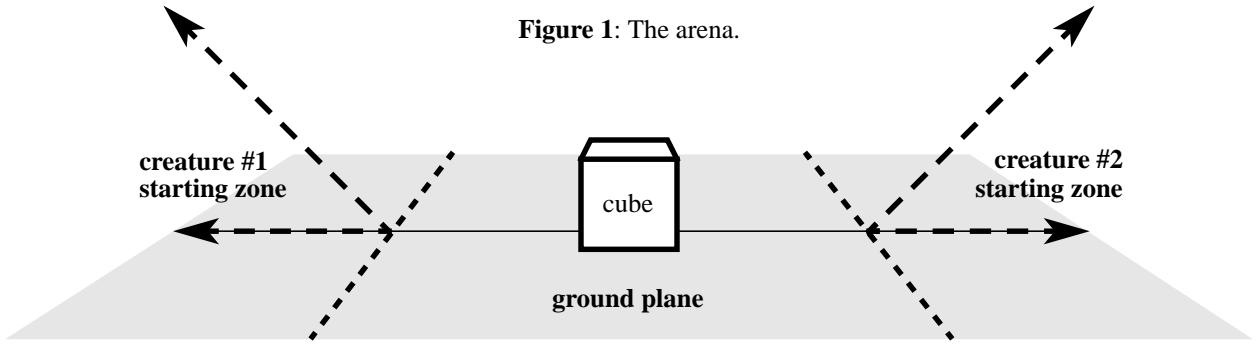
These effects are a distinguishing difference between natural evolution and optimization. Evolution proceeds with no explicit goal, but optimization, including the genetic algorithm, usually aims to search for individuals with the highest possible fitness values where the fitness measure has been predefined, remains constant, and depends only on the individual being tested.

The work presented here takes the former approach. The fitness of an individual is highly dependent on the specific behaviors of other individuals currently in the population. The hope is that virtual creatures with higher complexity and more interesting behavior will evolve than when applying the selection pressures of optimization alone.

Many simulations of co-evolving populations have been performed which involve competing individuals [1,2]. As examples, Lindgren has studied the evolutionary dynamics of competing game strategy rules [14], Hillis has demonstrated that co-evolving parasites can enhance evolutionary optimization [9], and Reynolds evolves vehicles for competition in the game of tag [19]. The work presented here involves similar evolutionary dynamics to help achieve interesting results when phenotypes have three-dimensional bodies and compete in physically simulated worlds.

In several cases, optimization has been used to automatically generate dynamic control systems for given two-dimensional articulated structures: de Garis has evolved weight values for neural networks [6], Ngo and Marks have applied genetic algorithms to generate stimulus-response pairs [16], and van de Panne and Fiume have optimized sensor-actuator networks [17]. Each of these methods has resulted in successful locomotion of two-dimensional stick figures.

The work presented here is related to these projects, but differs in several respects. Previously, control systems were generated for fixed structures that were user-designed, but here entire creatures are evolved: the evolution determines the creature morphologies as well as their control systems. The physical structure of a creature can adapt to its control system, and vice versa, as they evolve together. Also, here the creatures' bodies are three-dimensional and fully physically based. In addition, a developmental process is used to generate the creatures and their control systems, and allows similar components including their local neural circuitry to be defined once and then replicated, instead of requiring each to be separately specified. This approach is related to L-systems, graftal grammars, and object instancing techniques [8,11,13,15,23]. Finally, the previous work on articulated structures relies only on optimization, and competitions between individuals were not considered.



**Figure 1:** The arena.

A different version of the system described here has also been used to generate virtual creatures by optimizing for specific defined behaviors such as swimming, walking, and following [22].

Genotypes used in simulated evolutions and genetic algorithms have traditionally consisted of strings of binary digits [7,10]. Variable length genotypes such as hierarchical Lisp expressions or other computer programs can be useful in expanding the set of possible results beyond a predefined genetic space of fixed dimensions. Genetic languages such as these allow new parameters and new dimensions to be added to the genetic space as an evolution proceeds, and therefore define rather a *hyperspace* of possible results. This approach has been used to genetically program solutions to a variety of problems [3,12], as well as to explore procedurally generated images and dynamical systems [20,21].

In the spirit of unbounded genetic languages, *directed graphs* are presented here as an appropriate basis for a grammar that can be used to describe both the morphology and neural systems of virtual creatures. The level of complexity is variable for both genotype and phenotype. New features and functions can be added to creatures or existing ones removed, as they evolve.

The next section of this paper describes the environment of the simulated contest and how the competitors are scored. Section 3 discusses different simplified competition patterns for approximating competitive environments. Sections 4 and 5 present the genetic language that is used to represent creatures with arbitrary structure and behavior, and section 6 summarizes the physical simulation techniques used. Section 7 discusses the evolutionary simulations including the methods used for mutating and mating directed graph genotypes, and finally sections 8 and 9 provide results, discussion, and suggestions for future work.

## 2 The Contest

Figure 1 shows the arena in which two virtual creatures will compete to gain control of a single cube. The cube is placed in the center of the world, and the creatures start on opposite sides of the cube. The second contestant is initially turned by 180 degrees so the relative position of the cube to the crea-

ture is consistent from contest to contest no matter which starting side it is assigned. Each creature starts on the ground and behind a diagonal plane slanting up and away from the cube. Creatures are wedged into these “starting zones” until they contact both the ground plane and the diagonal plane, so taller creatures must start further back. This helps prevent the inelegant strategy of simply falling over onto the cube. Strategies like this that utilize only potential energy are further discouraged by relaxing a creature’s body before it is placed in the starting zone. The effect of gravity is simulated until the creature reaches a stable minimum state.

At the start of the contest the creatures’ nervous systems are activated, and a physical simulation of the creatures’ bodies, the cube, and the ground plane begins. The winner is the creature that has the most control over the cube after a certain duration of simulated time (8 seconds were given). Instead of just defining a winner and loser, the margin of victory is determined in the form of a relative fitness value, so there is selection pressure not just to win, but to win by the largest possible margin.

The creatures’ final distances to the cube are used to calculate their fitness scores. The shortest distance from any point on the surface of a creature’s parts to the center of the cube is used as its distance value. A creature gets a higher score by being closer to the cube, but also gets a higher score when its opponent is further away. This encourages creatures to reach the cube, but also gives points for keeping the opponent away from it. If  $d_1$  and  $d_2$  are the final shortest distances of each creature to the cube, then the fitnesses for each creature,  $f_1$  and  $f_2$ , are given by:

$$f_1 = 1.0 + \frac{d_2 - d_1}{d_1 + d_2}$$

$$f_2 = 1.0 + \frac{d_1 - d_2}{d_1 + d_2}$$

This formulation puts all fitness values in the limited range of 0.0 to 2.0. If the two distances are equal the contestants receive tie scores of 1.0 each, and in all cases the scores will average 1.0.

Credit is also given for having “control” over the cube, beyond just as measured by the minimum distance to it. If both creatures end up contacting the cube, the winner is the one that surrounds it the most. This is approximated by further decreasing the distance value, as used above, when a creature is touching the cube on the side that opposes its center of mass. Since the initial distances are measured from the center of the cube they can be adjusted in this way and still remain positive.

During the simulated contest, if neither creature shows any movement for a full second, the simulation is stopped and the scores are evaluated early to save unnecessary computation.

### 3 Approximating Competitive Environments

There are many trade-offs to consider when simulating an evolution in which fitness is determined by discrete competitions between individuals. In this work, pairs of individuals compete one-on-one. At every generation of a simulated evolution the individuals in the population are paired up by some pattern and a number of competitions are performed to eventually determine a fitness value for every individual. The simulations of the competitions are by far the dominant computational requirement of the process, so the total number of competitions performed for each generation and the effectiveness of the pattern of competitions are important considerations.

In one extreme, each individual competes with all the others in the population and the average score determines the fitness (figure 2a). However, this requires  $(N^2 - N)/2$  total competitions for a single-species population of  $N$  individuals. For large populations this is often unacceptable, especially if the competition time is significant, as it is in this work.

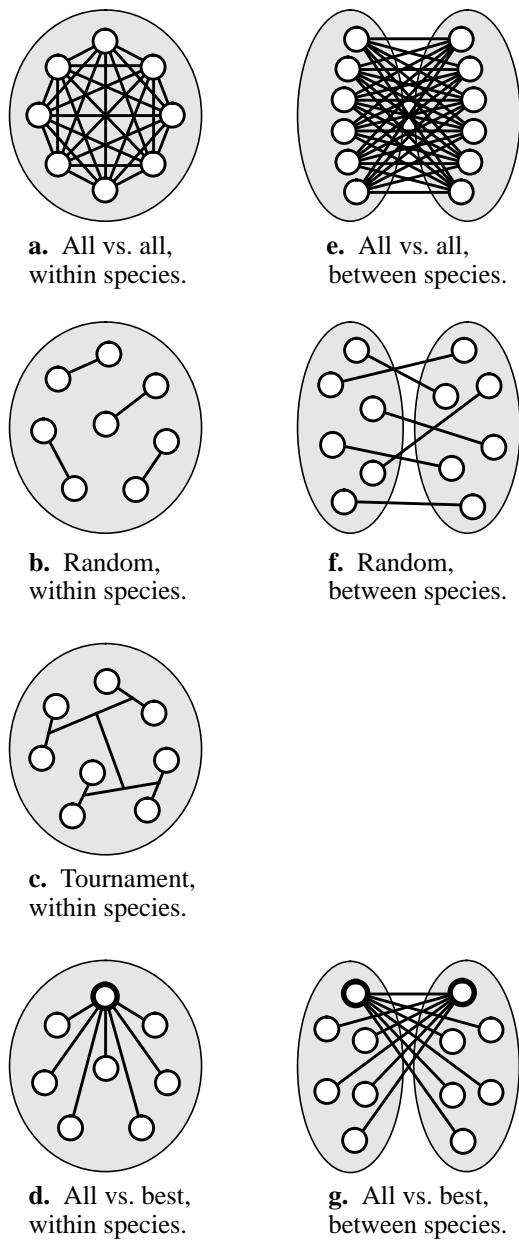
In the other extreme, each individual competes with just a single opponent (figure 2b). This requires only  $N/2$  total competitions, but can cause inconsistency in the fitness values since each fitness is often highly dependent on the specific individual that happens to be assigned as the opponent. If the pairing is done at random, and especially if the mutation rate is high, fitness can be more dependent on the luck of receiving a poor opponent than on an individual’s actual ability.

One compromise between these extremes is for each individual to compete against several opponents chosen at random for each generation. This can somewhat dilute the fitness inconsistency problem, but at the expense of more competition simulations.

A second compromise is a tournament pattern (figure 2c) which can efficiently determine a single overall winner with  $N - 1$  competitions. But this also does not necessarily give all individuals fair scores because of the random initial opponent assignments. Also, this pattern does not easily apply to multi-species evolutions where competitions are not

performed between individuals within the same species.

A third compromise is for each individual to compete once per generation, but all against the same opponent. The individual with the highest fitness from the previous generation is chosen as this one-to-beat (figure 2d). This also requires  $N - 1$  competitions per generation, but effectively gives fair relative fitness values since all are playing against the same opponent which has proven to be competent. Various interesting instabilities can still occur over generations



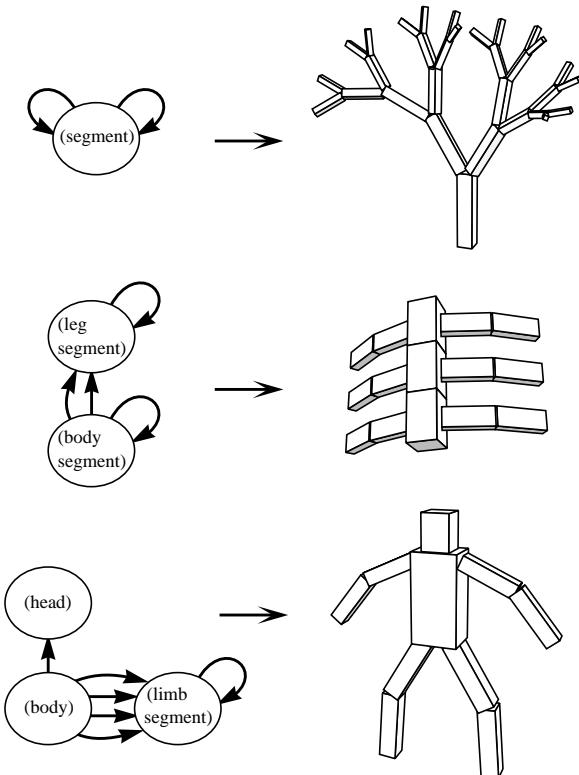
**Figure 2:** Different pair-wise competition patterns for one and two species. The gray areas represent species of interbreeding individuals, and lines indicate competitions performed between individuals.

however, since the strategy of the “best” individual can change suddenly between generations.

The number of species in the population is another element to consider when simulating evolutions involving competition. A species may be described as an interbreeding subset of individuals in the population. In single-species evolutions individuals will compete against their relatives, but in multi-species evolutions individuals can optionally compete only against individuals from other species. Figure 2 shows graphical representations of some of the different competition patterns described above for both one and two species.

The resulting effects of using these different competition patterns is unfortunately difficult to quantify in this work, since by its nature a simple overall measure of success is absent. Evolutions were performed using several of the methods described above with both one and two species, and the results were subjectively judged. The most “interesting” results occurred when the all vs. best competition pattern was used. Both one and two species evolutions produced some intriguing strategies, but the multi-species simulations tended to produce more interesting interactions between the evolving creatures.

**Genotype:** directed graph.      **Phenotype:** hierarchy of 3D parts.



**Figure 3:** Designed examples of genotype graphs and corresponding creature morphologies.

## 4 Creature Morphology

In this work, the phenotype embodiment of a virtual creature is a hierarchy of articulated three-dimensional rigid parts. The genetic representation of this morphology is a directed graph of nodes and connections. Each graph contains the developmental instructions for growing a creature, and provides a way of reusing instructions to make similar or recursive components within the creature. A phenotype hierarchy of parts is made from a graph by starting at a defined *root-node* and synthesizing parts from the node information while tracing through the connections of the graph. The graph can be recurrent. Nodes can connect to themselves or in cycles to form recursive or fractal like structures. They can also connect to the same child multiple times to make duplicate instances of the same appendage.

Each node in the graph contains information describing a rigid part. The *dimensions* determine the physical shape of the part. A *joint-type* determines the constraints on the relative motion between this part and its parent by defining the number of degrees of freedom of the joint and the movement allowed for each degree of freedom. The different joint-types allowed are: *rigid*, *revolute*, *twist*, *universal*, *bend-twist*, *twist-bend*, or *spherical*. *Joint-limits* determine the point beyond which restoring spring forces will be exerted for each degree of freedom. A *recursive-limit* parameter determines how many times this node should generate a phenotype part when in a recursive cycle. A set of local *neurons* is also included in each node, and will be explained further in the next section. Finally, a node contains a set of *connections* to other nodes.

Each connection also contains information. The placement of a child part relative to its parent is decomposed into *position*, *orientation*, *scale*, and *reflection*, so each can be mutated independently. The position of attachment is constrained to be on the surface of the parent part. Reflections cause negative scaling, and allow similar but symmetrical sub-trees to be described. A *terminal-only* flag can cause a connection to be applied only when the recursive limit is reached, and permits tail or hand-like components to occur at the end of chains or repeating units.

Figure 3 shows some simple hand-designed graph topologies and resulting phenotype morphologies. Note that the parameters in the nodes and connections such as *recursive-limit* are not shown for the genotype even though they affect the morphology of the phenotype. The nodes are anthropomorphically labeled as “body,” “leg segment,” etc. but the genetic descriptions actually have no concept of specific categories of functional components.

## 5 Creature Behavior

A virtual “brain” determines the behavior of a creature. The brain is a dynamical system that accepts input sensor values and provides output effector values. The output values are applied as forces or torques at the degrees of freedom of the

body's joints. This cycle of effects is shown in Figure 4.

Sensor, effector, and internal neuron signals are represented here by continuously variable scalars that may be positive or negative. Allowing negative values permits the implementation of single effectors that can both push and pull. Although this may not be biologically realistic, it simplifies the more natural development of muscle pairs.

## 5.1 Sensors

Each sensor is contained within a specific part of the body, and measures either aspects of that part or aspects of the world relative to that part. Three different types of sensors were used for these experiments:

1. *Joint angle sensors* give the current value for each degree of freedom of each joint.

2. *Contact sensors* activate (1.0) if a contact is made, and negatively activate (-1.0) if not. Each contact sensor has a sensitive region within a part's shape and activates when any contacts occur in that area. In this work, contact sensors are made available for each face of each part. No distinction is made between self-contact and environmental contact.

3. *Photosensors* react to a global light source position. Three photosensor signals provide the coordinates of the normalized light source direction relative to the orientation of the part. Shadows are not simulated, so photosensors continue to sense a light source even if it is blocked. Photosensors for two independent colors are made available. The source of one color is located in the desirable cube, and the other is located at the center of mass of the opponent. This effectively allows evolving nervous systems to incorporate specific "cube sensors" and "opponent sensors."

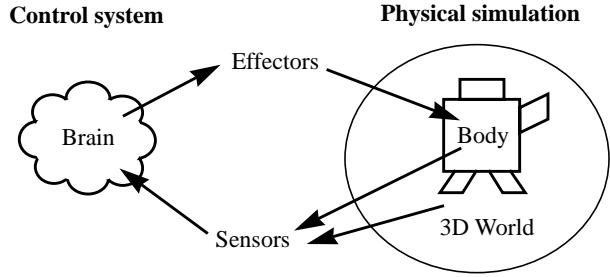
Other types of sensors, such as accelerometers, additional proprioceptors, or even sound or smell detectors could also be implemented, but these basic three are enough to allow some interesting and adaptive behaviors to occur.

## 5.2 Neurons

Internal neural nodes are used to give virtual creatures the possibility of arbitrary behavior. They allow a creature to have an internal state beyond its sensor values, and be affected by its history.

In this work, different neural nodes can perform diverse functions on their inputs to generate their output signals. Because of this, a creature's brain might resemble a dataflow computer program more than a typical artificial neural network. This approach is probably less biologically realistic than just using sum and threshold functions, but it is hoped that it makes the evolution of interesting behaviors more likely. The set of functions that neural nodes can have is: *sum, product, divide, sum-threshold, greater-than, sign-of, min, max, abs, if, interpolate, sin, cos, atan, log, expt, sigmoid, integrate, differentiate, smooth, memory, oscillate-wave, and oscillate-saw*.

Some functions compute an output directly from their



**Figure 4:** Cycle of effects between brain, body and world.

inputs, while others such as the oscillators retain some state and can give time varying outputs even when their inputs are constant. The number of inputs to a neuron depends on its function, and here is at most three. Each input contains a connection to another neuron or a sensor from which to receive a value. Alternatively, an input can simply receive a constant value. The input values are first scaled by weights before being operated on. The genetic parameters for each neural node include these weights as well as the function type and the connection information.

For each simulated time interval, every neuron computes its output value from its inputs. In this work, two brain time steps are performed for each dynamic simulation time step so signals can propagate through multiple neurons with less delay.

## 5.3 Effectors

Each effector simply contains a connection from a neuron or a sensor from which to receive a value. This input value is scaled by a constant weight, and then exerted as a joint force which affects the dynamic simulation and the resulting behavior of the creature. Different types of effectors, such as sound or scent emitters, might also be interesting, but only effectors that exert simulated muscle forces are used here.

Each effector controls a degree of freedom of a joint. The effectors for a given joint connecting two parts, are contained in the part further out in the hierarchy, so that each non-root part operates only a single joint connecting it to its parent. The angle sensors for that joint are also contained in this part.

Each effector is given a *maximum-strength* proportional to the maximum cross sectional area of the two parts it joins. Effector forces are scaled by these strengths and not permitted to exceed them. This is similar to the strength limits of natural muscles. As in nature, mass scales with volume but strength scales with area, so behavior does not always scale uniformly.

## 5.4 Combining Morphology and Control

The genotype descriptions of virtual brains and the actual phenotype brains are both directed graphs of nodes and connections. The nodes contain the sensors, neurons, and effec-

tors, and the connections define the flow of signals between these nodes. These graphs can also be recurrent, and as a result the final control system can have feedback loops and cycles.

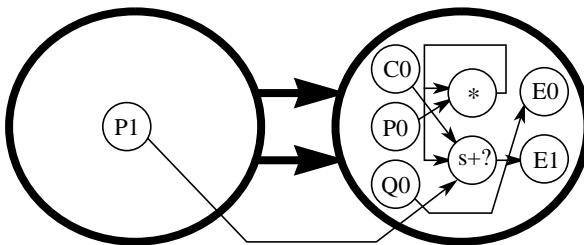
However, most of these neural elements exist within a specific part of the creature. Thus the genotype for the nervous system is a nested graph: the morphological nodes each contain graphs of the neural nodes and connections. Figure 5 shows an example of an evolved nested graph which describes a simple three-part creature as shown in figure 6.

When a creature is synthesized from its genetic description, the neural components described within each part are generated along with the morphological structure. This causes blocks of neural control circuitry to be replicated along with each instanced part, so each duplicated segment or appendage of a creature can have a similar but independent local control system.

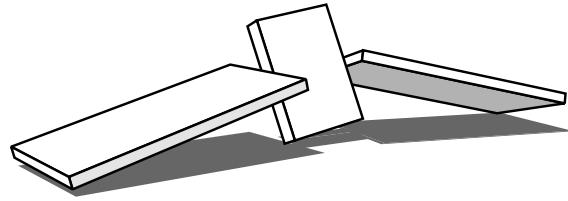
These local control systems can be connected to enable the possibility of coordinated control. Connections are allowed between adjacent parts in the hierarchy. The neurons and effectors within a part can receive signals from sensors or neurons in their parent part or in their child parts.

Creatures are also allowed a set of neurons that are not associated with a specific part, and are copied only once into the phenotype. This gives the opportunity for the development of global synchronization or centralized control. These neurons can receive signals from each other or from sensors or neurons in specific instances of any of the creature’s parts, and the neurons and effectors within the parts can optionally receive signals from these unassociated-neuron outputs.

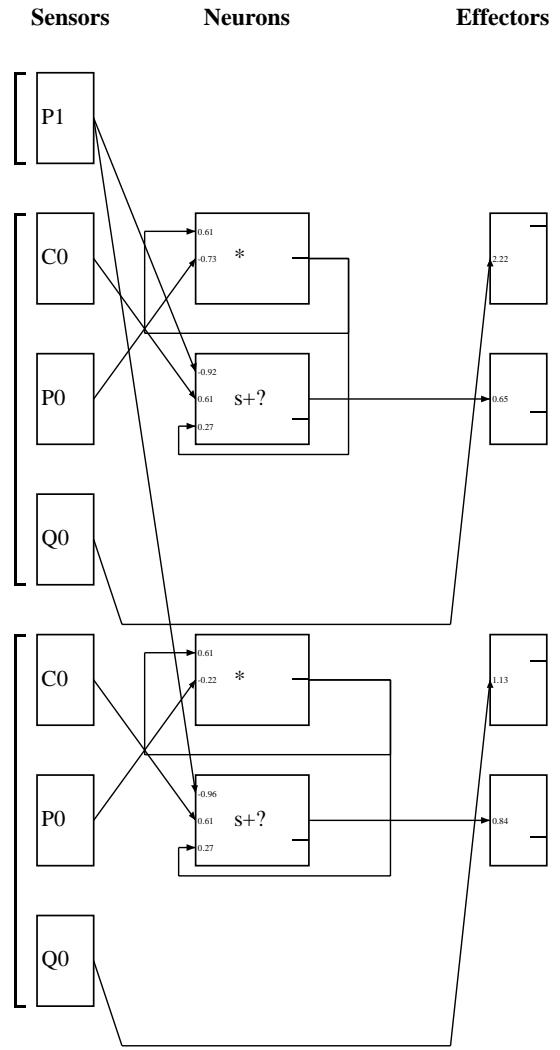
In this way the genetic language for morphology and control is merged. A local control system is described for each type of part, and these are copied and connected into the hierarchy of the creature’s body to make a complete distributed nervous system. Figure 6a shows the creature morphology resulting from the genotype in figure 5. Again, parameters describing shapes and weight values are not shown for the genotype even though they affect the pheno-



**Figure 5:** Example evolved nested graph genotype. The outer graph in bold describes a creature’s morphology. The inner graph describes its neural circuitry. C0, P0, Q0, and E0, E1 are sensor and effector nodes, and those labeled “\*” and “s+?” are neural nodes that perform product and sum-threshold functions.



**Figure 6a:** The phenotype morphology generated from the evolved genotype shown in figure 5.



**Figure 6b:** The phenotype “brain” generated from the evolved genotype shown in figure 5. The effector outputs of this control system cause the morphology above to roll forward in tumbling motions.

type. Figure 6b shows the corresponding brain of this creature. The brackets on the left side of figure 6b group the neural components of each part. Two groups have similar neural systems because they were synthesized from the same genetic description. This creature can roll over the ground by making cyclic tumbling motions with its two arm-like appendages. Note that it can be difficult to analyze exactly how a control system such as this works, and some components may not actually be used at all. Fortunately, a primary benefit of using artificial evolution is that understanding these representations is not necessary.

## 6 Physical Simulation

Dynamics simulation is used to calculate the movement of creatures resulting from their interaction with a virtual three-dimensional world. There are several components of the physical simulation used in this work: articulated body dynamics, numerical integration, collision detection, and collision response with friction. These are only briefly summarized here, since physical simulation is not the emphasis of this paper.

Featherstone's recursive  $O(N)$  articulated body method is used to calculate the accelerations from the velocities and external forces of each hierarchy of connected rigid parts [5]. Integration determines the resulting motions from these accelerations and is performed by a Runge-Kutta-Fehlberg method which is a fourth order Runge-Kutta with an additional evaluation to estimate the error and adapt the step size. Typically between 1 and 5 integration time steps are performed for each frame of 1/30 second.

The shapes of parts are represented here by simple rectangular solids. Bounding box hierarchies are used to reduce the number of collision tests between parts from  $O(N^2)$ . Pairs whose world-space bounding boxes intersect are tested for penetrations, and collisions with a ground plane are also tested. If necessary, the previous time-step is reduced to keep any new penetration depths below a certain tolerance. Connected parts are permitted to interpenetrate but not rotate completely through each other. This is achieved by using adjusted shapes when testing for collisions between connected parts. The shape of the smaller part is clipped halfway back from its point of attachment so it can swing freely until its remote end makes contact.

Collision response is accomplished by a hybrid model using both impulses and penalty spring forces. At high velocities, instantaneous impulse forces are used, and at low velocities springs are used, to simulate collisions and contacts with arbitrary elasticity and friction parameters.

It is important that the physical simulation be reasonably accurate when optimizing for creatures that can move within it. Any bugs that allow energy leaks from non-conservation, or even round-off errors, will inevitably be discovered and exploited by the evolving creatures. Although this can be a lazy and often amusing approach for debugging a

physical modeling system, it is not necessarily the most practical.

## 7 Creature Evolution

An evolution of virtual creatures is begun by first creating an initial population of genotypes. Seed genotypes are synthesized "from scratch" by random generation of sets of nodes and connections. Alternatively, an existing genotype from a previous evolution can be used to seed an initial population.

Before creatures are paired off for competitions and fitness evaluation, some simple viability checks are performed, and inappropriate creatures are removed from the population by giving them zero fitness values. Those that have more than a specified number of parts are removed. A subset of genotypes will generate creatures whose parts initially interpenetrate. A short simulation with collision detection and response attempts to repel any intersecting parts, but those creatures with persistent interpenetrations are also discarded.

A *survival-ratio* determines the percentage of the population that will survive each generation. In this work, population sizes were typically 300, and the survival-ratio was 1/5. If the initially generated population has fewer individuals with positive fitness than the number that should survive, another round of seed genotypes is generated to replace those with zero fitness.

For each generation, creatures are grown from their genotypes, and their fitness values are measured by simulating one or more competitions with other individuals as described. The individuals whose fitnesses fall within the survival percentile are then reproduced, and their offspring fill the slots of those individuals that did not survive. The number of offspring that each surviving individual generates is proportional to its fitness. The survivors are kept in the population for the next generation, and the total size of the population is maintained. In multi-species evolutions, each sub-population is independently treated in this way so the number of individuals in each species remains constant and species do not die out.

Offspring are generated from the surviving creatures by copying and combining their directed graph genotypes. When these graphs are reproduced they are subjected to probabilistic variation or mutation, so the corresponding phenotypes are similar to their parents but have been altered or adjusted in random ways.

### 7.1 Mutating Directed Graphs

A directed graph is mutated by the following sequence of steps:

1. The internal parameters of each node are subjected to possible alterations. A mutation frequency for each parameter type determines the probability that a mutation will be applied to it at all. Boolean values are mutated by simply flipping their state. Scalar values are mutated by adding several random numbers to them for a Gaussian-like distribution

so small adjustments are more likely than drastic ones. The scale of an adjustment is relative to the original value, so large quantities can be varied more easily and small ones can be carefully tuned. A scalar can also be negated. After a mutation occurs, values are clamped to their legal bounds. Some parameters that only have a limited number of legal values are mutated by simply picking a new value at random from the set of possibilities.

2. A new random node is added to the graph. A new node normally has no effect on the phenotype unless a connection also mutates a pointer to it. Therefore a new node is always initially added, but then garbage collected later (in step 5) if it does not become connected. This type of mutation allows the complexity of the graph to grow as an evolution proceeds.

3. The parameters of each connection are subjected to possible mutations in the same way the node parameters were in step 1. With some frequency the connection pointer is moved to point to a different node which is chosen at random.

4. New random connections may be added and existing ones may be removed. In the case of the neural graphs these operations are not performed because the number of inputs for each element is fixed, but the morphological graphs can have a variable number of connections per node. Each existing node is subject to having a new random connection added to it, and each existing connection is subject to possible removal.

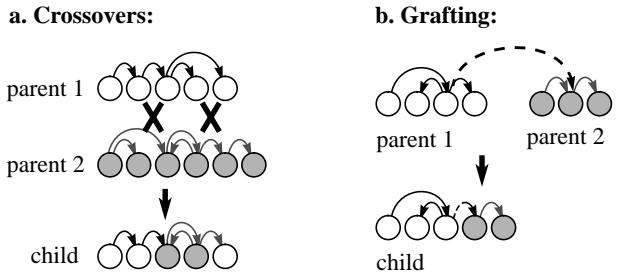
5. Unconnected elements are garbage collected. Connectedness is propagated outwards through the connections of the graph, starting from the root node of the morphology, and from the effector nodes of the neural graphs. Although leaving the disconnected nodes for possible reconnection might be advantageous, and is probably biologically analogous, at least the unconnected newly added ones are removed to prevent unnecessary growth in graph size.

Since mutations are performed on a per element basis, genotypes with only a few elements might not receive any mutations, where genotypes with many elements would receive enough mutations that they would rarely resemble their parents. This is compensated for by scaling the mutation frequencies by an amount inversely proportional to the size of the current graph being mutated, such that on the average at least one mutation occurs in the entire graph.

Mutation of nested directed graphs, as are used here to represent creatures, is performed by first mutating the outer graph and then mutating the inner layer of graphs. The inner graphs are mutated last because legal values for some of their parameters (inter-node neural input sources) can depend on the topology of the outer graph.

## 7.2 Mating Directed Graphs

Sexual reproduction allows components from more than one parent to be combined into new offspring. This permits features to evolve independently and later be merged into a single



**Figure 7:** Two methods for mating directed graphs.

individual. Two different methods for mating directed graphs are used in this work.

The first is a *crossover* operation (figure 7a). The nodes of two parents are each aligned in a row as they are stored, and the nodes of the first parent are copied to make the child, but one or more crossover points determine when the copying source should switch to the other parent. The connections of a node are copied with it and simply point to the same relative node locations as before. If the copied connections now point out of bounds because of varying node numbers they are randomly reassigned.

A second mating method *grafts* two genotypes together by connecting a node of one parent to a node of another (figure 7b). The first parent is copied, and one of its connections is chosen at random and adjusted to point to a random node in the second parent. Newly unconnected nodes of the first parent are removed and the newly connected node of the second parent and any of its descendants are appended to the new graph.

A new directed graph can be produced by either of these two mating methods, or asexually by using only mutations. Offspring from matings are sometimes subjected to mutations afterwards, but with reduced mutation frequencies. In this work a reproduction method is chosen at random for each child to be produced by the surviving individuals using the ratios: 40% asexual, 30% crossovers, and 30% grafting. A second parent is chosen from the survivors if necessary, and a new genotype is produced from the parent or parents.

After a new generation of genotypes is created, a phenotype creature is generated from each, and again their fitness values are evaluated. As this cycle of variation and selection continues, the population is directed towards creatures with higher fitness.

## 7.3 Parallel Implementation

This process has been implemented to run in parallel on a Connection Machine® CM-5 in a master/slave message passing model. A single processing node contains the population and performs all the selection and reproduction operations. It farms out pairs of genotypes to the other nodes to be fitness tested, and gathers back the fitness values after they have been determined. The fitness tests each include a dynamics

simulation for the competition and although many can execute in nearly real-time, they are still the dominant computational requirement of the system. Performing a fitness test per processor is a simple but effective way to parallelize this process, and the overall performance scales quite linearly with the number of processors, as long as the population size is somewhat larger than the number of processors.

Each fitness test takes a different amount of time to compute depending on the complexity of the creatures and how they attempt to move. To prevent idle processors from just waiting for others to finish, the slowest few simulations at the end of a generation are suspended and those individuals are removed from the population by giving them zero fitness. With this approach, an evolution with population size 300, run for 100 generations, might take about four hours to complete on a 32 processor CM-5.

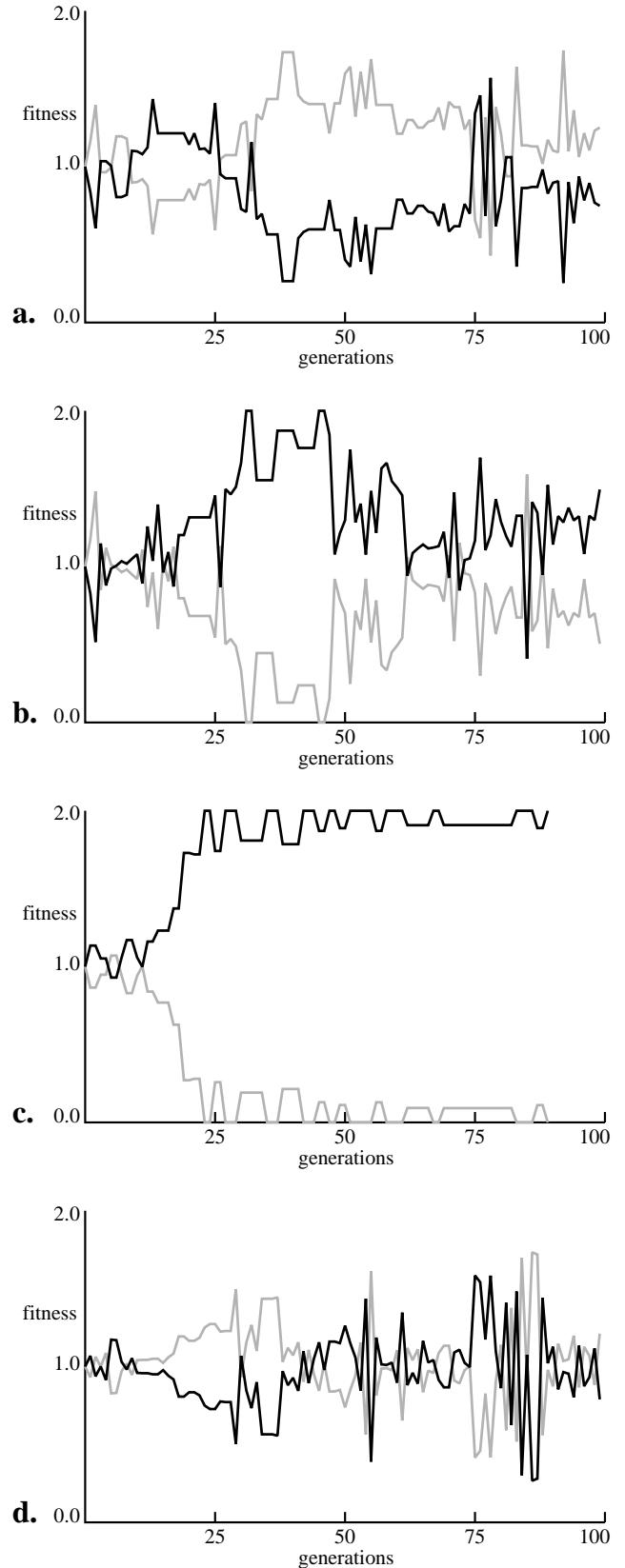
## 8 Results and Discussion

Many independent evolutions were performed using the “all vs. best” competition pattern as described in section 3. Some single-species evolutions were performed in which all individuals both compete and breed with each other, but most included two species where individuals only compete with members of the opponent species.

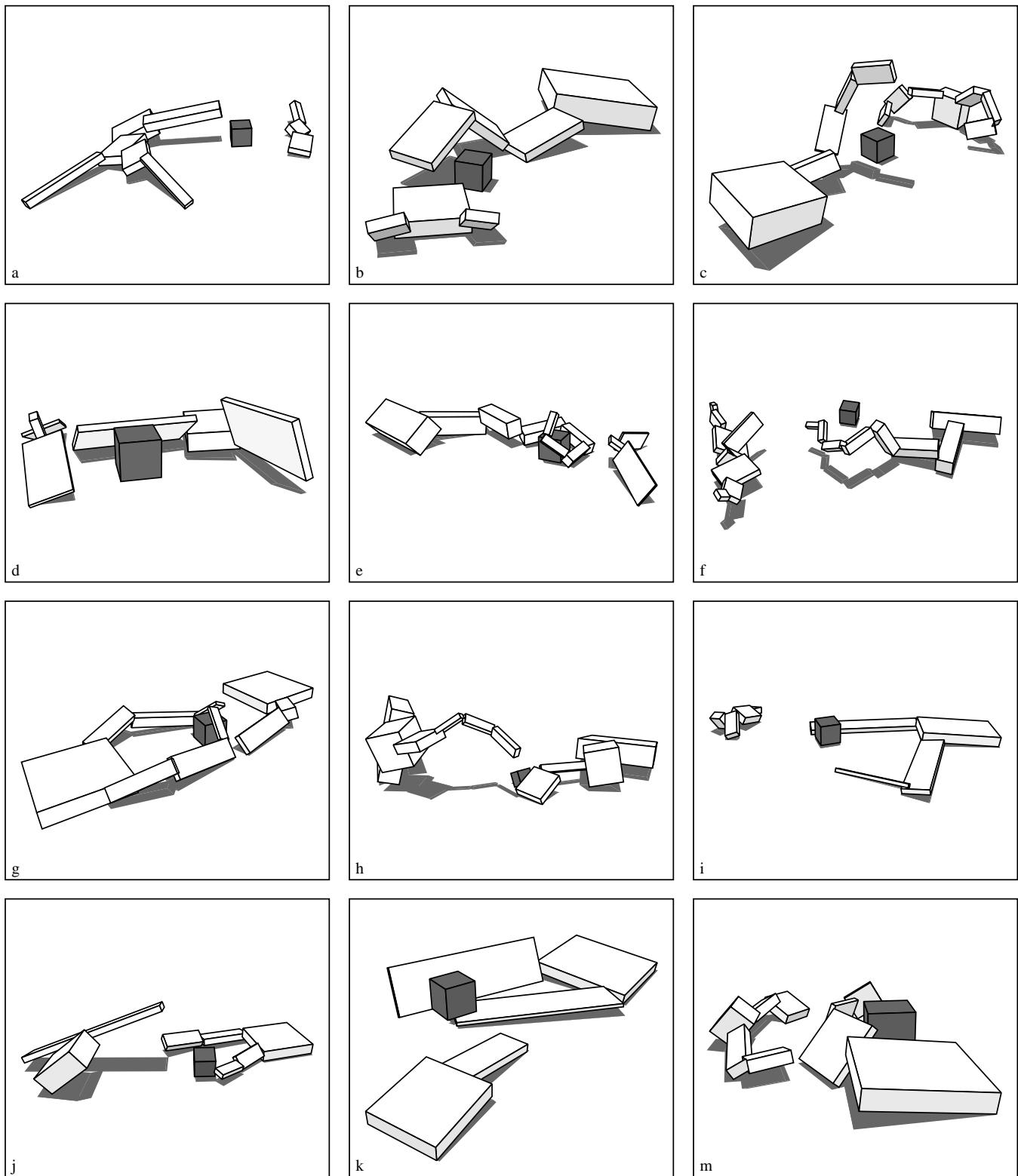
Some examples of resulting two-species evolutionary dynamics are shown in Figure 8. The relative fitness of the best individuals of each species are plotted over 100 generations. The rate of evolutionary progress varied widely in different runs. Some species took many generations before they could even reach the cube at all, while others discovered a fairly successful strategy in the first 10 or 20 generations. Figure 8c shows an example where one species was successful fairly quickly and the other species never evolved an effective strategy to challenge it. The other three graphs in figure 8 show evolutions where more interactions occurred between the evolving species.

A variety of methods for reaching the cube were discovered. Some extended arms out onto the cube, and some reached out while falling forward to land on top of it. Others could crawl inch-worm style or roll towards the cube, and a few even developed leg-like appendages that they used to walk towards it.

The most interesting results often occurred when both species discovered methods for reaching the cube and then further evolved strategies to counter the opponent’s behavior. Some creatures pushed their opponent away from the cube, some moved the cube away from its initial location and then followed it, and others simply covered up the cube to block the opponent’s access. Some counter-strategies took advantage of a specific weakness in the original strategy and could be easily foiled in a few generations by a minor adaptation to the original strategy. Others permanently defeated the original strategy and required the first species to evolve another level of counter-counter-strategy to regain the lead.



**Figure 8:** Relative fitness between two co-evolving and competing species, from four independent simulations.



**Figure 9:** Evolved competing creatures.

In some evolutions the winners alternated between species many times with new strategies and counter-strategies. In other runs one species kept a consistent lead with the other species only providing temporary challenges.

After the results from many simulations were observed, the best were collected and then played against each other in additional competitions. The different strategies were compared, and the behavior and adaptability of creatures were observed as they faced new types of opponents that were not encountered during their evolutions. A few evolutions were also performed starting with an existing creature as a seed genotype for each species so they could further evolve to compete against a new type of opponent.

Figure 9 shows some examples of evolved competing creatures and demonstrates the diversity of the different strategies that emerged. Some of the behaviors and interactions of these specific creatures are described briefly here. The larger creature in figure 9b nudges the cube aside and then pins down his smaller opponent. The crab-like creature in 9c can successfully walk forward, but then continues blindly past the cube and over the opponent. Figure 9d shows a creature that has just pushed its opponent away from the cube, and the arm-like creature in 9e also jabs at its opponent before curling around the cube.

Most creatures perform similar behavior independently of the opponent's actions, but a few are adaptive in that they can reach towards the cube wherever it moves. For example the arm-like creature in figure 9f pushes the cube aside and then uses photosensors to adaptively follow it. If its opponent moves the cube in a different direction it will successfully grope towards the new location.

The two-armed creature in figure 9g blocks access to the cube by covering it up. Several other two-armed creatures in 9i, 9j, and 9k use the strategy of batting the cube to the side with one arm and catching it with the other arm. This seemed to be the most successful strategy of the creatures in this group, and the one in 9k was actually the overall winner because it could whisk the cube aside very quickly. However, it was a near tie between this and the photosensitive arm in 9f. The larger creature in 9m wins by a large margin against some opponents because it can literally walk away with the cube, but it does not initially reach the cube very quickly and tends to loose against faster opponents.

It is possible that adaptation on an evolutionary scale occurred more easily than the evolution of individuals that were themselves adaptive. Perhaps individuals with adaptive behavior would be significantly more rewarded if evolutions were performed with many species instead of just one or two. To be successful, a single individual would then need to defeat a larger number of different opposing strategies.

## 9 Future Work

Several variations on this system could be worth further experimentation. Other types of contests could be defined in

which creatures compete in different environments and different rules determine the winners. Creatures might also be rewarded for cooperative behavior somehow as well as competitive, and teams of interacting creatures could be simulated.

Evolutions containing larger numbers of species should certainly be performed, with the hope of increasing the chances for emergence of more adaptive individuals as hypothesized above.

An additional extension to this work would be to simulate a more complex but more realistic environment in which many creatures simultaneously compete and/or cooperate with each another, instead of pairing off in one-on-one contests. Speciation, mating patterns, competing patterns, and even offspring production could all be determined by one long ecological simulation. Experiments like this have been performed with simpler organisms and have produced interesting results including specialization and various social interactions [18,24].

Perhaps the techniques presented here should be considered as an approach toward creating artificial intelligence. When a genetic language allows virtual entities to evolve with increasing complexity, it is common for the resulting system to be difficult to understand in detail. In many cases it would also be difficult to design a similar system using traditional methods. Techniques such as these have the potential of surpassing those limits that are often imposed when human understanding and design is required. The examples presented here suggest that it might be easier to evolve virtual entities exhibiting intelligent behavior than it would be for humans to design and build them.

## 10 Conclusion

In summary, a system has been described that can automatically generate autonomous three-dimensional virtual creatures that exhibit diverse competitive strategies in physically simulated worlds. A genetic language that uses directed graphs to describe both morphology and behavior defines an unlimited hyperspace of possible results, and a variety of interesting virtual creatures have been shown to emerge when this hyperspace is explored by populations of evolving and competing individuals.

## Acknowledgments

Thanks to Gary Oberbrunner and Matt Fitzgibbon for Connection Machine and software support. Thanks to Thinking Machines Corporation and Lew Tucker for supporting this research. Thanks to Bruce Blumberg and Peter Schröder for dynamic simulation help and suggestions. And special thanks to Pattie Maes.

## References

1. Angeline, P.J., and Pollack, J.B., "Competitive Environments Evolve Better Solutions for Complex Tasks," in *Proceedings of the 5th International Conference on Genetic Algorithms*, ed. by S. Forrest, Morgan Kaufmann 1993, pp.264-270.
2. Axelrod, R., "Evolution of Strategies in the Iterated Prisoner's Dilemma", in *Genetic Algorithms and Simulated Annealing*, ed. by L. Davis, Morgan Kaufmann, 1989.
3. Cramer, N.L., "A Representation for the Adaptive Generation of Simple Sequential Programs," *Proceedings of the First International Conference on Genetic Algorithms*, ed. by J. Grefenstette, 1985, pp.183-187.
4. Dawkins, R., *The Blind Watchmaker*, Harlow Longman, 1986.
5. Featherstone, R., *Robot Dynamics Algorithms*, Kluwer Academic Publishers, Norwell, MA, 1987.
6. de Garis, H., "Genetic Programming: Building Artificial Nervous Systems Using Genetically Programmed Neural Network Modules," *Proceedings of the 7th International Conference on Machine Learning*, 1990, pp.132-139.
7. Goldberg, D.E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.
8. Hart, J., "The Object Instancing Paradigm for Linear Fractal Modeling," *Graphics Interface*, 1992, pp.224-231.
9. Hillis, W.D., "Co-evolving parasites improve simulated evolution as an optimization procedure," *Artificial Life II*, ed. by Langton, Taylor, Farmer, & Rasmussen, Addison-Wesley, 1991, pp.313-324.
10. Holland, J.H., *Adaptation in Natural and Artificial Systems*, Ann Arbor, University of Michigan Press, 1975.
11. Kitano, H., "Designing neural networks using genetic algorithms with graph generation system," *Complex Systems*, Vol.4, pp.461-476, 1990.
12. Koza, J., *Genetic Programming: on the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.
13. Lindenmayer, A., "Mathematical Models for Cellular Interactions in Development, Parts I and II," *Journal of Theoretical Biology*, Vol.18, 1968, pp.280-315.
14. Lindgren, K., "Evolutionary Phenomena in Simple Dynamics," in *Artificial Life II*, ed. by Langton, Taylor, Farmer, & Rasmussen, Addison-Wesley, 1991, pp.295-312.
15. Mjolsness, E., Sharp, D., and Alpert, B., "Scaling, Machine Learning, and Genetic Neural Nets," *Advances in Applied Mathematics*, Vol.10, 1989, pp.137-163.
16. Ngo, J.T., and Marks, J., "Spacetime Constraints Revisited," *Computer Graphics*, Annual Conference Series, 1993, pp.343-350.
17. van de Panne, M., and Fiume, E., "Sensor-Actuator Networks," *Computer Graphics*, Annual Conference Series, 1993, pp.335-342.
18. Ray, T., "An Approach to the Synthesis of Life," *Artificial Life II*, ed. by Langton, Taylor, Farmer, & Rasmussen, Addison-Wesley, 1991, pp.371-408.
19. Reynolds, C., "Competition, Coevolution and the Game of Tag," to be published in: *Artificial Life IV Proceedings*, ed. by R. Brooks & P. Maes, MIT Press, 1994.
20. Sims, K., "Artificial Evolution for Computer Graphics," *Computer Graphics*, Vol.25, No.4, July 1991, pp.319-328.
21. Sims, K., "Interactive Evolution of Dynamical Systems," *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, ed. by Varela, Francisco, & Bourgine, MIT Press, 1992, pp.171-178.
22. Sims, K., "Evolving Virtual Creatures," *Computer Graphics*, Annual Conference Series, July 1994, pp.15-22.
23. Smith, A.R., "Plants, Fractals, and Formal Languages," *Computer Graphics*, Vol.18, No.3, July 1984, pp.1-10.
24. Yaeger, L., "Computational Genetics, Physiology, Metabolism, Neural Systems, Learning, Vision, and Behavior or PolyWorld: Life in a New Context," *Artificial Life III*, ed. by C. Langton, Santa Fe Institute Studies in the Sciences of Complexity, Proceedings Vol. XVII, Addison-Wesley, 1994, pp.263-298.



# Building Behaviors for Animation and Interactive Multimedia

*Craig W. Reynolds*

*DreamWorks SKG  
Universal City, CA*

## Behavioral Models



- Autonomous agents for simulated worlds
- Intersection of several fields
  - ethology
  - artificial life
  - autonomous robotics
  - dramatic characters
- Adjunct to physically-based behavior
  - dynamics versus volition
  - bouncing ball versus pursuing puppy

## Reactive Behavior



- Behavior driven by reaction to environment
  - both passive scenery and active characters
- Simplifies complex animation
  - many characters can be animated by a single behavior
- Allows user interaction
  - improvisational style permits unscripted action

## Applications of Behavioral Models



- Behavioral animation
  - coordinated group motion
  - extras / background action
- Interactive multimedia
  - games / virtual reality environments
  - opponents and allies
- Autonomous robotics
  - search / exploration / mapping
- Theoretical biology
  - testing theories of emergent natural behavior

## Creating Behaviors



- By design
  - programming
  - authoring
- Through self-organization
  - evolution
  - other forms of “machine learning”  
neural nets / decision trees /  
classifier systems / simulated annealing

## *Ad hoc Behavioral Hierarchy*



- Action selection, goals and strategies  
*(see Blumberg and Galyean, SIGGRAPH 95)*
- Path selection / steering / global motion
- Pose selection / locomotion / local motion (animation)

## Combining Simultaneous Behaviors



- Behaviors with overlapping effect
  - (behaviors with disjoint effect are independent)
- Combination
  - discrete selection
  - behavioral blending
- Low priority behavior should not be:
  - completely locked out
  - allowed to contradict (perhaps cancel out) a higher priority behavior

## Behavioral Blending



- Summation / averaging
- Prioritized sequential selection
  - first active
  - stochastic (dither)
- Prioritized acceleration allocation

## Behavioral Examples



- Hand programmed
  - steering behavior library
  - boids
  - hockey players
- Evolution
  - corridor following
  - tag players

## Behavioral Animation



- Background action
- Autonomous characters
  - behavioral model
  - graphical model
- Improvised action

## **Behavioral Animation: Group Motion**



- Individual
  - simple local behavior
  - interaction with:
    - nearby individuals
    - local environment
- Group:
  - complex global behavior

## **Behavioral Animation: Examples of Group Motion**



- People
  - crowds, mobs, passersby
- Animal
  - flocks, schools, herds
- Vehicle
  - traffic

## Applications of Behavioral Animations



- 1987: *Stanley and Stella in: Breaking the Ice*, (short)  
Director: Larry Malone, Producer: Symbolics, Inc.
- 1988: *Behave*, (short)  
Produced and directed by Rebecca Allen
- 1989: *The Little Death*, (short)  
Director: Matt Elson, Producer: Symbolics, Inc.
- 1992: *Batman Returns*, (feature)  
Director: Tim Burton, Producer: Warner Brothers
- 1993: *Cliffhanger*, (feature)  
Director: Renny Harlin, Producer: Carolco.
- 1994: *The Lion King*, (feature)  
Director: Allers / Minkoff, Producer: Disney.

## Applications of Behavioral Animations



- 1996: *From Dusk Till Dawn*, (feature)  
Director: Robert Rodriguez, Producer: Miramax
- 1996: *The Hunchback of Notre Dame*, (feature)  
Director: Trousdale / Wise, Producer: Disney.
- 1997: *Hercules*, (feature)  
Director: Clements / Musker, Producer: Disney.
- 1997: *Spawn*, (feature)  
Director: Dippé, Producer: Disney.

## Steering Behaviors



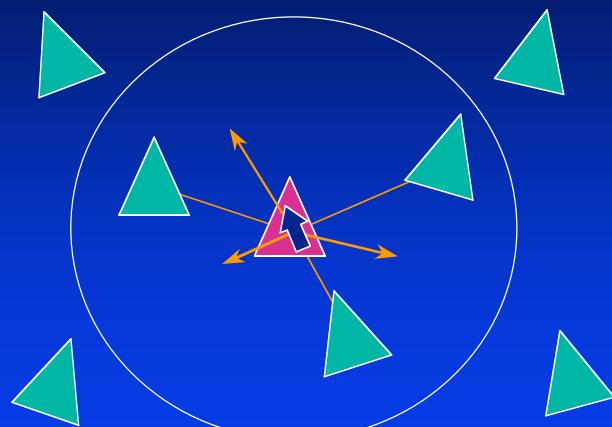
- seek or flee from a location
- pursuit and evasion
- arrival (position / velocity / time constraints)
- obstacle avoidance
- path / wall following
- group behaviors
  - unaligned collision avoidance
  - flocking (three components)

## Flocking (three component behaviors)

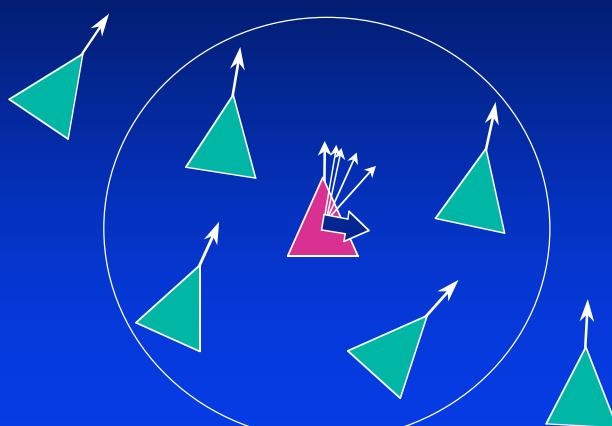


- Separation
  - steer to move away from nearby flockmates
- Alignment
  - steer toward average heading of nearby flockmates
  - (accelerate to match average velocity of nearby flockmates)
- Cohesion
  - steer toward the average position of nearby flockmates

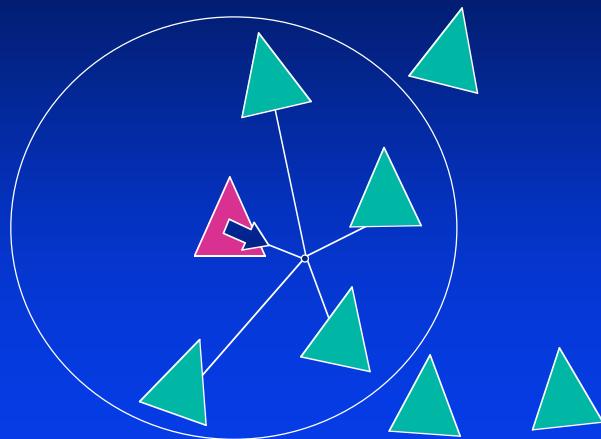
## Separation



## Alignment



## Aggregation



## Boids



- Obstacle avoidance
- Flocking
  - separation
  - alignment
  - cohesion
- Migratory (attraction / repulsion)

## Boids Web Page



<http://hmt.com/cwr/boids.html>

## Basic Hockey Player



- Physical model
  - point mass
  - limited force and velocity
  - collision modeling (as cylinder)
- Awareness of
  - position and velocity of players and puck
  - position of rink and markings
- Behaviors:
  - avoid rink walls and goal nets
  - chase loose puck, skate towards location...
- Assigned role
  - (forward, wing, defenseman, goalie)

## Hockey **Role Model**



- Defenseman
  - if you have the puck...
  - if your teammate has the puck...
  - if puck is within your zone:
    - discourage shot on goal
    - discourage pass to opponent
    - don't crowd goalie
  - do basic hockey play stuff

## Evolution of Behavior



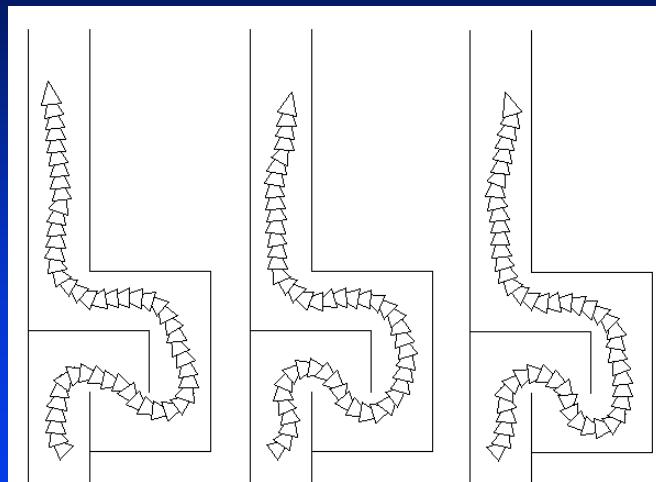
- Agent in simulated world
- Evolution of
  - behavioral controller
  - agent morphology (see Sims SIGGRAPH 94)
- Fitness based on agent's performance
  - objective fitness metric
  - competitive fitness

## Evolution of Corridor Following Behavior in a Noisy World

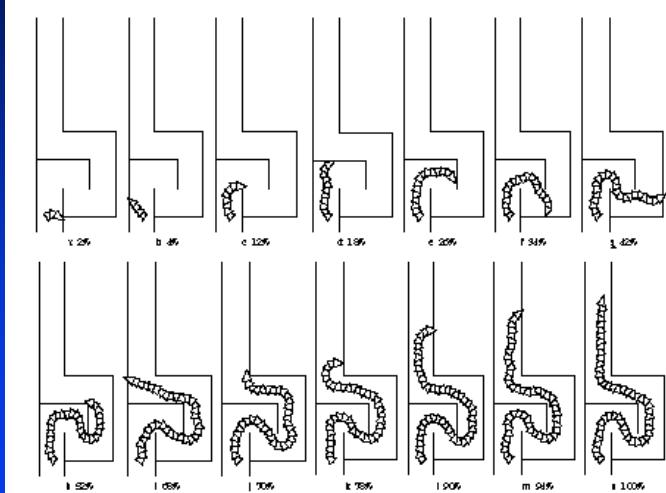


- Evolve controller for abstract vehicle
- Task: corridor following
  - noisy range sensors
  - noisy steering mechanism
- Evolution of sensor morphology

## Corridor Following: goal



## Corridor following: fitness



## Corridor Following: Results



- Works well
- Difficulty strongly related to the representation used
- “Competent” controllers easy to find
- Reliability of controllers is difficult to measure

## Corridor Following: Experimental Design



- Vehicle model
  - constant speed
  - limited steering angle
  - noisy sensors (arbitrary number & direction)
  - noisy steering mechanism
- Genetic Programming
  - hybrid steady-state model
  - worst of four noisy trials
  - population: 2000
  - size limit for evolved programs: 50

## Competition, Coevolution and the Game of Tag



- The game of tag
  - symmetrical pursuit and evasion
  - role reversal
- Goal: discover steering behavior for tag
- Method: emergence of behavior
  - evolution
  - competitive fitness
- Self-organization: no expert knowledge required

## The Results



- It works (more or less)
- An ecology of competing behaviors will arise
- All evolved behaviors have been sub-optimal

## The Experimental Design



- Vehicle model
  - kinematic model (not physically based)
  - constant speed
  - unlimited steering angle
  - no sensors: direct knowledge of environment
- Ad hoc scoring system for tag games
- Genetic programming
  - hybrid steady-state model
    - parents: tournament selection
    - removal: inverse tournament and uniform
  - competitive fitness: new versus several old
  - population: 1000
  - evolved program size: 50 to 100

The following paper:

**Flocks, Herds, and Schools: A Distributed Behavioral Model**

by  
Craig W. Reynolds

originally appeared in

**Computer Graphics**  
**Volume 21, Number 4**  
**SIGGRAPH '87 Conference Proceedings**

Edited by  
Maureen C. Stone

Published by the

**Association for Computing Machinery**

New York, New York  
1987

(pages 25 to 34)

# Flocks, Herds, and Schools: A Distributed Behavioral Model<sup>1</sup>

Craig W. Reynolds  
Symbolics Graphics Division

[obsolete addresses removed<sup>2</sup>]

## Abstract

The aggregate motion of a flock of birds, a herd of land animals, or a school of fish is a beautiful and familiar part of the natural world. But this type of complex motion is rarely seen in computer animation. This paper explores an approach based on simulation as an alternative to scripting the paths of each bird individually. The simulated flock is an elaboration of a particle system, with the simulated birds being the particles. The aggregate motion of the simulated flock is created by a distributed behavioral model much like that at work in a natural flock; the birds choose their own course. Each simulated bird is implemented as an independent actor that navigates according to its local perception of the dynamic environment, the laws of simulated physics that rule its motion, and a set of behaviors programmed into it by the "animator." The aggregate motion of the simulated flock is the result of the dense interaction of the relatively simple behaviors of the individual simulated birds.

*Categories and Subject Descriptors:* 1.2.10 [Artificial Intelligence]: Vision and Scene Understanding; 1.3.5 [Computer Graphics]: Computational Geometry and Object Modeling; 1.3.7 [Computer Graphics]: Three Dimensional Graphics and Realism-Animation; 1.6.3 [Simulation and Modeling]: Applications.

*General Terms:* Algorithms, design.b

*Additional Key Words, and Phrases:* flock, herd, school, bird, fish, aggregate motion, particle system, actor, flight, behavioral animation, constraints, path planning.

## Introduction

The motion of a flock of birds is one of nature's delights. Flocks and related synchronized group behaviors such as schools of fish or herds of land animals are both beautiful to watch and intriguing to contemplate. A flock<sup>\*</sup> exhibits many contrasts. It is made up of discrete birds yet overall motion seems fluid; it is simple in concept yet is so visually complex, it seems randomly arrayed and yet is magnificently synchronized. Perhaps most puzzling is the strong impression of intentional, centralized control. Yet all evidence indicates that flock motion must be merely the aggregate result of the actions of individual animals, each acting solely on the basis of its own local perception of the world.

One area of interest within computer animation is the description and control of all types of motion.

Computer animators seek both to invent wholly new types of abstract motion and to duplicate (or make variations on) the motions found in the real world. At first glance, producing an animated, computer graphic portrayal of a flock of birds presents significant difficulties. Scripting the path of a large number of individual objects using traditional computer animation techniques would be tedious. Given the complex paths that birds follow, it is doubtful this specification could be made without error. Even if a reasonable number of suitable paths could be described, it is unlikely that the constraints of flock motion could be maintained (for example, preventing collisions between all birds at each frame). Finally, a flock scripted in this manner would be hard to edit (for example, to alter the course of all birds for a portion of the animation). It is not impossible to script flock motion, but a better approach is needed for efficient, robust, and believable animation of flocks and related group motions.

This paper describes one such approach. This approach assumes a flock is simply the result of the interaction between the behaviors of individual birds. To simulate a flock we simulate the behavior of an individual bird (or at least that portion of the bird's behavior that allows it to participate in a flock). To support this behavioral "control structure," we must also simulate portions of the bird's perceptual mechanisms and aspects of the physics of aerodynamic flight. If this simulated bird model has the correct flock-member behavior, all that should be required to create a simulated flock is to create some instances of the simulated bird model and allow them to interact.<sup>\*\*</sup>

Some experiments with this sort of simulated flock are described in more detail in the remainder of this paper. The success and validity of these simulations is difficult to measure objectively. They do seem to agree well with certain criteria [25] and some statistical properties [23] of natural flocks and schools which have been reported by the zoological and behavioral sciences. Perhaps more significantly, many people who view these animated flocks immediately recognize them as a representation of a natural flock, and find them similarly delightful to watch.

## Our Foreflocks

The computer graphics community has seen simulated bird flocks before. The Electronic Theater at SIGGRAPH '85 presented a piece labeled "motion studies for a work in progress entitled 'Eurythmy'" [4] by Susan Amkraut, Michael Girard, and George Karl from the Computer Graphics Research Group of Ohio State University. In the film, a flock of birds flies up out of a minaret and, passing between a series of columns, flies down into a lazy spiral around a courtyard. All the while the birds slowly flap their wings and avoid collision with their flockmates.

That animation was produced using a technique completely unlike the one described in this paper and apparently not specifically intended for flock modeling. But the underlying concept is useful and interesting in its own right. The following overview is based on unpublished communications [3]. The software is informally called "the force field animation system." Force fields are defined by a  $3 \times 3$  matrix operator that transform from a point in space (where an object is located) to an acceleration vector: the birds trace paths along the "phase portrait" of the force field. There are "rejection forces" around each bird and around static objects. The force field associated with each object has a bounding box, so object interactions can be culled according to bounding box tests. An incremental, linear time algorithm finds bounding box intersections. The "animator" defines the space field(s) and sets the initial positions, orientations, and velocities of objects. The rest of the simulation is automatic.

Karl Sims of MIT's Media Lab has constructed some behaviorally controlled animation of groups of moving objects (spaceships, inchworms, and quadrupeds), but they are not organized as flocks [35].

Another author kept suggesting [28, 29, 30] implementing a flock simulation based on a distributed behavioral model.

## Particle Systems

The simulated flock described here is closely related to particle systems [27], which are used to represent dynamic "fuzzy objects" having irregular and complex shapes. Particle systems have been used to model fire, smoke, clouds, and more recently, the spray and foam of ocean waves [27]. Particle systems are collections of large numbers of individual particles, each having its own behavior. Particles are created, age, and die off. During their life they have certain behaviors that can alter the particle's own state, which consists of color, opacity, location, and velocity.

Underlying the boid flock model is a slight generalization of particle systems. In what might be called a "subobject system," Reeves's dot-like particles are replaced by an entire geometrical object consisting of a full local coordinate system and a reference to a geometrical shape model. The use of shapes instead of dots is visually significant, but the more fundamental difference is that individual subobjects have a more complex geometrical state; they now have orientation.

Another difference between boid flocks and particle systems is not as well defined. The behavior of boids is generally more complex than the behaviors for particles as described in the literature. The present boid behavior model might be about one or two orders of magnitude more complex than typical particle behavior. However this is a difference of degree, not of kind. And neither simulated behavior is nearly as complex as that of a real bird.

Also, as presented, particles in particle Systems do not interact with one another, although this is not ruled out by definition. But birds and hence boids must interact strongly in order to flock correctly. Boid behavior is dependent not only on internal state but also on external state.

## Actors and Distributed Systems

The behavioral model that controls the boid's flight and flocking is complicated enough that rather than use an ad hoc approach, it is worthwhile to pursue the most appropriate formal computational model. The behaviors will be represented as rules or programs in some sense, and the internal state of each boid must be held in some sort of data structure. It is convenient to encapsulate these behaviors and state as an object, in the sense of object-oriented programming systems [10, 11, 21]. Each instance of these objects needs a computational process to apply the behavioral programs to the internal data. The computational abstraction that combines process, procedure, and state is called an actor [12, 26, 2]. An actor is essentially a virtual computer that communicates with other virtual computers by passing messages. The actor model has been proposed as a natural structure for animation control by several authors [28, 13, 29, 18]. It seems particularly apt for situations involving interacting characters and behavior simulation. In the literature of parallel and distributed computer systems, flocks and schools are given as examples of robust self-organizing distributed systems [15].

## Behavioral Animation

Traditional hand-drawn cel animation was produced with a medium that was completely inert. Traditional computer animation uses an active medium (computers running graphics software), but most animation systems do not make much use of the computer's ability to automate motion design. Using

different tools, contemporary computer animators work at almost the same low level of abstraction as do cel animators. They tell their story by directly describing the motion of their characters. Shortcuts exist in both media: it is common for computer animators and cel animators to use helpers to interpolate between specified keyframes. But little progress has been made in automating motion description; it is up to the animator to translate the nuances of emotion and characterization into the motions that the character performs. The animator cannot simply tell the character to "act happy" but must tediously specify the motion that conveys happiness.

Typical computer animation models only the shape and physical properties of the characters, whereas behavioral or character-based animation seeks to model the behavior of the character. The goal is for such simulated characters to handle many of the details of their actions, and hence their motions. These behaviors include a whole range of activities from simple path planning to complex "emotional" interactions between characters. The construction of behavioral animation characters has attracted many researchers [19, 21, 13, 14, 29, 30, 41, 40], but it is still a young field in which more work is needed.

Because of the detached nature of the control, the person who creates animation with character simulation might not strictly be an animator. Traditionally, the animator is directly responsible for all motion in animation production [40]. It might be more proper to call the person who directs animation via simulated characters a meta-animator, since the animator is less a designer of motion and more a designer of behavior. These behaviors, when acted out by the simulated characters, lead indirectly to the final action. Thus the animator's job becomes somewhat like that of a theatrical director: the character's performance is the indirect result of the director's instructions to the actor. One of the charming aspects of the work reported here is not knowing how a simulation is going to proceed from the specified behaviors and initial conditions; there are many unexpected, pleasant surprises. On the other hand, this charm starts to wear thin as deadlines approach and the unexpected annoyances pop up. This author has spent a lot of time recently trying to get uncooperative flocks to move as intended ("these darn boids seem to have a mind of their own!").

## Geometric Flight

A fundamental part of the boid model is the geometric ability to fly. The motion of the members of a simulated school or herd can be considered a type of "flying" by glossing over the considerable intricacies of wing, fin, and leg motion (and in the case of herds, by restricting freedom of motion in the third dimension). In this paper the term geometric flight refers to a certain type of motion along a path: a dynamic, incremental, rigid geometrical transformation of an object, moving along and tangent to a 3D curve. While the motion is rigid, the object's underlying geometric model is free to articulate or change shape within this "flying coordinate system." Unlike more typical animated motion along predefined spline curves, the shape of a flight path is not specified in advance.

Geometric flight is based on incremental translations along the object's "forward direction," its local positive Z axis. These translations are intermixed with steering-rotations about the local X and Y axes (pitch and yaw), which realign the global orientation of the local Z axis. In real flight, turning and moving happen continuously and simultaneously. Incremental geometric flight is a discrete approximation of this; small linear motions model a continuous curved path. In animation the motion must increment at least once per frame. Running the simulation at a higher rate can reduce the discrete sampling error of the flight model and refine the shape of motion blur patterns.

Flight modeling makes extensive use of the object's own coordinate system. Local space represents the

"boid's eye view;" it implies measuring things relative to the boid's own position and orientation. In Cartesian terms, the left/right axis is X, up/down is Y, and forward/back is Z. The conversion of geometric data between the local and global reference frames is handled by the geometric operators `localize` and `globalize`. It is convenient to use a local scale so that the unit of length of the coordinate system is one body length. Biologists routinely specify flock and school statistics in terms of body lengths.

Geometric flight models conservation of momentum. An object in flight tends to stay in flight. There is a simple model of viscous speed damping, so even if the boid continually accelerates in one direction, it will not exceed a certain maximum speed. A minimum speed can also be specified but defaults to zero. A maximum acceleration, expressed as a fraction of the maximum speed, is used to truncate over-anxious requests for acceleration, hence providing for smooth changes of speed and heading. This is a simple model of a creature with a finite amount of available energy.

Many physical forces are not supported in the current boid model. Gravity is modeled but used only to define banking behavior. It is defined procedurally to allow the construction of arbitrarily shaped fields. If each boid was accelerated by gravity each frame, it would tend to fall unless gravity was countered by lift or buoyancy. Buoyancy is aligned against gravity, but aerodynamic lift is aligned with the boid's local "up" direction and related to velocity. This level of modeling leads to effects like normally level flight, going faster when flying down (or slower up), and the "stall" maneuver. The speed limit parameter could be more realistically modeled as a frictional drag, a backward pointing force related to velocity. In the current model steering is done by directing the available thrust in the appropriate direction. It would be more realistic to separately model the tangential thrusting forces and the lateral steering forces, since they normally have different magnitudes.

## Banking

Geometric flight relates translation, pitch, and yaw, but does not constrain roll, the rotation about the local Z axis. This degree of freedom is used for banking-rolling the object to align the local Y axis with the (local XY component of the total) acceleration acting upon it. Normally banking is based on the lateral component of the acceleration, but the tangential component can be used for certain applications. The lateral components are from steering and gravity. In straight flight there is no radial force, so the gravitational term dominates and banking aligns the object's -Y axis with "gravitational down" direction. When turning, the radial component grows larger and the "accelerational down" direction swings outward, like a pendulum hanging from the flying object. The magnitude of the turning acceleration varies directly with the object's velocity and with the curvature of its path (so inversely with the radius of its turn). The limiting case of infinite velocity resembles banking behavior in the absence of gravity. In these cases the local + Y (up) direction points directly at the center of curvature defined by the current turn.

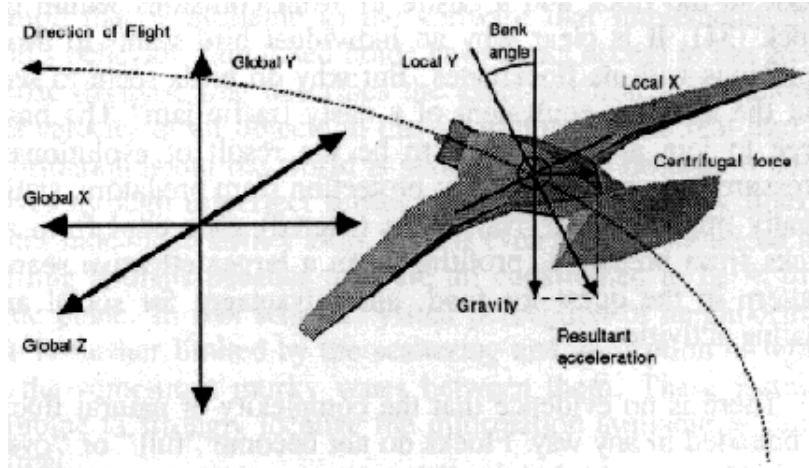


Figure 1.

With correct banking (what pilots call a coordinated turn) the object's local space remains aligned with the "perceptual" or "accelerational" coordinate system. This has several advantages: it simplifies the bird's (or pilot's) orientation task, it keeps the lift from the airfoils of the wings pointed in the most efficient direction ("accelerational up"), it keeps the passengers coffee in their cups, and most importantly for animation, it makes the flying boid fit the viewer's expectation of how flying objects should move and orient themselves. On the other hand, realism is not always the goal in animation. By simply reversing the angle of bank we obtain a cartoony motion that looks like the object is being flung outward by the centrifugal force of the turn.

## Boids and Turtles

The incremental mixing of forward translations and local rotations that underlies geometric flight is the basis of "turtle graphics" in the programming language Logo [5]. Logo was first used as an educational tool to allow children to learn experimentally about geometry, arithmetic, and programming [22]. The Logo turtle was originally a little mechanical robot that crawled around on large sheets of paper laid on the classroom floor, drawing graphic figures by dragging a felt tip marker along the paper as it moved. Abstract turtle geometry is a system based on the frame of reference of the turtle, an object that unites position and heading. Under program control the Logo turtle could move forward or back from its current position, turn left or right from its current heading, or put the pen up or down on the paper. The turtle geometry has been extended from the plane onto arbitrary manifolds and into 3D space [1]. These "3d turtles" and their paths are exactly equivalent to the boid objects and their flight paths.

## Natural Flocks, Herds, and Schools

"...and the thousands off fishes moved as a huge beast, piercing the water. They appeared united, inexorably bound to a common fate. How comes this unity?"

--Anonymous, 17th century (from Shaw)

For a bird to participate in a flock, it must have behaviors that allow it to coordinate its movements with those of its flockmates. These behaviors are not particularly unique; all creatures have them to some degree. Natural flocks seem to consist of two balanced, opposing behaviors: a desire to stay close to the

flock and a desire to avoid collisions within the flock [34]. It is clear why an individual bird wants to avoid collisions with its flockmates. But why do birds seem to seek out the airborne equivalent of a nasty traffic jam? The basic urge to join a flock seems to be the result of evolutionary pressure from several factors: protection from predators, statistically improving survival of the (shared) gene pool from attacks from predators, profiting from a larger effective search pattern in the quest for food, and advantages for social and mating activities [33].

There is no evidence that the complexity of natural flocks is bounded in any way. Flocks do not become "full" or "overloaded" as new birds join. When herring migrate toward their spawning grounds, they run in schools extending as long as 17 miles and containing millions of fish [32]. Natural flocks seem to operate in exactly the same fashion over a huge range of flock populations. It does not seem that an individual bird can be paying much attention to each and every one of its flockmates. But in a huge flock spread over vast distances, an individual bird must have a localized and filtered perception of the rest of the flock. A bird might be aware of three categories: itself, its two or three nearest neighbors, and the rest of the flock [23].

These speculations about the "computational complexity" of flocking are meant to suggest that birds can flock with any number of flockmates because they are using what would be called in formal computer science a constant time algorithm. That is, the amount of "thinking" that a bird has to do in order to flock must be largely independent of the number of birds in the flock. Otherwise we would expect to see a sharp upper bound on the size of natural flocks when the individual birds became overloaded by the complexity of their navigation task. This has not been observed in nature.

Contrast the insensitivity to complexity of real flocks with the situation for the simulated flocks described below. The complexity of the flocking algorithm described is basically  $O(N^2)$ . That is, the work required to run the algorithm grows as the square of the flock's population. We definitely do see an upper bound on the size of simulated flocks implemented as described here. Some techniques to address this performance issue are discussed in the section Algorithmic Considerations.

## Simulated Flocks

To build a simulated flock, we start with a boid model that supports geometric flight. We add behaviors that correspond to the opposing forces of collision avoidance and the urge to join the flock. Stated briefly as rules, and in order of decreasing precedence, the behaviors that lead to simulated flocking are:

1. Collision Avoidance: avoid collisions with nearby flockmates
2. Velocity Matching: attempt to match velocity with nearby flockmates
3. Flock Centering: attempt to stay close to nearby flockmates

Velocity is a vector quantity, referring to the combination of heading and speed. The manner in which the results from each of these behaviors are reconciled and combined is significant and is discussed in more detail later. Similarly, the meaning nearby in these rules is key to the flocking process. This is also discussed in more detail later, but generally one boid's awareness of another is based on the distance and direction of the offset vector between them.

Static collision avoidance and dynamic velocity matching are complementary. Together they ensure that

the members of a simulated flock are free to fly within the crowded skies of the flock's interior without running into one another. Collision avoidance is the urge to steer a way from an imminent impact. Static collision avoidance is based on the relative position of the flockmates and ignores their velocity. Conversely, velocity matching is based only on velocity and ignores position. It is a predictive version of collision avoidance: if the boid does a good job of matching velocity with its neighbors, it is unlikely that it will collide with any of them any time soon. With velocity matching, separations between boids remains approximately invariant with respect to ongoing geometric flight. Static collision avoidance serves to establish the minimum required separation distance; velocity matching tends to maintain it.

Flock centering makes a boid want to be near the center of the flock. Because each boid has a localized perception of the world, "center of the flock" actually means the center of the nearby flockmates. Flock centering causes the boid to fly in a direction that moves it closer to the centroid of the nearby boids. If a boid is deep inside a flock, the population density in its neighborhood is roughly homogeneous; the boid density is approximately the same in all directions. In this case, the centroid of the neighborhood boids is approximately at the center of the neighborhood, so the flock centering urge is small. But if a boid is on the boundary of the flock, its neighboring boids are on one side. The centroid of the neighborhood boids is displaced from the center of the neighborhood toward the body of the flock. Here the flock centering urge is stronger and the flight path will be deflected somewhat toward the local flock center.

Real flocks sometimes split apart to go around an obstacle. To be realistic, the simulated flock model must also have this ability. Flock centering correctly allows simulated flocks to bifurcate. As long as an individual boid can stay close to its nearby neighbors, it does not care if the rest of the flock turns away. More simplistic models proposed for flock organization (such as a central force model or a follow the designated leader model) do not allow splits.

The flock model presented here is actually a better model of a school or a herd than a flock. Fish in murky water (and land animals with their inability to see past their herdmates) have a limited, short-range perception of their environment. Birds, especially those on the outside of a flock, have excellent long-range "visual perception." Presumably this allows widely separated flocks to join together. If the flock centering urge was completely localized, when two flocks got a certain distance apart they would ignore each other. Long-range vision seems to play a part in the incredibly rapid propagation of a maneuver wave" through a flock of birds. It has been shown that the speed of propagation of this wavefront reaches three times the speed implied by the measured startle reaction time of the individual birds. The explanation advanced by Wayne Potts is that the birds perceive the motion of the oncoming "maneuver wave" and time their own turn to match it [25]. Potts refers to this as the "chorus line" hypothesis.

## Arbitrating Independent Behaviors

The three behavioral urges associated with flocking (and others to be discussed below) each produce an isolated suggestion about which way to steer the boid. These are expressed as acceleration requests. Each behavior says: "if I were in charge, I would accelerate in that direction." The acceleration request is in terms of a 3D vector that, by system convention, is truncated to unit magnitude or less. Each behavior has several parameters that control its function; one is a "strength," a fractional value between zero and one that can further attenuate the acceleration request. It is up to the navigation module of the boid brain to collect all relevant acceleration requests and then determine a single behaviorally desired acceleration. It must combine, prioritize, and arbitrate between potentially conflicting urges. The pilot module takes the acceleration desired by the navigation module and passes it to the flight module, which attempts to

fly in that direction.

The easiest way to combine acceleration requests is to average them. Because of the included "strength" factors, this is actually a weighted average. The relative strength of one behavior to another can be defined this way, but it is a precarious interrelationship that is difficult to adjust. An early version of the boid model showed that navigation by simple weighted averaging of acceleration requests works "pretty well." A boid that chooses its course this way will fly a reasonable course under typical conditions. But in critical situations, such as potential collision with obstacles, conflicts must be resolved in a timely manner. During high-speed flight, hesitation or indecision is the wrong response to a brick wall dead ahead.

The main cause of indecision is that each behavior might be shouting advice about which way to turn to avoid disaster, but if those acceleration requests happen to lie in approximately opposite directions, they will largely cancel out under a simple weighted averaging scheme. The boid would make a very small turn and so continue in the same direction, perhaps to crash into the obstacle. Even when the urges do not cancel out, averaging leads to other problems. Consider flying over a gridwork of city streets between the skyscrapers; while "fly north" or "fly east" might be good ideas, it would be a bad idea to combine them as "fly northeast."

Techniques from artificial intelligence, such as expert systems, can be used to arbitrate conflicting opinions. However, a less complex approach is taken in the current implementation. Prioritized acceleration allocation is based on a strict priority ordering of all component behaviors, hence of the consideration of their acceleration requests. (This ordering can change to suit dynamic conditions.) The acceleration requests are considered in priority order and added into an accumulator. The magnitude of each request is measured and added into another accumulator. This process continues until the sum of the accumulated magnitudes gets larger than the maximum acceleration value, which is a parameter of each boid. The last acceleration request is trimmed back to compensate for the excess of accumulated magnitude. The point is that a fixed amount of acceleration is under the control of the navigation module; this acceleration is parceled out to satisfy the acceleration request of the various behaviors in order of priority. In an emergency the acceleration would be allocated to satisfy the most pressing needs first; if all available acceleration is "used up," the less pressing behaviors might be temporarily unsatisfied. For example, the flock centering urge could be correctly ignored temporarily in favor of a maneuver to avoid a static obstacle.

## Simulated Perception

The boid model does not directly simulate the senses used by real animals during flocking (vision and hearing) or schooling (vision and fishes' unique "lateral line" structure that provides a certain amount of pressure imaging ability [23, 24]). Rather the perception model tries to make available to the behavior model approximately the same information that is available to a real animal as the end result of its perceptual and cognitive processes.

This is primarily a matter of filtering out the surplus information that is available to the software that implements the boid's behavior. Simulated boids have direct access to the geometric database that describes the exact position, orientation, and velocity of all objects in the environment. The real bird's information about the world is severely limited because it perceives through imperfect senses and because its nearby flockmates hide those farther away. This is even more pronounced in herding animals because they are all constrained to be in the same plane. In fish schools, visual perception of

neighboring fish is further limited by the scattering and absorption of light by the sometimes murky water between them. These factors combine to strongly localize the information available to each animal.

Not only is it unrealistic to give each simulated boid perfect and complete information about the world, it is just plain wrong and leads to obvious failures of the behavior model. Before the current implementation of localized flock centering behavior was implemented, the flocks used a central force model. This leads to unusual effects such as causing all members of a widely scattered flock to simultaneously converge toward the flock's centroid. An interesting result of the experiments reported in this paper is that the aggregate motion that we intuitively recognize as "flocking" (or schooling or herding) depends upon a limited, localized view of the world.

The behaviors that make up the flocking model are stated in terms of "nearby flockmates." In the current implementation, the neighborhood is defined as a spherical zone of sensitivity centered at the boid's local origin. The magnitude of the sensitivity is defined as an inverse exponential of distance. Hence the neighborhood is defined by two parameters: a radius and exponent. There is reason to believe that this field of sensitivity should realistically be exaggerated in the forward direction and probably by an amount proportional to the boid's speed. Being in motion requires an increased awareness of what lies ahead, and this requirement increases with speed. A forward-weighted sensitivity zone would probably also improve the behavior in the current implementation of boids at the leading edge of a flock, who tend to get distracted by the flock behind them. Because of the way their heads and eyes are arranged, real birds have a wide field of view (about 300 degrees), but the zone of overlap from both eyes is small (10 to 15 degrees). Hence the bird has stereo depth perception only in a very small, forward-oriented cone. Research is currently under way on models of forward-weighted perception for boids.

In an early version of the flock model, the metrics of attraction and repulsion were weighted linearly by distance. This spring-like model produced a bouncy flock action, fine perhaps for a cartoony characterization, but not very realistic. The model was changed to use an inverse square of the distance. This more gravity-like model produced what appeared to be a more natural, better damped flock model. This correlated well with the carefully controlled quantitative studies that Brian Partridge made of the spatial relationships of schooling fish [23]; he found that "a fish is much more strongly influenced by its near neighbors than it is by the distant members of the school. The contribution of each fish to the [influence] is inversely proportional to the square or the cube of the distance." In previous work he and colleagues [23, 24] demonstrated that fishes school based on information from both their visual system and from their "lateral line" organ which senses pressure waves. The area of a perspective image of the silhouette of an object (its "visual angle") varies inversely with the square of its distance, and that pressure waves traveling through a 3D medium like water fall off inversely with the cube of the distance.

The boid perception model is quite ad hoc and avoids actually simulating vision. Artificial vision is an extremely complex problem [38] and is far beyond the scope of this work. But if boids could "see" their environment, they would be better at path planning than the current model. It is possible to construct simple maze like shapes that would confuse the current boid model but would be easily solved by a boid with vision.

## **Impromptu Flocking**

The flocking model described above gives boids an eagerness to participate in an acceptable approximation of flock like motion. Boids released near one another begin to flock together, cavorting

and jostling for position. The boids stay near one another (flock centering) but always maintain prudent separation from their neighbors' (collision avoidance), and the flock quickly becomes "polarized"-its members heading in approximately the same direction at approximately the same speed (velocity marching); when they change direction they do it in synchronization. Solitary boids and smaller flocks join to become larger flocks, and in the presence of external obstacles (discussed below), larger flocks can split into smaller flocks.

For each simulation run, the initial position (within a specified ellipsoid), heading, velocity, and various other parameters of the boid model are initialized to values randomized within specified distributions. A restartable random number generator is used to allow repeatability. This randomization is not required; the boids could just as well start out arranged in a regular pattern, all other aspects of the flock model are completely deterministic and repeatable.

When the simulation is run, the flock's first action is a reaction to the initial conditions. If the boids started out too closely crowded together, there is an initial "flash expansion" where the mutual desire to avoid collision drives the boids radially away from the site of the initial over-pressure. If released in a spherical shell with a radius smaller than the "neighborhood" radius, the boids contract toward the sphere's center; otherwise they begin to coalesce into small flockettes that might themselves begin to join together. If the boids are confined within a certain region, the smaller flocks eventually conglomerate into a single flock if left to wander long enough.

## Scripted Flocking

The behaviors discussed so far provide for the ability of individual birds to fly and participate in happy aimless flocking. But to combine flock simulations with other animated action, we need more direct control over the flock. We would like to direct specific action at specific times (for example, "the flock enters from the left at :02.3 seconds into the sequence, turns to fly directly upward at :03.5, and is out of the frame at :04.0").

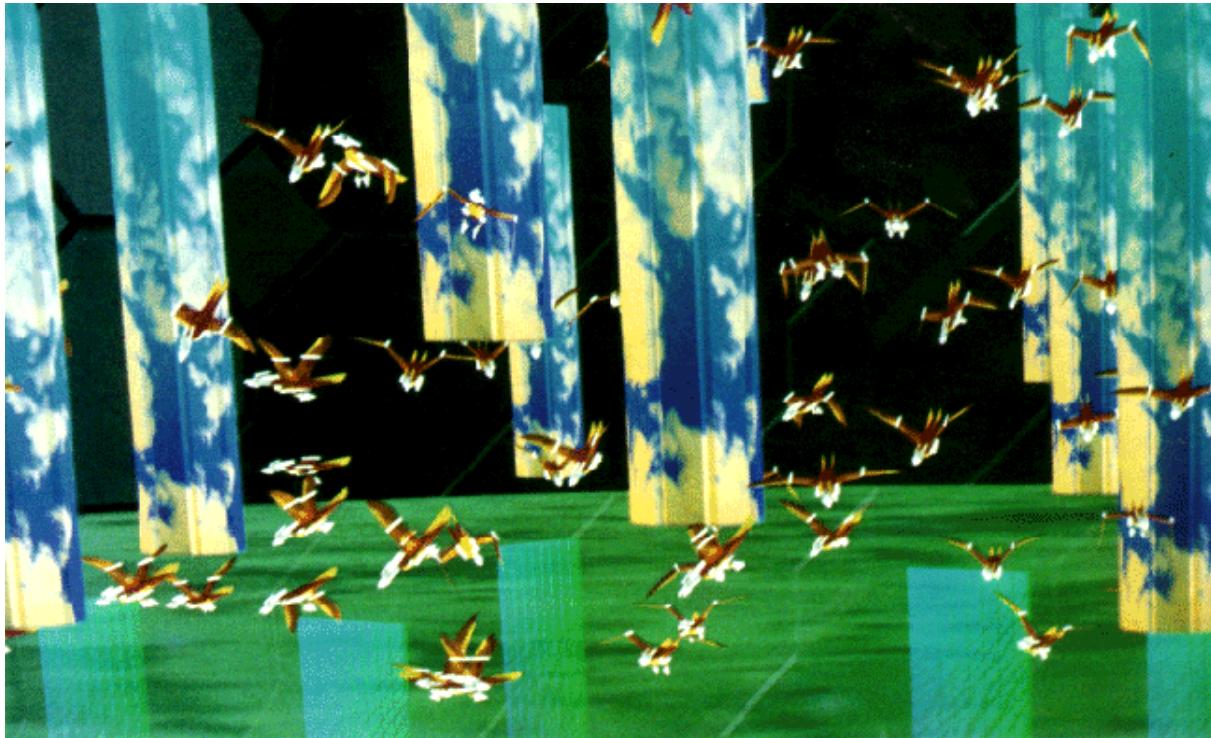
The current implementation of the boid model has several facilities to direct the motion and timing of the flock action. First, the simulations are run under the control of a general-purpose animation scripting system [36]. The details of that scripting system are not relevant here except that, in addition to the typical interactive motion control facilities, it provides the ability to schedule the invocation of user-supplied software (such as the flock model) on a frame-by-frame basis. This scripting facility is the basic tool used to describe the timing of various flock actions. It also allows flexible control over the time-varying values of parameters, which can be passed down to the simulation software. Finally the script is used to set up and animate all nonbehavioral aspects of the scene, such as backgrounds, lighting, camera motion, and other visible objects.

The primary tool for scripting the flock's path is the migratory urge built into the boid model. In the current model this urge is specified in terms of a global target, either as a global direction (as in "going Z for the winter") or as a global position-a target point toward which all birds fly. The model computes a bounded acceleration that incrementally turns the boid toward its migratory target.

With the scripting system, we can animate a dynamic parameter whose value is a global position vector or a global direction vector. This parameter can be passed to the flock, which can in turn pass it along to all boids, each of which sets its own "migratory goal register." Hence the global migratory behavior of all birds can be directly controlled from the script. (Of course, it is not necessary to alter all boids at the

same time, for example, the delay could be a function of their present position in space. Real flocks do not change direction simultaneously [25], but rather the turn starts with a single bird and spreads quickly across the flock like a shock wave.)

We can lead the flock around by animating the goal point along the desired path, somewhat ahead of the flock. Even if the migratory goal point is changed abruptly the path of each boid still is relatively smooth because of the flight model's simulated conservation of momentum. This means that the boid's own flight dynamics implement a form of smoothing interpolation between "control points."



## Avoiding Environmental Obstacles

The most interesting motion of a simulated flock comes from interaction with other objects in the environment. The isolated behavior of a flock tends to reach a steady state and becomes rather sterile. The flock can be seen as a relaxation solution to the constraints implied by its behaviors. For example, the conflicting urges of flock centering and collision avoidance do not lead to constant back and forth motion, but rather the boids eventually strike a balance between the two urges (the degree of damping controls how soon this balance is reached). Environmental obstacles and the boid's attempts to navigate around them increase the apparent complexity of the behavior of the flock. (In fact the complexity of real flocks might be due largely to the complexity of the natural environment.)

Environmental obstacles are also important from the standpoint of modeling the scene in which we wish to place the flock. If the flock is scripted to fly under a bridge and around a tree, we must be able to represent the geometric shape and dimension of these obstacles. The approach taken here is to independently model the "shape for rendering" and the "shape for collision avoidance." The types of shapes currently used for environmental obstacles are much less complicated than the models used for rendering of computer graphic models. The current work implements two types of shapes of

environmental collision avoidance. One is based on the force field concept, which works in undemanding situations but has some shortcomings. The other model called steer-to-avoid is more robust and seems closer in spirit to the natural mechanism.

The force field model postulates a field of repulsion force emanating from the obstacle out into space; the boids are increasingly repulsed as they get closer to the obstacle. This scheme is easy to model; the geometry of the field is usually fairly simple and so an avoidance acceleration can be directly calculated from the field equation. These models can produce good results, such as in "Eurythmy" [4], but they also have drawbacks that are apparent on close examination. If a boid approaches an obstacle surrounded by a force field at an angle such that it is exactly opposite to the direction of the force field, the boid will not turn away. In this case the force field serves only to slow the boid by accelerating it backwards and provides no side thrust at all. The worst reaction to an impending collision is to fail to turn. Force fields also cause problems with "peripheral vision." The boid should notice and turn away from a wall as it flies toward it, but the wall should be ignored if the boid is flying alongside it. Finally, force fields tend to be too strong close up and too weak far away; avoiding an obstacle should involve long-range planning rather than panicky corrections at the last minute.

Steer-to-avoid is a better simulation of a natural bird guided by vision. The boid considers only obstacles directly in front of it. (It finds the intersection, if any, of its local Z axis with the obstacle.) Working in local perspective space, it finds the silhouette edge of the obstacle closest to the point of eventual impact. A radial vector is computed which will aim the boid at a point one body length beyond that silhouette edge (see figure 2). Currently steer-to-avoid has been implemented for several obstacle shapes: spheres, cylinders, planes, and boxes. Collision avoidance for arbitrary convex polyhedral obstacles is being developed.

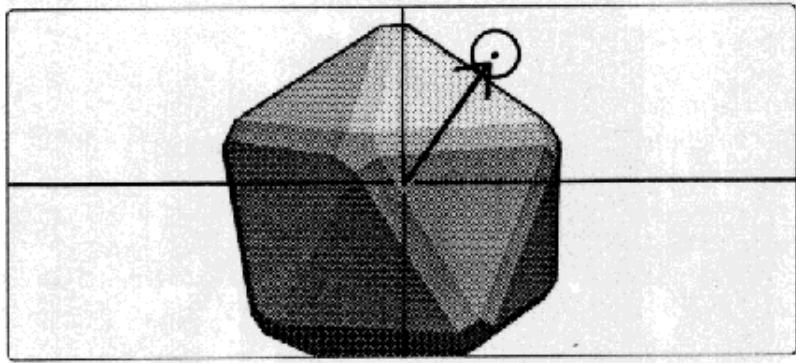


Figure 2.

Obstacles are not necessarily fixed in space; they can be animated around by the script during the animation. Or more interestingly, the obstacles can be behavioral characters. Sparrows might flock around a group of obstacles that is in fact a herd of elephants. Similarly, behavioral obstacles might not merely be in the way; they might be objects of fear such as predators. It has been noted [25] that natural flocking instincts seem to be sharpened by predators.

## Other Applications of the Flock Model

The model of polarized noncolliding aggregate motion has many applications, visual simulation of bird

flocks in computer animation being one. Certain modifications yield a fish school model. Further modifications, such as [imitation to a 2D surface and the ability to follow the terrain, lead to a herd model. Imagine a herd of PODA-style legged creatures [9], using Karl Sims' techniques for locomotion over uneven, complex terrain [35]. Other applications are less obvious. Traffic patterns, such as the flow of cars on a freeway, is a flock-like motion. There are specialized behaviors, such as being constrained to drive within the lanes, but the basic principles that keep boids from colliding are just as applicable on the freeway. We could imagine creating crowds of "extras" (human or otherwise) for feature films. However the most fun are the offbeat combinations possible in computer graphics by mixing and matching: a herd of pogo sticks, a flock of Pegasus-like winged horses, or a traffic jam of spaceships on a 3D interplanetary highway.

One serious application would be to aid in the scientific investigation of flocks, herds, and schools. These scientists must work almost exclusively in the observational mode; experiments with natural flocks and schools are difficult to perform and are likely to disturb the behaviors under study. It might be possible, using a more carefully crafted model of the realistic behavior of a certain species of bird, to perform controlled and repeatable experiments with "simulated natural flocks." A theory of flock organization can be unambiguously tested by implementing a distributed behavioral model and simply comparing the aggregate motion of the simulated flock with the natural one.

## Algorithmic Considerations

A naive implementation of the basic flocking algorithm would grow in complexity as the order of the square of the flock's population (" $O(N^2)$ "). Basically this is because each boid must reason about each of the other boids, even if only to decide to ignore it. This does not say the algorithm is slow or fast, merely that as the size of the problem (total population of the flock) increases, the complexity increases even faster. Doubling the number of boids quadruples the amount of time taken.

However, as stated before, real birds are probably not as sensitive to the total flock population. This gives hope that the simulated boid could be taught to navigate independently of the total population. Certainly part of the problem is that we are trying to run the simulation of the whole flock on a single computer. The natural solution is to use distributed processing, as the real flock does. If we used a separate processor for each boid, then even the naive implementation of the flocking algorithm would be  $O(N)$ , or linear with respect to the population. But even that is not good enough. It still means that as more boids are added to the flock, the complexity of the problem increases.

What we desire is a constant time algorithm, one that is insensitive to the total population. Another way to say this is that an  $N^2$  algorithm would be OK if there was an efficient way to keep  $N$  very small. Two approaches to this goal are currently under investigation. One is dynamic spatial partitioning of the flock; the boids are sorted into a lattice of "bins" based on their position in space. A boid trying to navigate inside the flock could get quick access to the flockmates that are physically nearby by examining the "bins" near its current position. Another approach is to do incremental collision detection ( $x$  'nearness testing'). General collision detection is another  $N^2$  algorithm, but if one does collision detection incrementally, based on a partial solution that described the situation just a moment before, then the algorithm need worry only about the changes and so can run much faster, assuming that the incremental changes are small. The incremental collision detection algorithm used in Girard's PODA system [9] apparently achieves constant time performance in the typical case.

## Computing Environment

The boids software was written in Symbolics Common Lisp. The code and animation were produced on a Symbolics 3600 Lisp Machine, a high-performance personal computer. The flock software is implemented in Flavors, the object-oriented programming extensions to Symbolics Common Lisp. The geometric aspects of the system are layered upon S-Geometry, an interactive geometric modeler [37]. Boids are based on the flavor 3D:OBJECT, which provides their geometric abilities. The flock simulations are invoked from scripts created and animated with the S Dynamics [36] animation system, which also provided the real-time playback facility used to view the motion tests. The availability of this graphical toolkit allowed the author to focus immediately on the issues unique to this project. One example of the value of this substrate is that the initial version of the flock model, including implementation, testing, debugging, and the production of seven short motion tests was accomplished in the ten days before the SIGGRAPH '86 conference.

The boid software has not been optimized for speed. But this report would be incomplete without a rough estimate of the actual performance of the system. With a flock of 80 boids, using the naive  $O(N^2)$  algorithm (and so 6400 individual boid-to-boid comparisons), on a single Lisp Machine without any special hardware accelerators, the simulation ran for about 95 seconds per frame. A ten-second (300 frame) motion test took about eight hours of real time to produce.

## Future Work

This paper has largely ignored the internal animation of the geometrical model that provides the visual representation of the boid. The original motion tests produced with these models all show flocks of little abstract rigid shapes that might be paper airplanes. There was no flapping of wings nor turning of heads, and there was certainly no character animation. These topics are all important and pertinent to believable animation of simulated flocks. But the underlying abstract nature of flocking as polarized, noncolliding aggregate motion is largely independent of these issues of internal shape change and articulation. This notion is supported by the fact that most viewers of these simulations identify the motion of these abstract objects as "flocking" even in the absence of any internal animation.

But doing a believable job of melding these two aspects of the motion is more than a matter of concatenating the action of an internal animation cycle for the character with the motion defined by geometrical flight. There are important issues of synchronization between the current state of the flight dynamics model, and the amplitude and frequency of the wing motion cycle. Topics of current development include internal animation, synchronization, and interfaces between the simulation-based flock model and other more traditional, interactive animation scripting systems. We would like to allow a skilled computer animator to design a bird character and define its "wing flap cycle" using standard interactive modeling and scripting techniques, and then be able to take this cyclic motion and "plug it in" to the flock simulation model causing the boids in the flock to fly according to the scripted cycle.

The behaviors that have been discussed in this paper are all simplistic, isolated behaviors of low complexity. The boids have a geometric and kinematic state, but they have no significant mental state. Real animals have more elaborate, abstract behaviors than a simple desire to avoid a painful collision: they have more complex motivations than a simple desire to fly to a certain point in space. More interesting behavior models would take into account hunger, finding food, fear of predators, a periodic need to sleep, and so on. Behavior models of this type have been created by other investigators [6, 19, 21], but they have not yet been implemented for the boid model described here.

## **Conclusion**

This paper has presented a model of polarized, noncolliding aggregate motion, such as that of flocks, herds, and schools. The model is based on simulating the behavior of each bird independently. Working independently, the birds try both to stick together and avoid collisions with one another and with other objects in their environment. The animations showing simulated flocks built from this model seem to correspond to the observer's intuitive notion of what constitutes "flock-like motion." However it is difficult to objectively measure how valid these simulations are. By comparing behavioral aspects of the simulated flock with those of natural flocks, we are able improve and refine the model. But having approached a certain level of realism in the model, the parameters of the simulated flock can be altered at will by the animator to achieve many variations on flock-like behavior.

## **Acknowledgements**

I would like to thank flocks, herds, and schools for existing; nature is the ultimate source of inspiration for computer graphics and animation. I would also like to acknowledge the contributions to this research provided by workers in a wonderfully diverse collection of pursuits:

To the natural sciences of behavior, evolution, and zoology: for doing the hard work, the Real Science, on which this computer graphics approximation is based. To the Logo group who invented the appropriate geometry, and so put us in the driver's seat. To the Actor semantics people who invented the appropriate control structure, and so gave the boid a brain. To the many developers of modern Lisp who invented the appropriate programming language. To my past and present colleagues at MIT, III, and Symbolics who have patiently listened to my speculations about flocks for years and years before I made my first boid fly. To the Graphics Division of Symbolics, Inc., who employ me, put up with my nasty disposition, provide me with fantastic computing and graphics facilities, and have generously supported the development of the work described here. And to the field of computer graphics, for giving professional respectability to advanced forms of play such as reported in this paper.



## **References**

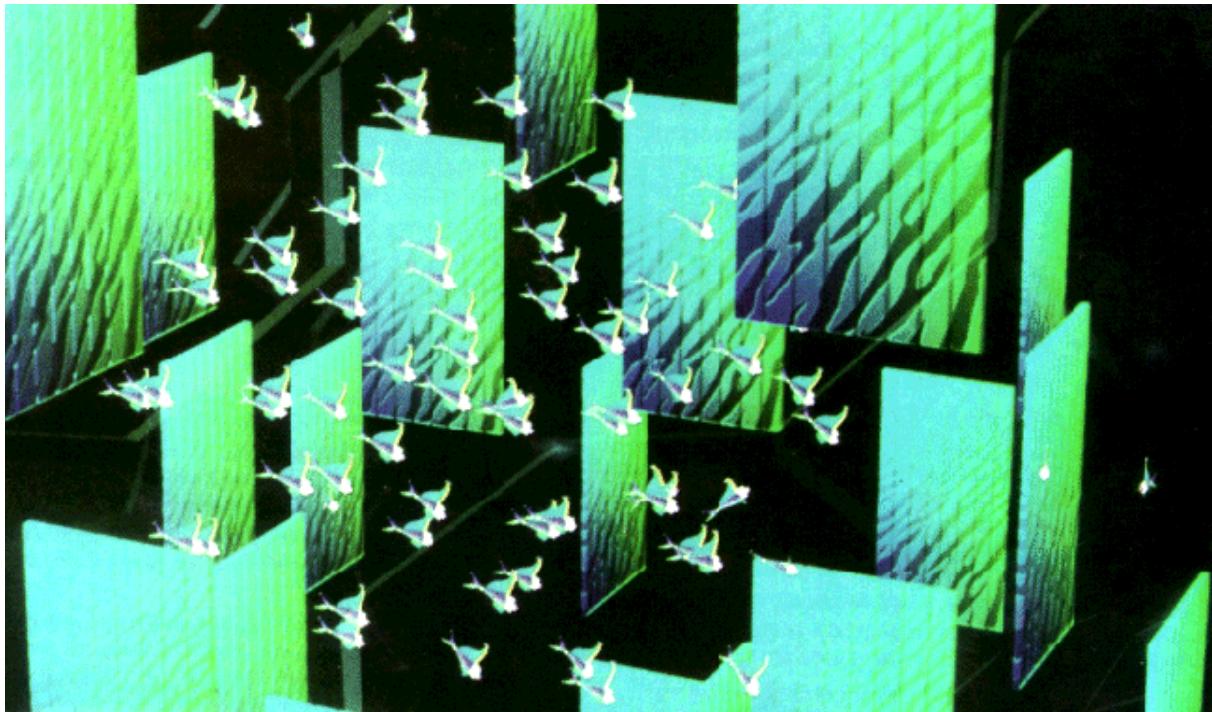
1. Abelson, H., and diSessa, A., "Maneuvering a Three Dimensional Turtle" in *Turtle Geometry*:

- The Computer as a Medium for Exploring Mathematics, The MIT Press, Cambridge, Massachusetts, 1981, pp. 140-159.
2. Agha, G., Actors: A Model of Concurrent Computation in Distributed Systems, The MIT Press, Cambridge, Massachusetts. 1986.
  3. Amkraut, S., personal communication, January 8, 1987.
  4. Amkraut. S. Girard. M., Karl. G. "motion studies for a work in progress entitled 'Eurythmy'" in SIGGRAPH Video Review. Issue 21 (second item, time code 3:58 to 7:35). 1985. produced at the Computer Graphics Research Group. Ohio State University. Columbus, Ohio.
  5. Austin, H., "The Logo Primer," MIT AI Lab, Logo Working Paper 19, 1974.
  6. Braitenberg. V. Vehicles: Experiments in Synthetic Psychology. The MIT Press. Cambridge. Massachusetts. 1984.
  7. Burton, R., Bird Behavior, Alfred A. Knopf, Inc., 1985.
  8. Davis. J. R. Kay. A. Marion, A., unpublished research on behavioral simulation and animation. Atari Research. 1983.
  9. Girard, M., Maciejewski, A. A., "Computational Modeling for the Computer Animation of Legged Figures" in Computer Graphics V19 43. 1985. (proceedings of acm SIGGRAPH '85), pp. 263-270.
  10. Goldberg. A. Robson. D. SMALLTALK-80, The Language and the Implementation, Addison-Wesley Publishing Company, Reading, Massachusetts, 1983.
  11. Goldberg, A., Kay, A., SMALLTALK-72 Instruction Manual, Learning Research Group, Xerox Palo Alto Research Center, 1976.
  12. Hewitt, C. Atkinson. R. "Parallelism and Synchronization in Actor Systems," acm Symposium on Principles of Programming Languages 4. January 1977, Los Angeles, California.
  13. Kahn, K. M., Creation of Animation from Story Descriptions. MIT Artificial Intelligence Laboratory, Technical Report 540 (doctoral dissertation), August 1979.
  14. Kahn, K. M., Hewitt, C., Dynamic Graphics using Quasi Parallelism, May 1978, proceedings of ACM SIGGRAPH, 1978.
  15. Kleinrock, L., "Distributed Systems," in Communications of the AC,4t, V28#11, November 1985, pp.1200- 1213.
  16. Lipton, J., An Exaltation of Larks (or, The Venereal Game), Grossman Publishers, 1977. Reprinted by Penguin Books 1977.
  17. Maciejewski, A. A., Klein, C. A., "Obstacle Avoidance for Kinematically Redundant

Manipulators in Dynamically Varying Environments," to appear in International Journal of Robotic Research.

18. Magnenat-Thalmann. N., Thalmann, D., Computer Animation: Theory and Practice, Springer-Verlag, Tokyo, 1985.
19. Marion, A., "Artificially Motivated Objects," [installation piece], ACM SIGGRAPH art show, 1985.
20. Moon, D. A., "Object-oriented Programming with Flavors," in Proceedings of the First Annual Conference on Object-Oriented Programming Systems, Languages, and Applications, ACM, 1986.
21. Myers, R., Broadwell, P., Schaufler, R., "Plasm: Fish Sample," [installation piece], ACM SIGGRAPH art show, 1985.
22. Papert, S., "Teaching Children to be Mathematicians vs. Teaching Them About Mathematics," International Journal of Mathematical Education and Sciences, V3, pp. 249-262, 1972.
23. Partridge, B. L., "The Structure and Function of Fish Schools," Scientific American, June 1982, pp. 114-123.
24. Pitcher, T. J., Partridge, B. L., Wardle, C. S., "Blind Fish Can School," Science 194, #4268 (1976), p. 964.
25. Potts, W. K., "The Chorus-Line Hypothesis of Manoeuvre Coordination in Avian Flocks," letter in Nature, Vol. 309, May 24, 1984, pp. 344-345.
26. Pugh, J., "Actors--The Stage is Set," acm SIGPLAN Notices, V19 #3, March 1984, pp. 61-65.
27. Reeves, W., T., "Particle Systems-A Technique for Modeling a Class of Fuzzy Objects," acm Transactions on Graphics, V2 #2, April 1983. and reprinted in Computer Graphics. V17 #3, July 1983, (acm SIGGRAPH '83 Proceedings), pp. 359-376.
28. Reynolds, C. W., Computer Animation in the World of Actors and Scripts, SM thesis, MIT (the Architecture Machine Group), May 1978.
29. Reynolds. C. W., "Computer Animation with Scripts and Actors," Computer Graphics, V16 #3, July 1982, (acm SIGGRAPH '82 Proceedings), pp. 289-296.
30. Reynolds, C. W. "Description and Control of Time and Dynamics in Computer Animation" in the notes for the course on Advanced Computer Animation at acm SIGGRAPH '85, and reprinted, and reprinted for the notes of the same course in 1986.
31. Selous, E. Thought-transference (or what?) in Birds, Constable, London, 1931.
32. Scheffer, V. B., Spires of Form.' Glimpses of Evolution, Harcourt Brace Jovanovich, San Diego, 1983 (reprinted 1985 by Harvest/HBJ), p. 64.

33. Shaw, E., "Schooling in Fishes: Critique and Review" in Development and Evolution of Behavior. W. H. Freeman and Company. San Francisco, 1970, pp. 452480.
34. Shaw, E., "Fish in Schools," Natural History 84, no. 8 (1975), pp. 4046.
35. Sims, K., Locomotion of Jointed Figures Over Complex Terrain, SM thesis, MIT Media Lab, currently in preparation, April 1987.
36. Symbolics Graphics Division, S-Dynamics (user's manual). Symbolics Inc., November 1986.
37. Symbolics Graphics Division, S-Geometry (user's manual), Symbolics Inc., October 1986.
38. Pinker, S. (editor), Visual Cognition, The MIT Press, Cambridge, Massachusetts, 1985.
39. Thomas, K, Johnson, Ox, Disney Animation.' The Illusion of Life, Abbeville Press, New York, 1981, pp. 47- 69.
40. Wilhelms, J., "Toward Automatic Motion Control," IEEE Computer Graphics and Applications. V7 #4, April 1987, pp. 11-22.
41. Zeltser, D., "Toward an Integrated View of 3-D Computer Animation," The Visual Computer, VI #4, 1985. pp. 249-259.



## Footnotes

<sup>1</sup> Note: this is a reprint of the original publication in the proceeding of SIGGRAPH '87 (Computer Graphics 21(4), July 1987, edited by Maureen C. Stone, pages 25-34). It was produced by applying optical character recognition software to scanned images of the original hardcopy pages. The author wishes to thank Ken Cushman of SGI who generously donated his time and facilities to perform the OCR work, which allowed this old paper to get back online. Be forewarned: the OCR process introduces errors into the text. Most of these have been corrected through spell-checking and spotty proof-reading. Some errors may persist.

<sup>2</sup> Author's current address: Silicon Studio, 2011 North Shoreline Boulevard, MS 980, Mountain View, CA 94043, USA -- craig@studio.sgi.com -- <http://reality.sgi.com/employees/craig/>

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish requires a fee and/or specific permission.

(C)1987 ACM-0-89791-227-6/87/007/0025 \$00.75

\* In this paper flock refers generically to a group of objects that exhibit this general class of polarized, non colliding, aggregate motion. The term polarization is from zoology, meaning alignment of animal groups. English is rich with terms for groups of animals; for a charming and literate discussion of such words see An Exultation of Larks. [16]

\*\* This paper refers to these simulated bird-like, "bird-oid" objects generically as "boids" even when they represent other sorts of creatures such as schooling fish.

The following paper:

## **Evolution of Corridor Following Behavior in a Noisy World**

by  
**Craig W. Reynolds**

originally appeared in

### **From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior**

Edited by  
**Dave Cliff, Philip Husbands, Jean-Arcady Meyer, and Stewart W. Wilson**

A Bradford Book

Published by

# **MIT Press**

Cambridge, Massachusetts  
1994

(pages 402 to 410)

and is reprinted here by the kind permission of MIT Press

# Evolution of Corridor Following Behavior in a Noisy World

Craig W. Reynolds

Electronic Arts

1450 Fashion Island Boulevard

San Mateo, CA 94404, USA

telephone: 415-513-7442 / fax: 415-571-1893

creynolds@ea.com

cwr@red.com

## Abstract

Robust behavioral control programs for a simulated 2d vehicle can be constructed by artificial evolution. Corridor following serves here as an example of a behavior to be obtained through evolution. A controller's fitness is judged by its ability to steer its vehicle along a collision free path through a simple corridor environment. The controller's inputs are noisy range sensors and its output is a noisy steering mechanism. Evolution determines the quantity and placement of sensors. Noise in fitness tests discourages brittle strategies and leads to the evolution of robust, noise-tolerant controllers. Genetic Programming is used to model evolution, the controllers are represented as deterministic computer programs.

## 1 Introduction

Designing reactive controllers for autonomous agents can be a challenging task. For increasingly complex behavior, building controllers by hand becomes prohibitively difficult. As suggested in [Cliff 1993b], a promising alternative is to *evolve* the controllers, using their ability to perform the desired behavior as a measure of *fitness*. In these experiments, *corridor following* behavior is used as a simple test case, representative of the more complex behaviors to which this approach might eventually be applied.

Previous work [Reynolds 1993b] has shown that the Genetic Programming Paradigm [Koza 1992] can be used to automatically create control programs which enable a simple moving 2d vehicle to avoid collisions with obstacles by mapping sensory input (range data) into motor output (steering action). In those experiments all fitness tests were identical. As a result, the control programs evolved to use *brittle* strategies. Their success was like a "house of cards" which stands only until anything changes.

In the absence of variability in fitness testing, evolution will discover solutions that capitalize on the deterministic, precisely repeatable nature of the fitness tests. Evolved controllers will come to depend on utterly insignificant coincidental properties of the vehicle's sensors, its actuators, and their interaction with the environment.

The current work concerns an approach to avoiding this brittle behavior and seeks to evolve robust, general purpose control programs for the corridor following problem. Determinism is removed from fitness testing by injecting noise into the system. This noise will tend to "jiggle" coincidental

relationships between elements of the system and so tend to discourage evolution from capitalizing on them. A house of cards cannot be built on a shaky table.

The controllers evolved in this work are computer programs composed of basic arithmetic operations, conditionals, and a function to aim and read a nonlinear proximity sensor. The number and orientation of sensors are determined through evolution of control programs. On each simulation step control programs read their sensors and compute a steering angle. The vehicle always moves forward at a constant rate, so steering is its only means of avoiding collision.

These experiments have produced robust control programs capable of corridor following behavior in the presence of noise. Figure 1 shows some examples of successful behavior.

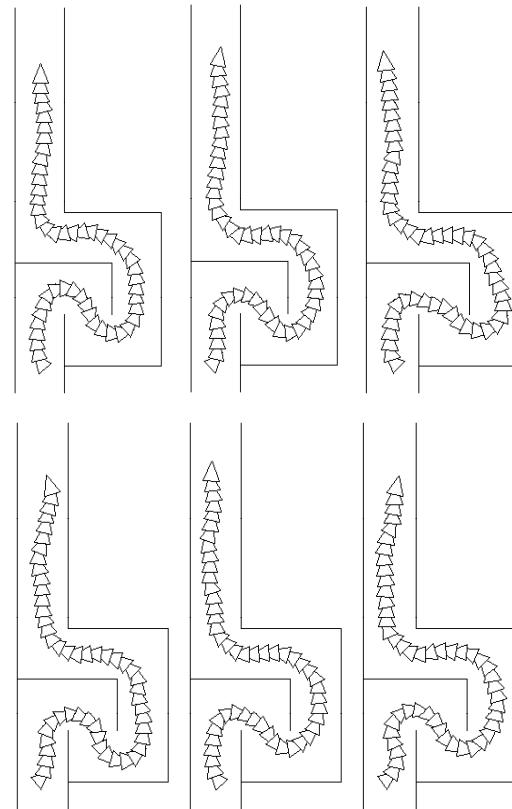


Figure 1: Several collision-free runs.

## 2 Related work

An early series of experiments [Reynolds 1993b] used Genetic Programming to create reactive controllers for a similar obstacle avoidance task. The fitness test employed a single, precisely repeatable simulation-based fitness test. This allowed evolution to take the easy path to discover a program which only solved this one specific control task. There was no incentive, no survival advantage, to find a controller that could generalize. As a result, the evolved controllers were “brittle” and could not solve similar but slightly different problems.

Subsequent experiments [Reynolds 1994a] attempted to use noise to promote robust solutions to the corridor following task, but were unsuccessful. The most significant difference between those experiments and these is the addition of a syntactic constraint to the sensor-reading function. In the earlier work, the control program could rotate its sensors relative to the vehicle. In the current work, the sensors are fixed to the vehicle during its run. Sensor orientation is still subject to change by the action of evolution.

Closely related to these experiments is the work of the Evolutionary Robotics Group at the University of Sussex. While using a different model of evolution (SAGA [Harvey 1992]) and a different model of controller architecture (dynamic, recurrent neural nets [Cliff 1993a], [Cliff 1993b], and [Harvey 1993]) they have investigated closely related problems in evolution of robotic controllers. They were the first to document the beneficial role of noise in the evolution of robust robotic controllers. The general approach used here, of evolving stimulus-response behavior based on simulated performance using simulated perception inside a closed simulated world, was originally inspired by [Cliff 1991a] and [Cliff 1991b]. The specific technique used here, of taking the “worst of four noisy trials” came directly from [Harvey 1993] and [Cliff 1993a].

The experiments reported here have much in common with some of Koza's work, particularly the evolution of behaviors such as “wall following” and “box pushing” as reported in [Koza 1992]. See also Simon Handley's GP robotics work [Handley 1994].

The evolution of robust controllers is related to the larger problem of generalization in evolutionary computation. This issue of generalization is an active area of research in many branches of evolutionary computation. In GP, see for example [Tackett 1994] on the evolution of generality in a classification problem, and [Kinnear 1993] on generalization in sorting.

Earlier work on obstacle avoidance behavior based on remote (distal) sensors (as opposed to touch sensors) for vehicles moving at “moderate speed” can be found in [Reynolds 1987], [Reynolds 1988], [Mataric 1993] and [Zapata 1993], among many others.

A classic reference on controllers and behaviors for this class of *vehicle* is [Braitenburg 1984] which is highly recommended.

## 3 Vehicle

The design of the simulated vehicle used in these experiments is kept intentionally vague and abstract. It

could equally well represent an animal as a wheeled or legged robotic vehicle. The intent is to gloss over the low level details of locomotion and to concentrate instead on the more abstract issues of “steering” and “path determination.” (Not “path planning,” since these reactive controllers neither plan nor learn.) In order to survive, the controllers need only steer along a clear pathway while avoiding contact with the danger region surrounding it. The skill involved is similar to that required by a squirrel running along a tree branch, or by an automobile's driver, negotiating through a narrow alley.

The control programs evolved by Genetic Programming represent the vehicle's “thought process” for a single simulation step. During fitness testing, the evolved program is run at each time step of the simulation. Using its sensors, the program inspects the environment surrounding the vehicle from its own point of view (that is, relative to the vehicle's local coordinate space), performs some arithmetic and logical processing, and decides how to steer (adjust the heading of) the vehicle. The value returned by the control program is interpreted as a steering angle. The vehicle then automatically moves forward by a constant amount (half of its body length). The fitness test continues until the vehicle takes the required number of steps (50), or until it collides with one of the obstacles. The raw fitness score for each corridor run is the number of steps taken divided by the maximum number of steps, producing a normalized score between 0 and 1, with 1 being best.

These vehicles have a fixed minimum turning radius because the maximum per-simulation-step turn is limited to  $\pm 0.05$  revolutions (18 degrees or 0.31 radians). This limitation implies a minimum turning circle which is somewhat larger than the width of the corridors of the obstacle course. As a result, the vehicle cannot spin in place, it cannot turn around in the corridor, and its only choice is to travel along the corridor.

Limiting turning radius produces a model of a vehicle moving at “moderate” speed. This qualitative description is intended to capture a relationship between the vehicle's momentum and its available turning acceleration. At “low” speed a vehicle has relatively little momentum, in a single time step it might be able to bring itself to a stop, or make an abrupt change of heading. At moderate speed momentum begins to dominate acceleration and changes of heading require many time steps. In this speed regime, maneuvers are less abrupt and paths tend to curve more gently, producing a motion more like running than crawling.

## 4 Corridor and Fitness Testing

The training environment used in these experiments was designed to test a control program's ability to follow a corridor, and to quickly reject those which are completely unsuited to the task. This approach allows the majority of the computational effort to go into testing higher fitness individuals. A fitness function based on this kind of simulation has the desirable property that execution time is roughly proportional to the individual's fitness.

Figure 2 shows the corridor and an series of increasingly successful runs. The vehicle is initialized in the center of

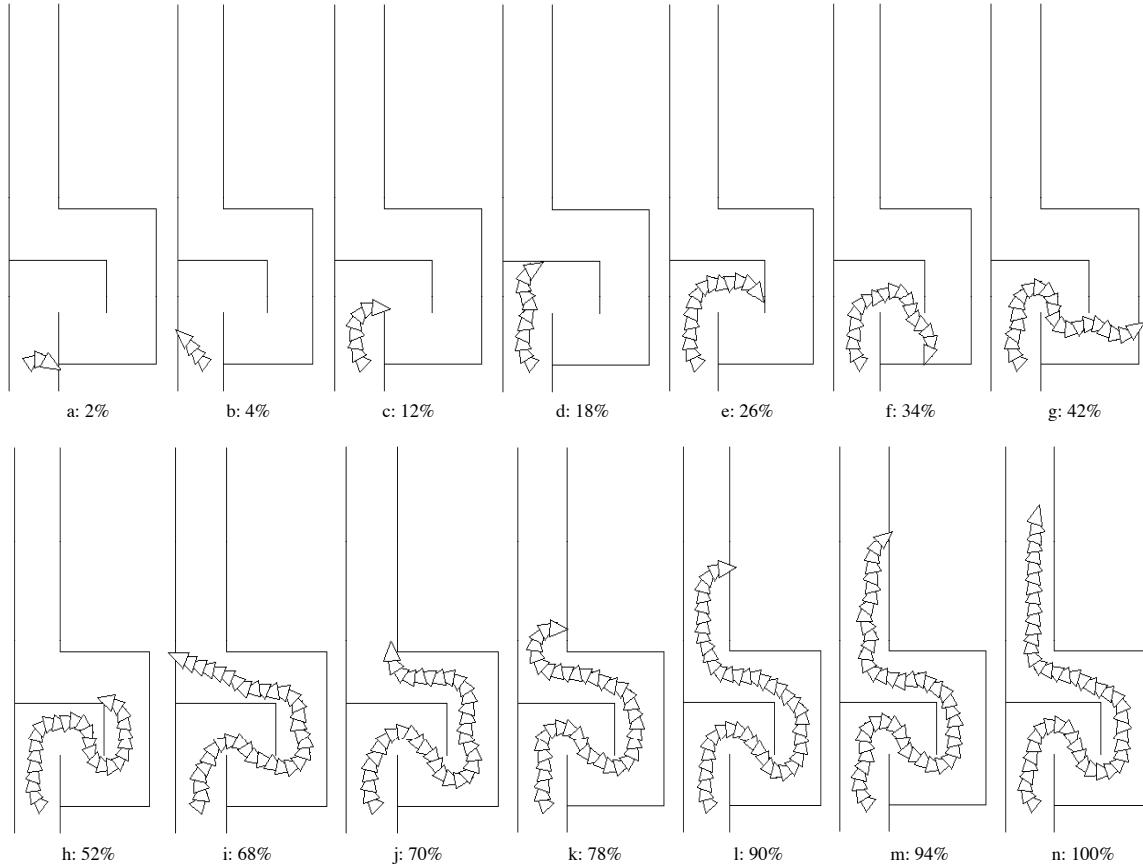
the corridor and pointing toward one wall or the other. Initial random headings range from 0.1 to 0.15 revolutions (36 to 54 degrees) off of the corridor's midline. If a controller turns continuously (see Figure 2(a)) or does not steer at all (see Figure 2(b)), it will run into a wall almost immediately. To survive more than a few steps the controller must develop the skill of sensing and turning away from a glancing collision so it can proceed down the corridor. To reach higher fitness levels the vehicle must be able to safely negotiate U-turns in both directions, right-angle turns in both directions, and long straight corridors.

In the early stages of evolution most of the controllers are quite inept and so incapable of following the corridor for long. During this stage it is important to notice and encourage progress, no matter how slight. In the initial population only a few controllers can take more than one or two collision-free steps so the scoring system must differentiate between degrees of ineptness, as illustrated in Figure 2. Later in the run some controllers will have progressed to the point where they can occasionally make a collision-free run through the corridor. At this stage the point of fitness testing begins to be reliability. One way to address this is to look at the controller's performance on a series of corridor runs. These considerations lead to the following scheme for each fitness trial: the controller is allowed to make a run through the corridor, if no collisions occur it is allowed to make another run, and so on up to a

maximum of 16 runs. This creates a selection pressure for controllers to be able, at least occasionally, to make it all the way through the corridor so as to have a chance at another run. This approach also serves to further focus effort on promising controllers: we don't even bother with a second trial unless the controller has "proven" itself worth on the first trial.

The scores of all runs of a trial (up until the first collision) are weighted and summed. The first collision-free run is worth 1/2, the second is 1/4, and so on. Figure 3 shows the cumulative fitness assigned to a controller based on the number of collision-free runs. This weighting scheme captures the idea of an open-ended reliability rating, but is probably not significant now that tournament selections is being used and all that matters is fitness rank. Monotone functions do not alter rank.

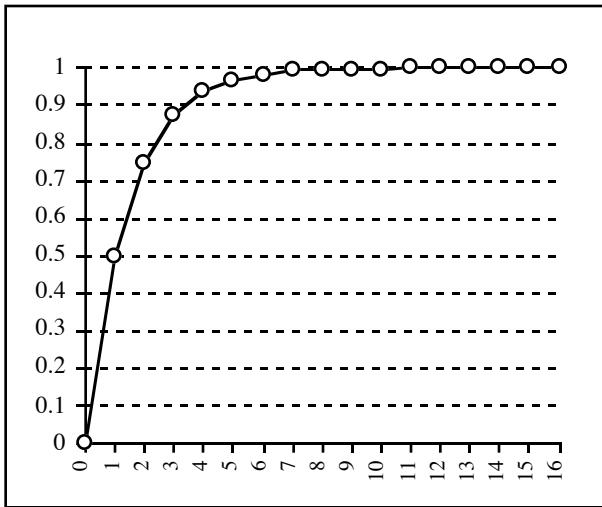
An obstacle course used in earlier experiments had a longer straight-away before and after the first turn. This arrangement seemed to promote premature convergence, particularly in small populations. Some individuals would discover how to make it to the first turn (and sometimes past it) but they would then so dominate the population that diversity would be lost and the population would never discover how to get around the second turn. In the subsequent experiments described here these problems were addressed in two ways. First, the long straight-aways were moved from the beginning of the course to the end. This



**Figure 2:** An assortment of runs, arranged in order of increasing scores. All but the last (*n*) end prematurely because of a collision.

forced the turning problem to be addressed sooner, and weeded out the non-turners sooner. Second, the entire obstacle course was mirrored at random about the axis of the first corridor. This procedure ensured that evolving controllers could turn equally well in both directions and so prevented convergence towards right-turners.

Because the training corridor is laid out on a grid, it has some built-in regularities. The passageway has uniform width throughout. All walls meet at right angle corners. Presumably evolution will construct controllers which depend on these regularities. That is, in the absence of counter-examples, it will assume that all corridors are the same width. A different kind of training environment may be required to evolve controllers capable of following irregular corridors.



**Figure 3:** Fitness versus number of sequential collision-free runs through the corridor.

## 5 Genetic Programming Considerations

The technique used to evolve computer programs in this work is known as Genetic Programming and was invented by John Koza. The best reference on this technique and its application is [Koza 1992]. While often used as a generational technique, it is also possible to combine Genetic Programming with Steady State Genetic Algorithms [Syswerda 1989], [Syswerda 1991] as described in [Reynolds 1993a].

A very brief description of Steady State Genetic Programming (SSGP) follows. First a population of random programs is created and fitness tested. (In these experiments the population consisted of 2000 to 10000 programs.) Thereafter SSGP proceeds by: (1) choosing two *parent* programs from the population, (2) creating a new *offspring* program from them, (3) fitness testing the new program as described in the previous section, (4) choosing a program to remove from the population to make room, and (5) adding the new program into the population. The parent programs are chosen in a way that favors the more fit while not totally ignoring the less fit, thus balancing *exploration* of the whole gene pool with *exploitation* of the champions. In these experiments this choosing is done using *tournament selection*

with  $k=7$ , that is: seven individuals are chosen from the population at random, and the most fit of those seven is selected as the winner. The recombination of two parents to form a new offspring is accomplished by the Genetic Programming crossover operator. GP crossover is a little like “random cut and paste” but is done in a way that guarantees the new program’s syntactic correctness.

Selecting a program to remove from the population could be done by using inverse tournament selection: taking the least fit of seven randomly chosen programs. However the greedy nature of SSGP, combined with the noisy fitness testing used in these experiments, leads to the possibility of a mediocre-but-lucky program receiving an undeservedly high fitness and going on to dominate the population. To combat this possibility a modified removal policy was used in these experiments: half the time inverse tournament selection was used, the other half of the time an individual was selected for removal at random (without regard to fitness). Hence all programs, even the best one, had a certain small but non-zero possibility of being removed at each SSGP step. This tended to ensure that the population could not stagnate with a collection of mediocre-but-lucky programs, winning strategies were continually being retested.

Because steady state genetic computation proceeds individual by individual, there is no demarcation of generations. However it is often convenient to describe the progress or length of a SSGP run in terms of “generation equivalents:” processing as many new individuals as there are programs in the population.

Applying Genetic Programming to a problem requires specifying several parameters such as the genetic population size and the fitness function (both described above). In addition we must specify the *functions* and *terminals* that define the language in which evolved program will be expressed. In these experiments the terminals were simply random floating point numbers. The function set consisted of:

```
+  
-  
*  
%  
abs  
iflte  
look-for-obstacle
```

The first three are the standard Common Lisp functions for addition, subtraction, and multiplication. The % and iflte are standard GP functions [Koza 1992]: % is “protected divide” (returns zero when denominator is zero), and iflte is an arithmetic conditional (if A is less than or equal to B, then C else D). abs is the standard Common Lisp function for absolute value.

The look-for-obstacle function is specific to the obstacle avoidance problem. It takes a single numeric argument which represents an angle relative to the current vehicle heading. Angles are specified in units of *revolutions*, a normalized angle measure: 1 revolution equals 360 degrees or  $2\pi$  radians. look-for-obstacle points its sensor in the given direction and returns a measure of obstacle proximity. In this implementation, the range is computed by performing a 2d ray-tracing operation on the obstacles.

The Genetic Programming substrate used here, and the application-specific functions for the simulation, were originally developed on Symbolics Lisp Machines. For the current series of experiments, the software was ported to Macintosh Common Lisp (version 2.0p2) and was run on Macintosh Quadra 950 workstations. In this implementation, an average fitness test (composed of up to 64 corridor runs) in run z6 took about 65 seconds to perform.

## 6 Results

Three runs will be discussed in this section. One uses fixed sensor positions, the other two allow the sensor placement to evolve. The genetic population size and upper limit on evolved program size also differ between the runs:

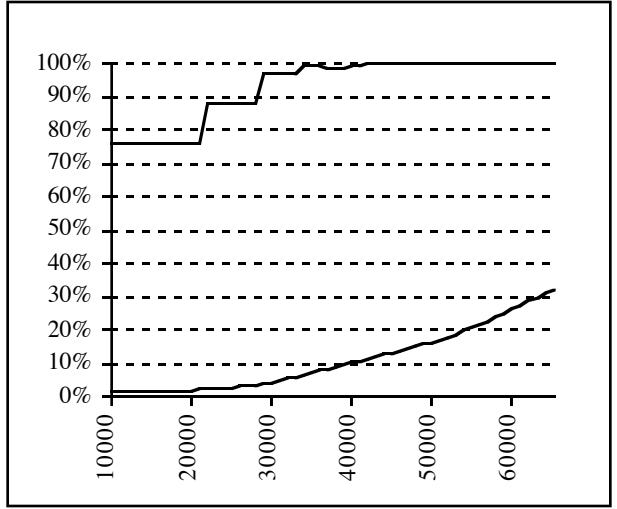
run:	population:	sensors:	program size limit:
z4	10000	fixed	50
z6	2000	variable	35
z7	10000	variable	35

Run "z4" used a fixed-sensor model. The vehicle was defined to have exactly nine sensors, spaced 1/16 of a revolution (22.5 degrees) apart across the vehicle's front. (If the vehicle's heading is "north," this would correspond to placing sensors at compass points: W, WNW, NW, NNW, N, NNE, NE, ENE, and E.) In Genetic Programming terms, this was implemented by adding a set of nine sensor-reading functions to the GP function set. For comparison with the other two runs described below, this fixed sensor model could also have been implemented by restricting the argument to `look-for-obstacle` to be one of the nine numeric values (3/4, 13/16, 7/8, 15/16, 0, 1/16, 1/8, 3/16, 1/4). In this model, the evolved control programs would call various sensor-reading functions, perform some arithmetic and logical processing, and return a steering angle. Evolution could not alter the sensor placement, it could only select which sensors to use, and how to combine their readings.

This formulation of the corridor following turned out to be very easy. So much so that the GP system almost solved it by random search. The initial generation of a GP run corresponds to random search over the space of programs (subject to the limitation of size, function set, and terminal set which are parameters to GP). During the initial generation of 10000 random programs for run "z4," the best program had a fitness of 76.5%. This value represents better performance than evolved in any of the previous variations of this problem as reported in [Reynolds 1994a], even though this program was found by random search before the beneficial effects of evolution were applied to the population. During the third generation equivalent (around individual number 35000) of run "z4" it began to attain best-of-population fitness scores above 99%.

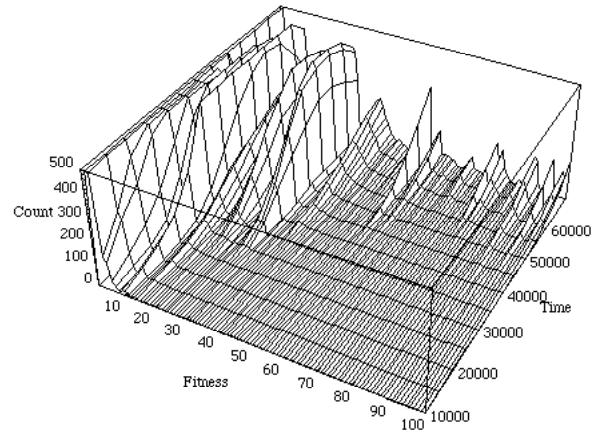
Figure 5 summarizes the z4 population's fitness distribution over time. It shows a time series of discrete fitness histograms portrayed as a bilinear surface. The population's progress towards increasing fitness over time can be seen. Dominating this landscape are fin-shaped features probably caused by two artifacts of the fitness function. The corridor is a series of turns, most runs end at a turn, so scores tend to

be clumped at certain values. Then as described in Figure 3, this distribution is repeated at half scale, at quarter scale, and so on for additional runs.



**Figure 4:** Fitness plots for run z4.  
(best-of-population and population average)

Koza has described what he calls the "lens effect," [Koza 1994] the way in which every representation alters the difficulty of a given problem. Koza explores the lens effect by looking at the distribution of fitness values found during random search through the specified program space. In those terms, it appears that the fixed sensor representation of the corridor following problem is a "lens" that makes the problem quite easy to solve. An analysis of fitness distribution in the initial generation of these three runs can be found in [Reynolds 1994b]. By that metric it appears that the fixed sensor representation is easier than the variable sensor representation, which is in turn easier than the original "roving" sensor representation of [Reynolds 1994a].



**Figure 5:** Fitness histograms of run z4 over time.

While not the best-of-run, one high scoring (99.82%) and compact (size 15) controller which caught the author's eye was this elegant three sensor design that appeared during the

fourth generation-equivalent (individual number 46515, see Figure 6). It is shown here hand-simplified down to size 7:

```
(% (- (obs-3/16) (obs-13/16))
  (- (obs-15/16) 2.5))
```

This program works by comparing the proximity of obstacles (walls) at relative headings of  $\pm 3/16$  (the expression  $(- (obs-3/16) (obs-13/16))$  returns a value between -1 and 1 which indicates relative lateral proximity), then scales that value down by dividing it by a number ranging between -2.5 and -1.5 which is related to the obstacle proximity at a heading of  $-1/16$ . This appears to increase the rate of turn as to the amount of free space “ahead” decreases.

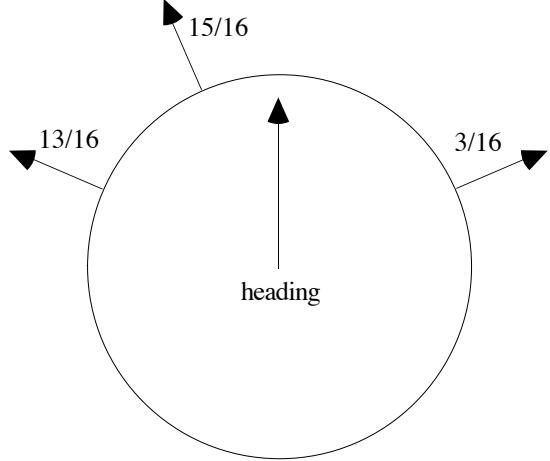
The controllers evolved in run z4 were both more and less complicated than the individual discussed above. Many were variations on the same theme: divide the relative lateral proximity by a sensor-dependent scalar, most varied only in the form of modulation. At the end of the run, after about 6.5 generation equivalents, there were two individuals who tested at 100% fitness. After simplification by hand, they were of size 5 and 33:

```
(% (- (obs-3/16)
      (obs-13/16))
  -1.995)

(% (- (obs-3/16)
      (obs-13/16))
  (- (abs (iflte (obs-15/16)
    0.0
    (+ (obs-3/16)
        (obs-7/8)
        (obs-15/16))
    (* (* (iflte (obs-1/8)
      (obs-0)
      (obs-1/8)
      0)
    (iflte (obs-3/16)
      (- (obs-1/8)
          (obs-3/16))
      (obs-7/8)
      0)
    (obs-13/16)
    0.5)
    (obs-0)
    (obs-1/8)
    0.2)))
  2.0))
```

Based on those results, run z6 was designed to test whether evolution could correctly determine quantity and placement of sensors in addition to determining how to combine the various sensor readings into a correct steering angle. The approach was to restrict the argument to `look-for-obstacle` to be a number. That is, each call to `look-for-obstacle` has a constant numeric argument representing a certain angle. Any number is allowed, but more complicated expressions, particularly those involving additional calls to `look-for-obstacle` are not allowed. When a new program is created (randomly or by crossover) it is checked to ensure that arguments to `look-for-obstacle` are each numeric constants. When non-constant arguments are found, they are replaced by a random number between 0 and 1. As a result, each call such as `(look-for-obstacle 0.17)` corresponds to a sensor fixed at an arbitrary angular offset

from the vehicle’s heading. In this way, sensor *morphology* can evolve in parallel with the processing needed to map sensor data into a steering signal.



**Figure 6:** Sensor morphology of individual z4-46515.

Sensors pointing in certain directions (such as directly ahead) are presumably more relevant to corridor following than sensors pointing in other directions (such as directly behind). Therefore certain sensors, represented as code fragments, will tend to increase the fitness of programs in which they appear. This is known as the *constructional fitness* of the code fragment [Altenberg 1994]. Programs containing the more useful sensor-defining fragments will have a survival advantage, and so those fragments will tend to proliferate in the population. In this way the evolutionary process decides which sensor directions are most useful for solving the corridor following problem.

On the tenth generation equivalent of run z6, the best-of-population individual has a fitness of 98.03%. This run developed a rather elaborate *hitch-hiking intron*<sup>1</sup> that has spread throughout the population, so while the best-of-population individual is almost as large as it can be (size 34), most of its genetic material is irrelevant. After removing the irrelevant *intron wrapper* this size 10 program remains:

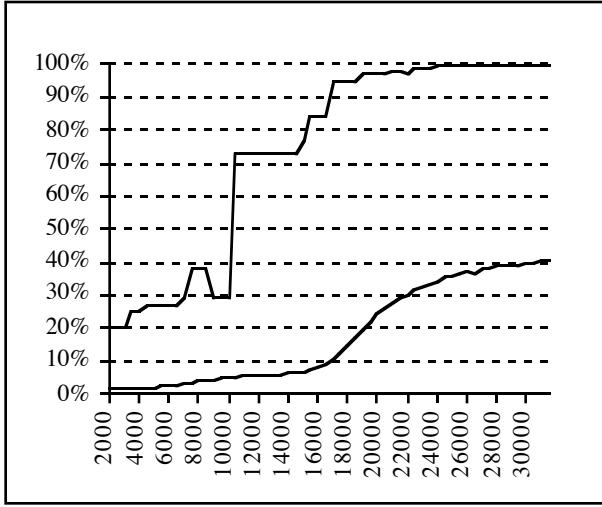
```
(* (look-for-obstacle 0.13)
  (look-for-obstacle 2.0)
  (- (look-for-obstacle 0.81)
    (look-for-obstacle 0.17)))
```

A  
B  
C  
D

While based on a slightly different approach, this program shares certain features of the program from run “z4” analyzed above. Specifically, the subtraction (on lines C and D) is computing the same sort of lateral proximity measure by taking the difference between a left pointing sensor

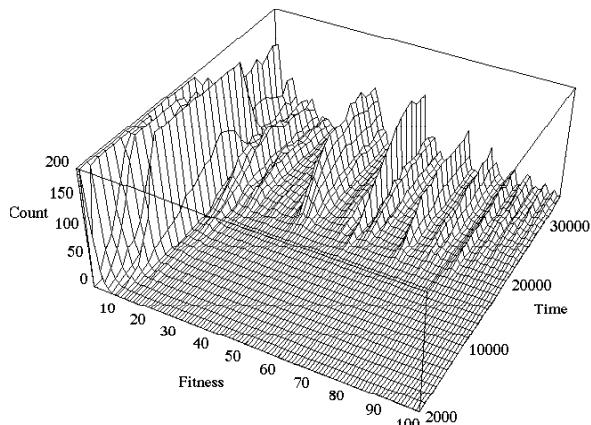
<sup>1</sup> By analogy with its usage in biological genetics, the term *intron* is used in GP to refer to code that is included in an evolved program but which does not contribute to the program’s action or result. An intron is part of the genotype but not of the phenotype. When an intron becomes structurally associated with a highly fit code fragment, the intron can hitch-hike and so proliferate through the population. In the case of run z6, essentially all programs had a large no-op conditional wrapped around the active code.

and a right pointing sensor. In fact the left sensor used is essentially identical to the left sensor used in the "z4" program. The pair of sensors used here are not quite symmetrically placed: +0.17 revolutions on the right and -0.19 on the left. The lateral proximity value is then scaled down by multiplication by two numbers each of which are between 0 and 1. The sensor specified on line B points directly ahead (2.0 revolutions is the same angle as 0 revolutions) and the sensor on line A points 0.13 revolutions (47 degrees) to the right. Hence the lateral proximity signal is reduced as the obstacle-free path, either ahead or to the right, increases.



**Figure 7:** Fitness plots for run z6.  
(best-of-population and population average)

Essentially all of the high-fitness individuals in run z6 use this same basic framework, within which there are a few variations of the sensor placement. For example the best-of-population individual at 21600 is identical except that it uses the values 0, 2.0, 0.82, and 0.17 (on lines A, B, C, and D, respectively). Effectively it has become a three sensor design, since both forward-pointing sensors are at the same angle. It is almost exactly symmetrical, which seems appropriate for its task.



**Figure 8:** Fitness histograms of run z6 over time.

Run z6 found what might be regarded as a competent controller, but one that is clearly less than perfect. To attain 100% fitness in these experiments, a controller must be able to execute 64 (16 times 4) consecutive perfect runs through the corridor, for a total of 3200 correct steps. The best-of-generation fitness in z6 was always below 100%. This implies that the failure rate for these controllers is at least once in 64 runs. To put this in perspective, imagine a human driver who averages one collision every 64 automobile trips! While there is clearly room for improvement in the controllers from run z6, there is also evidence that the population became converged and would be unable to progress any further.

Another run was attempted to duplicate and hopefully surpass the results of z6. Run "z7" was identical to z6 except that the population was increased by a factor of five to 10000. Unfortunately the run appeared to "top out" running out of steam perhaps because the population converged too rapidly on an inferior strategy. See Figure 9. After about 8.5 generation equivalents, the best of population was stuck at 40% and population average was leveling off at 15%. This outcome demonstrates that evolutionary computation is a stochastic technique, and not all runs succeed. There are suggestions that most reliable way to use computational resources is to use a series of many smaller runs in place of a single large run.

## 7 Conclusions

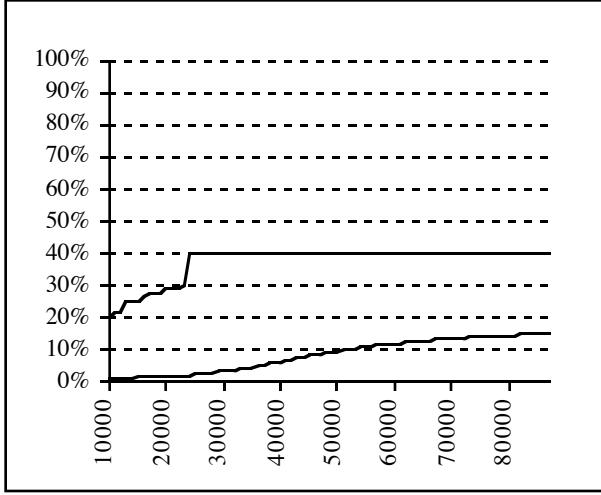
These results indicate that behavioral control programs can be evolved using Genetic Programming and a noisy simulation-based fitness test. The solutions discovered by this process are simple and robust. It appears that noise in fitness testing discourages strategies that are brittle, opportunistic, or overly complicated. The only solutions that can survive noisy fitness testing are compact and robust.

## 8 Future Work

As corridor following behavior developed in these experiments it became clear that *reliability* is a difficult property to measure. While competent behavior clearly arose, it is hard to measure just how robust and how reliable these controllers are. If we wanted to measure the reliability of a human automobile driver, what would be the criterion? Would we count the number of collisions over a long period of time, say a year? How would we differentiate between all of the individuals who had zero collisions? Is a year a long enough test period, and if not, how long are we willing to wait for an answer? All of these same issue come up in trying to rate the reliability of the corridor following controllers evolved here. A topic of future study will be to determine the most efficient techniques for testing reliability.

The competent behavior evolved in these experiments represented a breakthrough after a long series of unsuccessful experiments in applying very similar GP techniques to the same problem. The fact that a seemingly tiny change in the GP representation (restricting the argument of look-for-obstacle to be a constant) made a huge difference in the difficulty of the problem is a tantalizing result which deserves further study. What does this say about the problem

domain? What does it say about Genetic Programming? How can we characterize this representational difficulty, and how can we avoid it in the future? Some first thoughts are explored in [Reynolds 1994b].



**Figure 9:** Fitness plots for run z7  
(best-of-population and average-of-population).

## Acknowledgments

This work was supported by Electronic Arts. The author wishes to thank his supervisor Steve Crane and Vice President of Technology Luc Barthelet, for allowing blue-sky research to coexist with product development. Thanks to John Koza and James Rice for advice. Thanks to “assistant GP guy” Emmanuel Berriet for providing extra CPU cycles. Special thanks to my wife Lisa and to our first child Eric, whose gestation corresponded with this paper’s.

## References

- Altenberg, L. (1994) The Evolution of Evolvability in Genetic Programming, in *Advances in Genetic Programming*, K. E. Kinnear, Jr., Ed. Cambridge, MA: MIT Press.
- Braitenburg, V. (1984) *Vehicles*, MIT press, Cambridge, Massachusetts.
- Cliff, D. (1991a) Computational Neuroethology: A Provisional Manifesto, in *From Animals To Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior* (SAB90), Meyer and Wilson editors, MIT Press, Cambridge, Massachusetts.
- Cliff, D. (1991b) The Computational Hoverfly; a Study in Computational Neuroethology, in *From Animals To Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior* (SAB90), Meyer and Wilson editors, MIT Press, Cambridge, Massachusetts.
- Cliff, D. (1993a) P. Husbands, and I. Harvey Evolving Visually Guided Robots, in *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior* (SAB92), Meyer, Roitblat and Wilson editors, MIT Press, Cambridge, Massachusetts, pages 374-383.
- Cliff, D. (1993b) I. Harvey, and P. Husbands, Explorations in Evolutionary Robotics, *Adaptive Behavior* 2(1), pages 73-110.
- Collins, R. J. (1992) *Studies in Artificial Evolution*, Ph.D. thesis, University of California at Los Angeles..
- Handley, S. (1994) The Automatic Generation of Plans for a Mobile Robot via Genetic Programming with Automatically defined Functions, in *Advances in Genetic Programming*, K. E. Kinnear, Jr., Ed. Cambridge, MA: MIT Press.
- Harvey, I. (1992) Species Adaptation Genetic Algorithms: The Basis for a Continuing SAGA, in *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, Varela and Bourgine editors, MIT Press/Bradford Books, pages 346-354.
- Harvey, I. (1993) P. Husbands, and D. Cliff, Issues in Evolutionary Robotics, in *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior* (SAB92), Meyer, Roitblat and Wilson editors, MIT Press, Cambridge, Massachusetts, pages 364-373.
- Kinnear, K. E. Jr. (1993) Generality and Difficulty in Genetic Programming: Evolving a Sort, in *Proceedings of the Fifth International Conference on Genetic Algorithms*, S. Forrest, Ed San Mateo, CA: Morgan Kaufmann, pages 287-294.
- Koza, J. R. (1992) *Genetic Programming: on the Programming of Computers by Means of Natural Selection*, ISBN 0-262-11170-5, MIT Press, Cambridge, Massachusetts.
- Koza, J. R. (1994) *Genetic Programming II: Scalable Automatic Programming by Means of Automatically Defined Functions*, MIT Press, Cambridge, Massachusetts (in press).
- Mataric, M. J. (1993) Designing Emergent Behaviors: From Local Interactions to Collective Intelligence, in *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior* (SAB92), Meyer, Roitblat and Wilson editors, MIT Press, Cambridge, Massachusetts, pages 432-441.
- Ngo, J. T. (1993) and J. Marks, Spacetime Constraints Revisited, Proceedings of SIGGRAPH 93 (Anaheim, California, August 1-6, 1993), in *Computer Graphics Proceedings*, Annual Conference Series, 1993, ACM SIGGRAPH, New York, pages 343-350.
- Reynolds, C. W. (1987) Flocks, Herds, and Schools: A Distributed Behavioral Model, in *Computer Graphics*, 21(4) (SIGGRAPH '87 Conference Proceedings) pages 25-34.
- Reynolds, C. W. (1988) Not Bumping Into Things, in the notes for the SIGGRAPH '88 course *Developments in Physically-Based Modeling*, pages G1-G13, published by ACM-SIGGRAPH.
- Reynolds, C. W. (1993) An Evolved, Vision-Based Behavioral Model of Coordinated Group Motion, in *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior* (SAB92), Meyer, Roitblat and Wilson editors, MIT Press, Cambridge, Massachusetts, pages 384-392.
- Reynolds, C. W. (1993) An Evolved, Vision-Based Model of Obstacle Avoidance Behavior, in *Artificial Life III*, Santa Fe Institute Studies in the Sciences of Complexity,

Proceedings Volume XVI, C. Langton, Ed. Redwood City, CA: Addison-Wesley.

Reynolds, C. W. (1994a) Evolution of Obstacle Avoidance Behavior: Using Noise to Promote Robust Solutions, in *Advances in Genetic Programming*, K. E. Kinnear, Jr., Ed. Cambridge, MA: MIT Press.

Reynolds, C. W. (1994b) The Difficulty of Roving Eyes, in *Proceedings of the IEEE World Congress on Computational Intelligence*, IEEE (in press).

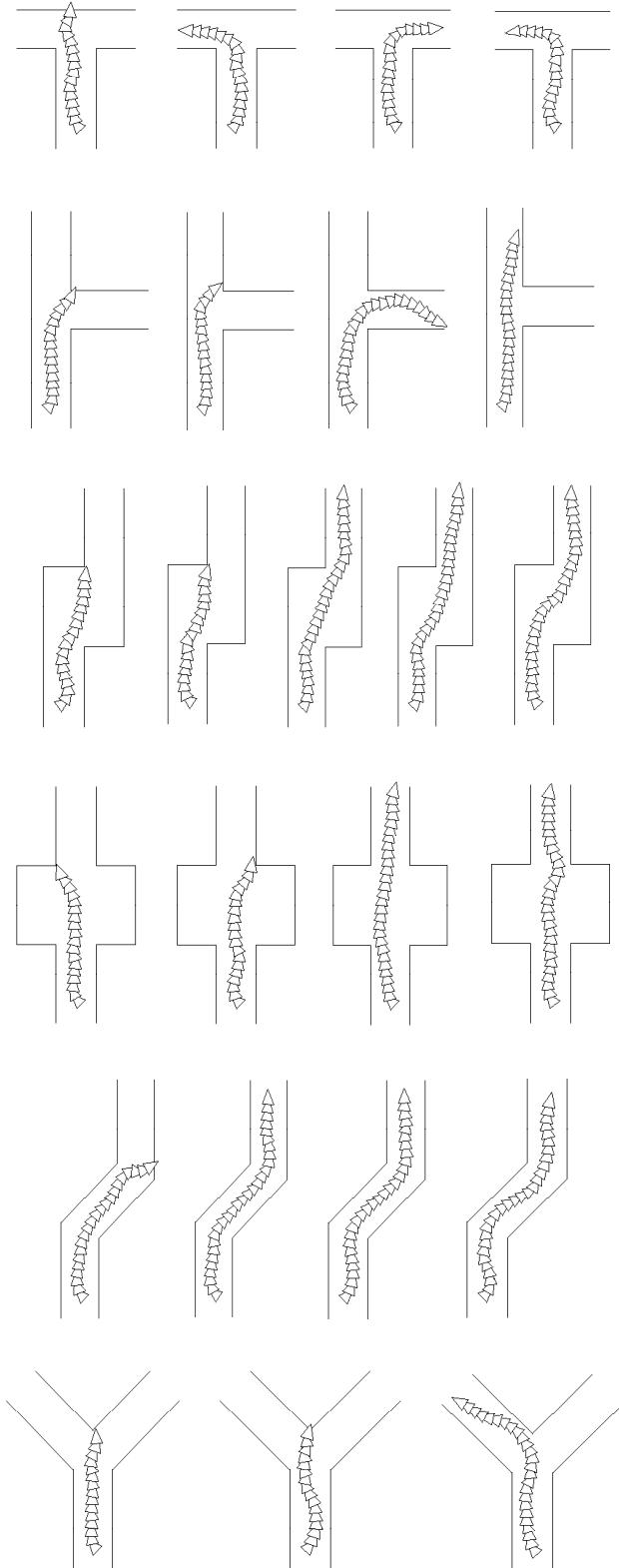
Syswerda, G. (1989) Uniform Crossover in Genetic Algorithms, in *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2-9, Morgan Kaufmann Publishers.

Syswerda, G. (1991) A Study of Reproduction in Generational and Steady-State Genetic Algorithms, in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed. San Mateo, CA: Morgan Kaufmann, pages 94-101.

Tackett, W. A. (1994) and A. Carmi, The Donut Problem: Scalability, Generalization, and Breeding Policy in the Genetic Programming, in *Advances in Genetic Programming*, K. E. Kinnear, Jr., Ed. Cambridge, MA: MIT Press.

von Neumann, J. (1987) Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components, in *Papers of John von Neumann on Computing and Computer Theory*, W. Aspray and A. Burks Eds. Cambridge, MA: MIT Press.

Zapata, R. (1993) P. Lépinay, C. Novales, and P. Deplanques, Reactive Behaviors of Fast Mobile Robots in Unstructured Environments: Sensor-based Control and Neural Networks, in *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior* (SAB92), Meyer, Roitblat and Wilson editors, MIT Press, Cambridge, Massachusetts, pages 108-115.



**Figure 10:** some tests of generalization. Controllers evolved in runs z4 and z6 have been placed in corridors containing novel features. Corridors used in their evolution had only right angles, were of constant width, and contained no branch points.



The following paper:

**Competition, Coevolution and the Game of Tag**

by  
Craig W. Reynolds

originally appeared in

**Artificial Life IV:**

**Proceedings of the Fourth International Workshop  
on the Synthesis and Simulation of Living Systems**

Edited by  
Rodney A. Brooks and Pattie Maes

A Bradford Book

Published by

**MIT Press**

Cambridge, Massachusetts  
1994

(pages 59 to 69)

and is reprinted here by the kind permission of MIT Press

# Competition, Coevolution and the Game of Tag

**Craig W. Reynolds**

Electronic Arts

1450 Fashion Island Boulevard

San Mateo, CA 94404 USA

telephone: 415-513-7442, fax: 415-571-1893

creynolds@ea.com

cwr@red.com

## Abstract

*Tag* is a children's game based on symmetrical pursuit and evasion. In the experiments described here, control programs for mobile agents (simulated vehicles) are evolved based on their skill at the game of tag. A player's fitness is determined by how well it performs when placed in competition with several opponents chosen randomly from the coevolving population of players. In the beginning, the quality of play is very poor. Then slightly better strategies begin to exploit the weaknesses of others. Through evolution, guided by competitive fitness, increasingly better strategies emerge over time.

## 1. Introduction

Many of us remember playing the *game of tag* as children. Tag is played by two or more, one of whom is designated as *it*. The *it* player chases the others, who all try to escape. Tag is a simple contest of pursuit and evasion. These activities are common in the natural world, most predator-prey interactions involve pursuit and evasion. Tag also includes an aspect of role-reversal, both pursuit and evasion skills are required. *It*'s goal is to catch up with another player, to get close enough to reach out and touch the other player. At this point the pursuer shouts "Tag! You're *it*!" and the former evader becomes the new pursuer.

The game of tag serves here as a toy example, to study the use of competitive fitness in the evolution of agent behavior. Tag is intended as a simple model of behavior based on control of locomotion direction, or *steering*. By evolving a vehicular steering controller for the game of tag we have a test case to learn about evolving controllers for related, but more complex tasks.

We seek to automatically discover a controller through evolution based solely on competition between controllers. This approach stands in contrast to evolving controllers by pitting them against a static, predetermined expert strategy.

**Table 1:**

competitive architecture	matches per generation of n	opponents per individual	reference
new versus all	$(n^2-n)/2$	n-1	[Koza 1992]
new versus several	$nk$	k	this paper
single elimination tournament tree	$n-1$	$\log_2 n$	[Angeline 1993]
new versus previous best	$n$	1	[Sims 1994]
new versus new	$n/2$	1	[Smith 1994]

The use of *competitive fitness* has the significant advantage of avoiding the paradoxical need for an expert controller as a prerequisite for evolving an expert controller. Another advantage of competitive fitness is that since each fitness test is unique, there is no danger of overfitting a static expert.

## 2. Related Work

This research was originally inspired by Pete Angeline's elegant work on coevolution of players for the game of Tic Tac Toe, using competitive fitness [Angeline 1993]. That fitness could be tested by competition alone, even in the absence of an expert player, is a key insight in applying evolutionary techniques to the discovery of complex goal-directed behaviors. The work reported here extends Angeline's paradigm from games of pure strategy to those involving geometric motion. The competition in [Angeline 1993] was in the form of a single elimination tournament tree. The players in each new generation were paired up in competition. The winners of each pairing went on to a second round, and so on for several rounds until only one champion remained. A player's fitness was determined by the number of matches won before a defeat. See Table 1 for a comparison of several competitive architectures described in this section.

A similar approach has recently been used to coevolve strategies for the game of Othello [Smith 1994]. A genetic algorithm was used with competitive fitness to determine time-varying weightings for the static evaluator used in an alpha-beta search of the Othello game tree. In this work, two new players are placed in competition with each other, the score of their game determines the fitness of both.

Another paper on competitive evolution of behavior appears elsewhere in this volume [Sims 1994]. It describes experiments where the morphology and behavior of artificial creatures evolve through competition for control of an inanimate object. The creatures each try to get closest to the object, and if possible to surround it. Each new creature

is placed in competition with the best creature from the previous generation.

John Koza evolved pursuer-evader systems closely related to those reported here (see pages 423-428 of [Koza 1992]). But in that work the pursuers and evaders existed in separate populations. Their fitness was determined by comparison with a pre-existing optimal player. In the same book, Koza first discusses coevolution in Genetic Programming (pages 429-437) in the context of a discrete strategy game.

In the work reported here, the vehicle model, and the noise-tolerant Steady-State Genetic Programming system was taken from [Reynolds 1994a] and [Reynolds 1994c]. The vehicle model draws heavily from [Braitenberg 1984] and is equivalent to the *turtle* of the LOGO programming language.

Competitive fitness and coevolution were first explored in evolutionary computation in the context of the Iterated Prisoner's Dilemma in [Axelrod 1984], [Axelrod 1989], [Miller 1989], [Lindgren 1992] and [Lindgren 1994]. A restricted form of competitive fitness was used in [Hillis 1992] to drive the evolution of sorting networks: antagonistic test cases were coevolved to force generality. Coevolution and competitive fitness are fundamental to a wide class of ecological simulations studied in Artificial Life such as: ECHO [Holland 1992], Tierra [Ray 1992], BioLand [Werner 1993], PolyWorld [Yeager 1994], LEE [Menczer 1994], and the work reported in [D'haeseleer 1994]. Chapter six of [Kauffman 1993] is an authoritative analysis of "The Dynamics of Coevolving Systems."

The reader should understand that this work is *not* about optimal control theory, despite the fact that frequent mention is made of optimal strategies, and that this type of pursuer-evader system has a long history in the optimal control literature [Isaacs 1965]. The work reported here benefits from (but does not depend upon) the ability to compare evolved behavior with optimal behavior. It should be noted that this is practical only because of the extremely simple vehicle model used in these studies. Once slightly more complicated models are used, optimal behavior becomes a much more complicated issue, see for example [Merz 1971].

### 3. Experimental Design

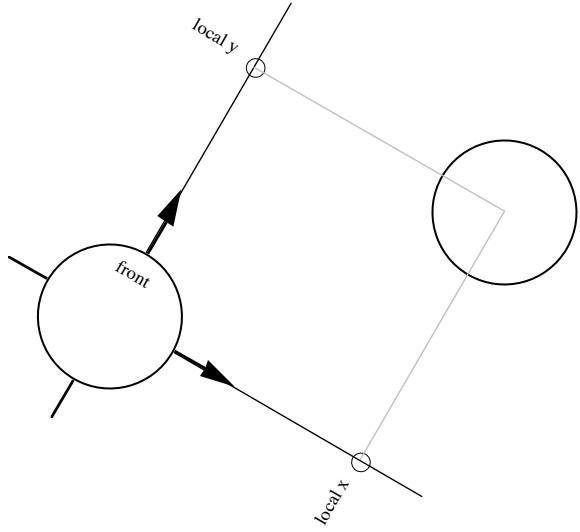
In this work, Genetic Programming is used to evolve control programs for simulated vehicles, based on their ability to play the game of tag. Each new player's fitness is determined through competition with existing players. The game of tag provides a framework wherein the relative fitness of two players is judged through direct competition.

The vehicles are abstract autonomous agents, moving at constant speed on a two dimensional surface. Each vehicle's evolved control program is executed once per simulation step. Its job is to inspect the environment and to compute a steering angle for the vehicle. Angles are specified in units of *revolutions*, a normalized angle measure: 1 revolution equals 360 degrees.

For each player, at each simulation step: (1) its control program is run to determine a steering angle, (2) the

vehicle's heading is altered by this angle, (3) the vehicle is moved a fixed distance along its new heading, and (4) tags are detected and handled. The forward step length is typically 125% longer for *it*. The per-step steering angle is unconstrained: these vehicle can instantaneously turn by any amount. This is an abstract *kinematic* vehicle model, as opposed to a *physically realistic* model. In this work there is no simulation of force, mass, acceleration, or momentum. In contrast, a physically realistic model would serve to limit the turning radius.

In these experiments there are always two players in a tag game. The playing field is featureless. Each vehicle's environment consists of just one object: the opponent vehicle. The opponent's current heading is not relevant. In the absence of momentum, there is no correlation between the vehicle's current heading and its position in the next time step. For a given controller the entire state of the world consists of: a flag indicating who is *it*, and the relative position of the opponent's vehicle. The position information is presented to the control program in terms of x and y coordinates of the opponent, expressed in the local coordinate system of the controller's own vehicle. A vehicle's local coordinate system is defined with the positive y axis aligned with the vehicle's heading and the positive x axis extending to the vehicle's right. See Figure 1.



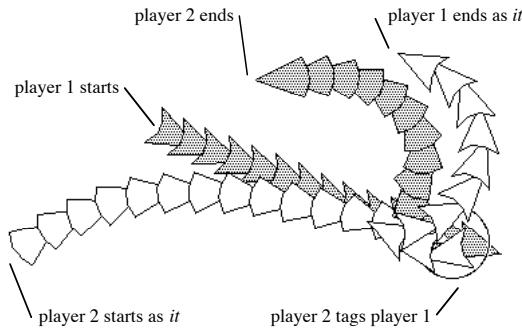
**Figure 1:** the input to a vehicle's controller is the position of the opponent vehicle, expressed in terms of local x and y coordinates.

While the traditional game of tag has no formal method of scoring, the intuitive goal is to avoid being *it*. Therefore, fitness is defined here to be the portion of time (simulation steps) spent not being *it*. Note that this is a normalized, bigger-is-better fitness metric: zero is worst, one is best. To determine a player's fitness it competes with other players.

To compare two players, a series of four games is played. Before each game the players are given random initial headings and are randomly positioned within a starting box measuring about 3.5 vehicle-body-lengths on a side. The two players alternate starting as *it* for each game of the

series. If *it* tags the opponent, by getting to within one vehicle length, the *it* and non-*it* roles are reversed. Each game consisted of 25 simulation steps. A player's score for a game is the number-of-non-*it*-steps divided by 25. See Figure 2.

To determine a player's fitness, it is pitted against 6 other players. These opponents are chosen by uniform random selection from the existing population. Scores from these 24 games (a series of 4 games against each of 6 opponents) are averaged together to obtain the final fitness value. A value of 50% indicates that the player has an ability comparable to the population average. Fitness values above 50% indicate increasingly better players. Because opponents are randomly selected, and because initial conditions are randomized, these fitness values have significant variance and provide only a rough estimate of a player's actual fitness.



**Figure 2:** a game of tag between player 1: an evolved controller from run G, and player 2: the optimal strategy. As in all such diagrams in this paper, the pursuer (*it*) is shown in white, the evader is gray, and the site of a tag is indicated by a circle. Player 1 is shown here with a concave back edge and player 2 has a convex back edge. In this game player 1 begins by fleeing in a naive direction. Player 2 chases down and tags player 1. Player 2 then escapes because player 1 is too slow turning around. Player 1 was not *it* for 14 of the 25 steps so earned a score of 56%. Player 2 was not *it* for 11 steps so scored 44%.

In the discussion of experimental results below, reference will be made to an *optimal player*. Because of the unrealistically simple vehicle model used here, particularly the absence of momentum, the optimal strategy is quite straightforward. For the pursuer, the optimal strategy is to turn toward the evader (so the evader lies on the pursuer's positive y axis). Similarly the optimal strategy for the

evader is to turn directly away from the pursuer (so that the pursuer lies on the negative y axis).

The “four quadrant arctangent” implements this optimal strategy. However the fitness criterion in these experiments is performance in tag competition. The optimality of a given controller is irrelevant to its fitness, all that matters is its performance in tag competition with the coevolving population. As will be seen below, controllers which are clearly non-optimal can still produce behavior which is asymptotically close to optimal performance.

#### 4. Genetic Programming and Tag

The technique used to evolve computer programs in this work is known as Genetic Programming (“GP”), invented by John Koza. The best reference on this technique and its application is [Koza 1992]. In its original formulation, GP operated on a population of individuals generation by generation. Alternatively, GP can be combined with Steady State Genetic Algorithms [Syswerda 1991] as described in [Reynolds 1993a].

A very brief description of Steady State Genetic Programming (SSGP) follows. First a population of random programs is created and tested for fitness. In these experiments the population consisted of 1000 to 5000 programs. Thereafter SSGP proceeds by: (1) choosing two *parent* programs from the population, (2) creating a new *offspring* program from them, (3) testing the fitness of the new program as described in the previous section, (4) choosing a program to remove from the population to make room, and (5) adding the new program into the population. The parent programs are chosen in a way that favors the more fit while not totally ignoring the less fit, thus balancing *exploration* of the whole gene pool with *exploitation* of the current champions. In these experiments this choosing is done using *tournament selection*: seven individuals are chosen from the population at random, the most fit of those is selected as the winner. The recombination of two parents to form a new offspring is accomplished by the Genetic Programming *crossover* operator. GP crossover is a bit like “random cut and paste” done on balanced parenthetical expressions to guarantee the new program’s syntactic correctness. After crossover, a program might be *mutated* by substituting one function for another, one terminal for another, or by crossover with a new random program fragment.

Selecting a program to remove from the population could be done by using inverse tournament selection: removing

**Table 2**

function	usage	description	source
+	(+ a b)	a plus b	Common Lisp
-	(- a b)	a minus b	Common Lisp
*	(* a b)	a times b	Common Lisp
%	(% a b)	if b=0 then 1 else a divided by b	[Koza 1992]
min	(min a b)	if a<b then a else b	Common Lisp
max	(max a b)	if a>b then a else b	Common Lisp
abs	(abs a)	absolute value of a	Common Lisp
iflte	(iflte a b c d)	if a ≤ b then c else d	[Koza 1992]
if-it	(if-it a b)	if this player is <i>it</i> then a else b	this paper

the least fit of seven randomly chosen programs. However the greedy nature of SSGP, combined with the variance of fitness measures used in these experiments, leads to the possibility of a mediocre-but-lucky program receiving an undeservedly high fitness and going on to dominate the population. To combat this possibility, a modified removal policy was used in these experiments: half the time inverse tournament selection was used, the other half of the time an individual was selected for removal at random (without regard to fitness). All programs, even the best one, has a certain small but non-zero probability of being removed at each SSGP step. This approach reduces the possibility that the population could stagnate with a collection of mediocre-but-lucky programs. To survive, winning strategies must continue to perform well.

Another issue is that competitive fitness values are measured relative to the population at a certain point in time. These fitness value becomes less and less relevant as time passes. The hybrid SSGP removal policy ensures that the population is continually being recycled.

Because steady state genetic computation proceeds individual by individual, there is no demarcation of generations. However it is often convenient to describe the progress or length of a SSGP run in terms of "generation equivalents:" processing as many new individuals as there are programs in the population.

In order to evolve tag-playing control programs with Genetic Programming, we first choose a set of functions and terminals to form the language in which they will be expressed. Certain choices are dictated by the application itself, and some are intended to provide the logical and arithmetic "glue" with which GP will construct control programs. Three functions used here are specific to the underlying application, they provide the three parameters for the control programs. `local-x` and `local-y` are functions of zero arguments which return the x and y coordinates of the opponent player. The `if-it` macro provides conditional execution of its two subexpressions depending on whether this player is currently *it*. In addition to these functions, several others are provided to help form control programs. The choice is rather arbitrary, it was influenced by the author's domain knowledge and by preliminary attempts to write a tag-playing control program by hand. This function set provides for arithmetic, thresholding, sign manipulation, and conditional computation, see Table 2.

In addition to (`local-x`) and (`local-y`) mentioned above, the terminal set includes `:random-0-1`, an ephemeral random constant. After a new program is formed, any occurrences of this terminal are replaced by a pseudo-random floating point number between 0.0 and 1.0.

**Table 3:**

run name	population size	it speed ratio	program size limit	mutation used	segregated branches	opponent selection
A	5000	1.00	50	no	yes	uniform
B	2000	1.25	50	no	yes	uniform
C	1000	1.25	50	yes	yes	uniform
D	1000	1.25	50	yes	yes	tournament
E	1000	1.25	50	yes	no	uniform
F	1000	1.25	50	no	no	uniform
G	1000	1.25	100	no	no	uniform

Evolved programs are subject to a size limitation which is measured in term of the total number of functions and/or terminals. When crossover produces a program whose size exceeds this limit, the *hoist* genetic operator [Kinnear 1994] is used to find a smaller (but hopefully still fit) subexpression. In these experiments the size limit was either 50 or 100.

These experiments used the `if-it` conditional in two different ways. In the early runs, a *syntactic constraint* was used to ensure that evolved programs always contained exactly one occurrence of `if-it` and it was always at the top (outermost) level of the program. That is, all programs in those runs had this structure:

```
(if-it <pursuer-branch> <evader-branch> )
```

The GP crossover operation was modified to exchange code fragments only within branches of the same type. With this constrained structure and crossover, the effect is to create two distinct gene pools, one for pursuit behavior and one for evasion behavior. Thus the frequency of code fragments are allowed to evolve differently in each population. This approach is similar to the Automatically Defined Functions described in [Koza 1992].

In later experiments this syntactic constraint was removed and the `if-it` conditional was treated as just another non-terminal. As a result there could be any number of `if-it` forms in an evolved program. There was no segregation of the gene pool, code fragments could cross over from the pursuit branch to the evasion branch, and vice-versa.

The Genetic Programming substrate used here was originally developed by the author on Symbolics Lisp Machines and subsequently ported to Macintosh Common Lisp (version 2.0p2). These experiments were run on Macintosh Quadra 950 workstations. In this implementation a fitness test consisting of 24 tag games takes 7 to 12 seconds to run, depending on program size.

## 5. Results

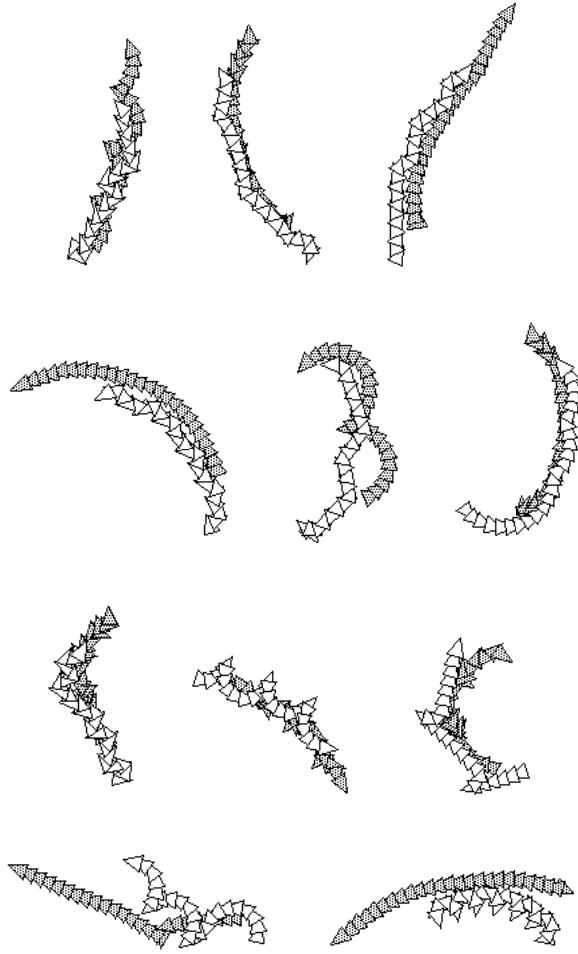
Seven evolution runs were performed using the basic experimental design described above. The runs varied in terms of population, the relative speed of the two players, the program size limit, whether mutation was used, whether the *it/not-it* branches were segregated, and the method used for selecting opponents. See Table 3. Runs A, C and G will be discussed in more detail in the following sections.

### 5.1 Run A

Run A had a population of 5000 individuals. Both players moved at the same speed. This puts *it* at a fundamental

disadvantage: as long as the other player moves away, the best *it* can do is to follow at a constant distance, unable to close the gap.

Most of the initial, random programs in run A used ineffective strategies. A typical behavior was to bumble around aimlessly. Some of the early programs effectively had no steering behavior. These control program always returned zero (or some very small value) and the vehicle would simply travel in a straight line. While this is not a very effective pursuit strategy (for *it*), it is occasionally a good evasion strategy. If *it* happened to start off positioned behind you, running straight ahead is essentially optimal evasion. Since initial position and orientation is random, this non-turning evasion behavior would be effective between one-half and one-quarter of the time, depending on how good *its* pursuit strategy is. As a result, early in the run these non-turners began to proliferate through the population of evasion strategies.



**Figure 3:** pursuit and evasion from run A in which both players moved at the same speed.

At this early stage, the most effective pursuit strategies appear to have been looping (constant steering angle) and “stumblers” that seemed to move erratically, but managed to creep slowly towards their target. The looping behavior was successful because it allowed the pursuer to cover more

ground, increasing its chance of blindly tagging the evader. The stumblers, while quite inefficient when compared to optimal pursuit behavior, were able to survive by *preying* on incompetent, aimless evaders. The large number of bad strategies in the population provided fertile ground for innovation: new strategies, even inefficient ones, could prosper by exploiting the weaknesses of others.

Later in run A, an improved evasion strategy appeared. The gist of it was: if the pursuer is behind you, go straight ahead, otherwise turn randomly. The effect is that the evader would twitch randomly for a few steps, then it would find itself pointing away from the pursuer and head straight away. Not only is this a fairly robust strategy, but it was easily implemented (hence easily evolved) in the programming language used in these experiments:

```
(if-it <pursuer-branch> (max 0 (local-y)))
```

This code fragment, and many variations of it, appeared in the population. Most of the variations were larger expressions which were functionally equivalent. Once this simple but effective evader appeared, the pursuers were in trouble. Because of the advantage mentioned earlier, that the evader need only move away from the pursuer in order to win, the pursuers could only achieve mediocre performance. In general the quality of pursuit in run A improved. The pursuers got to be very good at picking off the easy targets, the inefficient evaders. But this only served to improve the gene pool of the evaders. Soon only fairly good evaders remained, and they could easily escape from the best of the pursuers.

This sounds like it might have been the end of the story, but something interesting happened. In the interval between individuals 57000 and 76000, the pursuers developed several different techniques for following their targets, each with a characteristic and visually distinct pattern of motion (see Figure 3). Each of these patterns of motion had to have a certain measure of success to survive, but none could overcome the built-in advantage of the evaders in run A. There is a temptation (unsupported by any data) to see these distinct patterns of motion as something akin to evolutionary stable strategies (or perhaps species in environmental niches). It is also possible that these classes of motion arose solely as artifacts of the function set used in the GP representation.

One pursuit strategy seen in this run (and others) used a competent but inefficient “three phase” technique. These players would always do one of three things: turn left, turn right, or turn around. It is inefficient because turning left or turning right is a good idea only if the quarry is off to the side. If your opponent is directly ahead, a zig-zag path will slow your rate of progress. The code below is the pursuit branch of individual number 179120, unedited except to add comments:

```
(iflte 0.029628 ; if y > 0 (quarry ahead)
  (local-y) ;
  (iflte 0.212021 ; if x > 0.2 (quarry on right)
    (Local-x) ;
    0.862946 ; turn 49 degrees to right
    0.134561) ; turn 48 degrees to left
    0.541760) ; turn 195 degrees (about face)
```

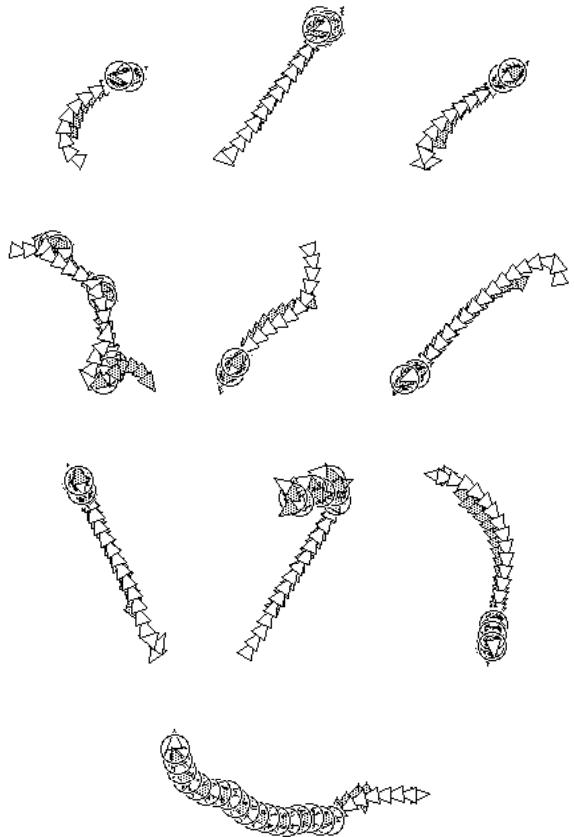
As evolution in run A proceeded, the evaders developed an unexpected behavior. For the majority of them life was easy, they could usually escape the pursuers. (Those who could not were quickly killed off.) One might guess that the evader population would settle down into using a simple, efficient strategy. Instead it seems that they had an overabundance of genetic material and were determined to use it! They adopted a complicated, elaborate strategy. Because both players move at the same speed, the pursuer is not a threat until it moves close to the evader. This defines a certain “threat radius” around the evader. If the pursuer is outside this radius it doesn’t matter what the evader does, it could just as well bumble around aimlessly. Once the pursuer crosses the threat radius the evader must snap to attention and run away efficiently. These elaborate evaders are very large programs and hard to analyze in detail. One possible explanation is that an effective evasion strategy was discovered, but its implementation depended on the values of local-x and local-y being limited to certain small values. The result was an evader that would flee very efficiently when closely pursued, but otherwise would appear to randomly flip around. Subjectively this behavior looked for all the world like the evader was “saving its strength” -- moving slowly until the pursuer got close enough to be a threat. Yet this could not have been what actually happened, since there was no modeling of expended energy, nor did the fitness function reward conservation of energy.

A note about the selection of games shown in, for example, Figure 3. The choice of images for this paper was a biased, subjective process. An unedited random sampling of images from the run might have been more representative, but would have been much less visually interesting. These “photogenic” images were chosen because of their clarity, simplicity, and visual appeal. Several were typical, a few were unusual. Collecting these images was a process somewhat like nature photography. Lots of pictures were taken, many were discarded, and a few were selected that captured the spirit of the behavior.

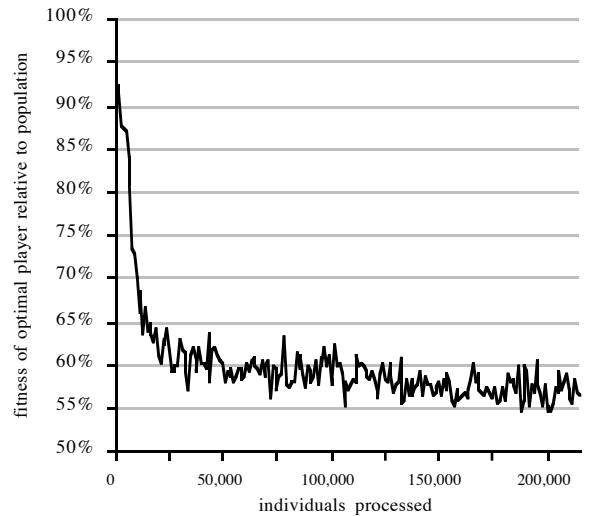
## 5.2 Run C

Evolution of pursuit in run A was stymied by the evader’s advantage. Run C sought to even out the competition by giving *it* a 25% speed advantage. Now a good pursuer could close in on and tag a good evader. This served to encourage efficient pursuit, which in turn encouraged improved evasion.

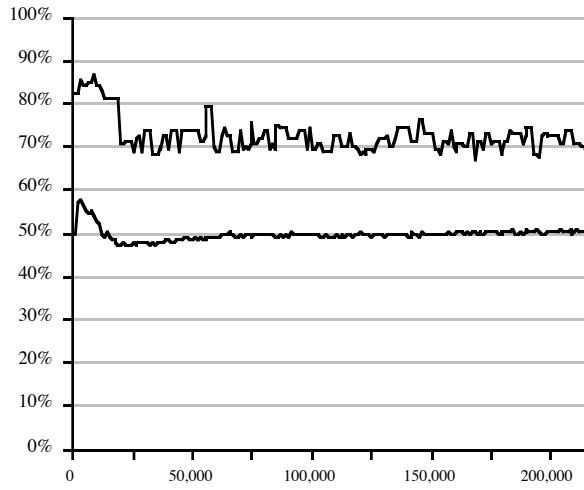
Run C used a population of 1000. Mutation was added in an attempt to help prevent the loss of diversity observed in earlier runs. Run C was evolved for 215 generation equivalents. The quality of pursuit and evasion improved steadily, becoming skillful and well matched. Many games consisted of a chase featuring near-optimal pursuit and evasion followed by a series of rapid tags with *it* alternating back and forth each step. After each tag the new *it* would just turn around, take one big step, and tag the other player. This was reminiscent of the games seen when two optimal players were pitted against each other.



**Figure 4:** tag games from run C. Most consist of a successful chase followed by a series of tags.



**Figure 5:** plot of the fitness of the optimal player placed in competition with the evolving population of run C. Since fitness is measured relative to the existing population, the decline in fitness shown here is indicative of the increasing average fitness of the population. Initially the optimal player avoided being *it* 93% of the time. After 22000 individuals (22 generation equivalents), the optimal player is *it* about 60% of the time. Thereafter the value drifts slowly down into the range between 55 and 60%.



**Figure 6:** plots of best-of-population fitness (top) and average-of-population fitness (bottom) for 215 generations of run C.

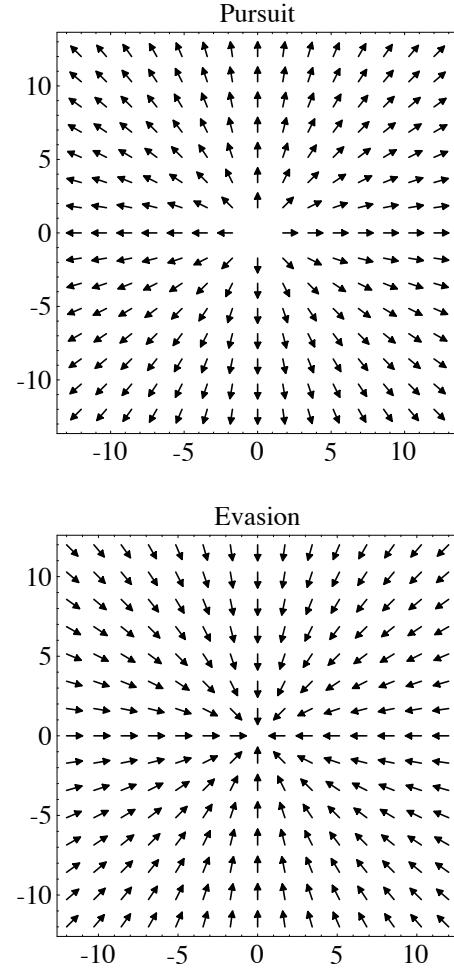
After 215415 individuals were processed in run C there were four individuals with the same best fitness value. One of those was selected for further analysis. It was compared to the optimal player in a series of 100 games. The evolved player got a score of 49.3% indicating that was holding its own against the optimal player, being *not-it* just less than half of the time. The evolved program's size was 48. A few simplifications were made by hand to produce this size 38 version:

```
(if-it (max (% (local-y)
  (+ (min (local-y)
    (min (abs (local-y))
      (iflte 0.25235322
        (local-y)
        (* 0.09556236
          (local-x))
        (local-x))))
      (local-y)))
  0.25235322)
  (iflte (local-y)
    0.0055459295
    (iflte (local-y)
      (* (local-y)
        (min (* (local-x) (local-x))
          0.47677472))
      (* 0.051421385 (local-x))
      (* 0.107852586 (local-x)))
    0.47677472))
```

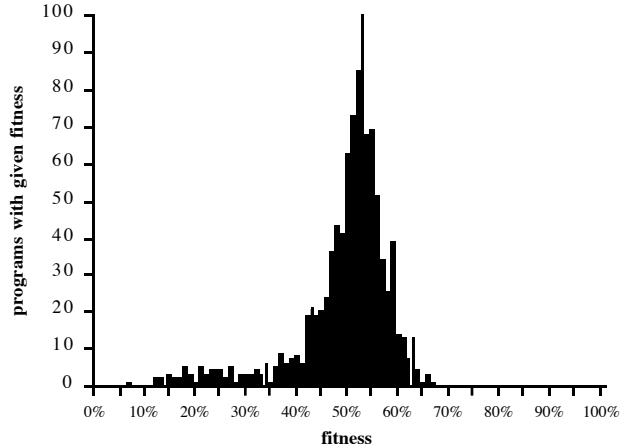
To visualize the overall behavior of a player's evolved control program, the two inputs can be used to define a perceptual field embedded in the local space surrounding the vehicle. Mapping the control program over a mesh of points in this field yields an array of steering angles. This map describes the stimulus-response relationship implemented by the control program as it maps an opponent's position into a steering angle.

Figure 7 shows plots of this relationship for the optimal player. Each arrow in this diagram indicates the player's steering angle response to an opponent in the vicinity of the arrow's tail. Figure 9 summarizes the behavioral mapping

for an evolved player from run C. This useful visualization tool has long been applied to the study of reactive robotic control system in the work of Ronald Arkin, see for example [Arkin 1987].

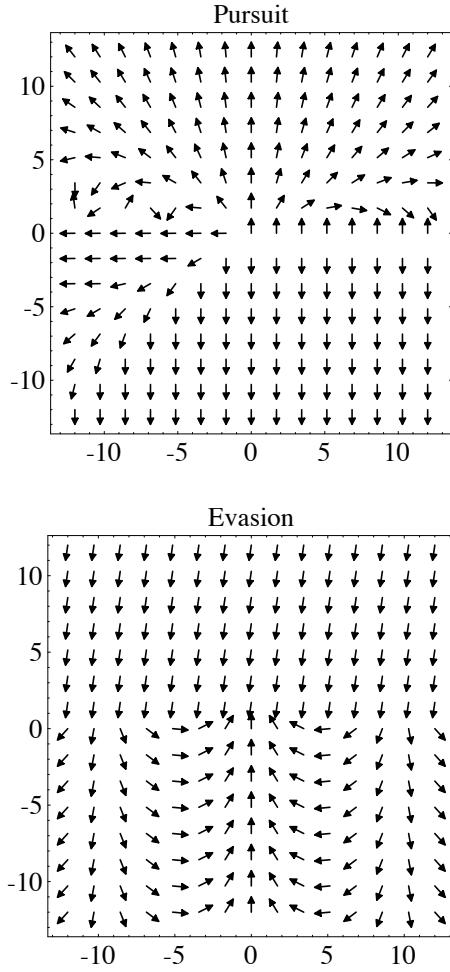


**Figure 7:** Plots of steering angle versus opponent position for the optimal player. These stimulus-response diagrams summarize the player's behavior.



**Figure 8:** histogram of fitness distribution in population of run C after 215 generation equivalents.

An examination of Figure 9 suggests that a "foveal region" has developed in the perceptual field of both pursuer and evader. In the pursuer the forward facing quadrant has a near-optimal structure. In the evader the backward facing quadrant (particularly in the region close to the player and along its negative y axis) has a near-optimal structure.



**Figure 9:** Plots of steering angle versus opponent position for the best-of-population player from run C after 215 generation equivalents

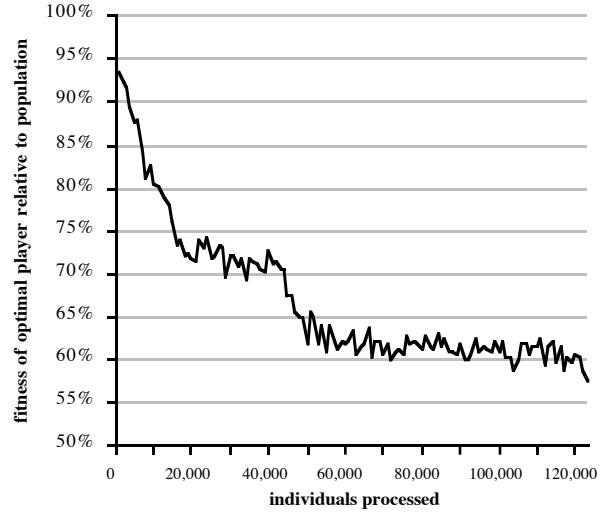
Figure 9 also indicates that in the "anti-foveal" direction the response is simply to turn halfway around. The pursuer will turn around if the evader is behind it, and the evader will turn around if the pursuer is in front of it. The general strategy appears to be to turn "quickly" toward (or away from) the opponent, to place them in your foveal region, wherein your response is close to optimal. The behavior is less organized in the direction perpendicular to the foveal axis, but the effect seems to be the same: after a step or two the vehicle is heading in the right direction.

Since the vast majority of the player's time is spent in this nearly optimal foveal region, there is correspondingly less selection pressure to "optimize" the behavior in other regions. This is particularly true given the relative small number of steps executed in these simulations. For

example, sometimes an evader escapes because its behavior is more efficient, but sometimes it escapes because the 25 steps of the simulation have expired. There are very few tag games where non-optimal behavior in the off-foveal regions makes a measurable difference in fitness.

### 5.3 Run G

Run G did not segregate the pursuer and evader code and used a larger limit on program size. One of the changes seemed to make the problem harder to solve. The syntactic constraint used in earlier runs is a user-provided "hint" that there are two distinct cases to solve. This may lower the difficulty of the problem. Further comparisons of more runs would provide more reliable information about difficulty. Figure 10 shows the performance of the G population against the optimal player. Comparing this to Figure 5 shows that run C generally did better sooner.



**Figure 10:** plot of the fitness of the optimal player placed in competition with the evolving population of run G over 123 generation equivalents.

Individual 113520 of run G was the best of population when it was created. The size 98 program is listed below. Its stimulus-response map is shown in Figure 11. This individual has many strange behavioral traits, for example pursuit behavior has a reasonable two phase strategy for opponents up to 5 units ahead but is very inept for opponents further away. The evasion behavior is strongly asymmetrical.

```
(% (% (if-it (abs (local-x)) (iflte (iflte (local-x)
0.57168305 (local-x) (+ (iflte (local-y) (iflte (local-y)
(if-it (local-x) (abs (local-x))) (iflte 0.40530929
0.26004231 (abs (local-x)) (local-y)) (if-it 0.40530929
0.57168305)) (min (abs (local-x)) (+ (local-x) (local-
x)) (local-x)) (local-x))) 0.57168305 (local-x) (+
(iflte (local-y) (iflte (local-y) (if-it (local-x)
(local-x)) (iflte 0.40530929 (local-x) (abs (iflte
(local-x) 0.37254661 0.32281655 (local-x))) (local-x))
(if-it 0.40530929 (abs (local-x)))) (min 0.1637349 (iflte
(local-x) (local-y) (abs (iflte (abs (local-x)) (max (if-
it (local-y) (abs 0.53183758)) (local-x) 0.32281655
```

```
(local-x))) 0.53183758)) (local-x)) (local-x)))) (+  

(local-x) (local-x)) (iflte (- (abs 0.53183758) (if-it  

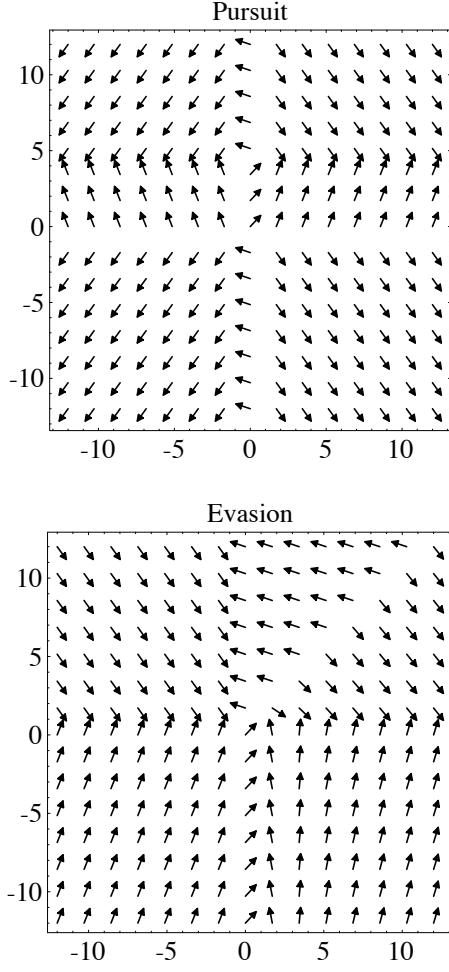
(% 0.57168305 (local-y)) (- 0.1637349 (local-y))))  

0.40530929 (abs 0.53183758) 0.83426005))
```

## 6. Conclusions

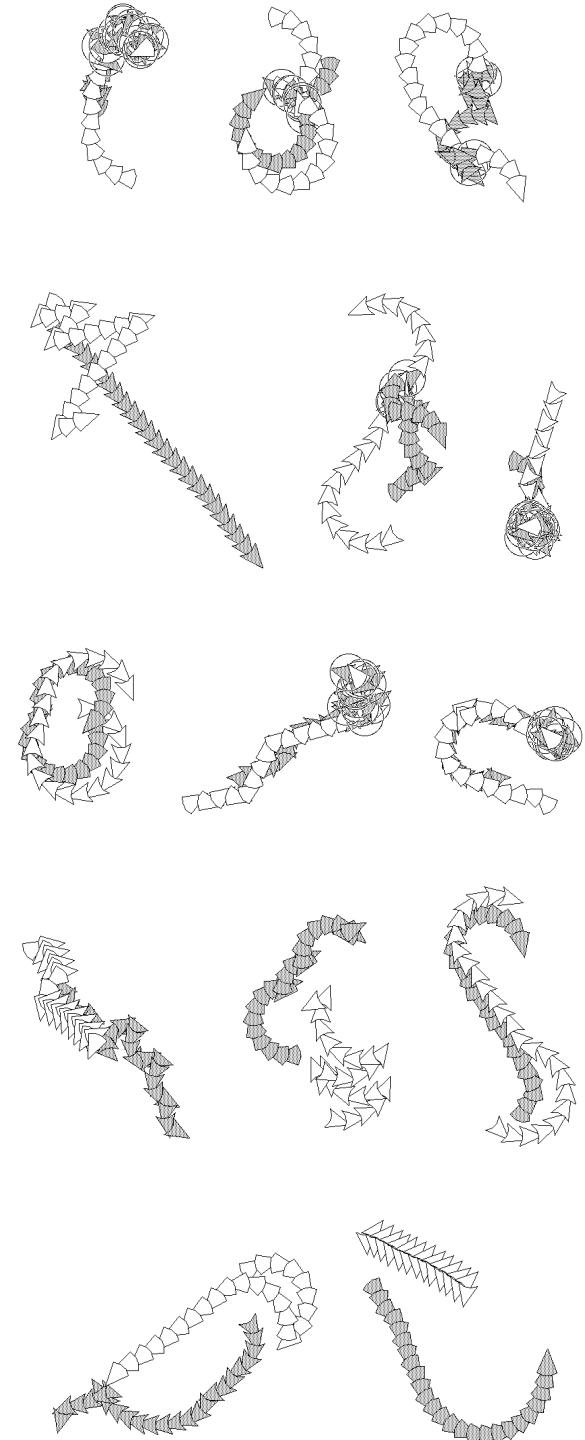
Using the game of tag to test relative fitness, artificial evolution was able to discover skillful tag players in the form of vehicle-steering control programs.

These near-optimal players arose solely from direct competition with each other. Good results were obtained despite the considerable variance of fitness values inherent in the staging of individual games and the random selection of opponents. Fitness was based only on relative performance and did not require knowledge of the optimal strategy.



**Figure 11:** Plots of steering angle versus opponent position for the best-of-population player from run g after 113 generation equivalents

Since the optimal strategy was known for the problem studied here, it could be used as a benchmark. The player population evolved according to relative competitive fitness. Its progress in absolute terms could be measured by placing the evolved players in competition with the optimal player.



**Figure 12:** tag games from run G. A common flaw with all of these players is that they tend to have only two forward steering directions. Efficient pursuit and evasion require the ability to modulate turning angle. Some of these strategies look nothing like optimal behavior. They survive because they have adapted to their environment. For example the pursuer (in white) in the lower right image could never catch a good evader, but it is well suited to mowing down incompetent evaders.

Attempts to characterize the quality of evolved players are subject to differences of interpretation, but in at least one experimental configuration (run C) the population's average performance was within 10% of the optimal player, and the best of population individual performed within a few percentage points of optimal. Note that these quality metrics are only as meaningful as the somewhat *ad hoc* fitness measure used in these experiments.

In other runs described here, the quality of evolved players approached, but did not reach, that of the optimal player. It is not clear if this is because of a fundamental limitation of competitive fitness, a flaw in the experimental design, or simply because of limitations of genetic population size and length of runs.

## 7. Future Work

These experiments used a simple competitive behavior to test the concept of self organizing development of goal-directed behavioral control programs. The generally positive results found here encourage plans for more ambitious future projects.

Eventually the goal is to apply this technique to more complex games. But first, a more elaborate version of the game of tag is an obvious next step. A more complicated control problem, and a more visually interesting class of motion, would result from adding momentum to create a physically realistic model of vehicle motion in place of the current kinematic model. In a physically based model, the angle returned from evolved control programs would be interpreted as the direction of *steering force* to be applied to the vehicle's current momentum. If tag playing vehicles have momentum, their velocity and acceleration become relevant to the control problem. In the real world, the game of tag is usually played with multiple players and in the presence of obstacles. Adding these features, and sensory facilities for dealing with them, would create a rich environment for future studies. In place of the isolated, episodic, pair-wise tag games used in this work, it would be interesting to investigate an ongoing ecological simulation where a large population of agents interact with each other through the game of tag.

In [Angeline 1993] a individual's competitive fitness is determined in a single elimination tournament involving a generation's entire population. In the current work competitive fitness is tested against a randomly selected subset of the existing steady-state population. Several arbitrary choices were made when this selection mechanism was designed. Six opponents are selected with uniform random distribution. Is six too many or too few? Should the selection be skewed toward higher fitness opponents (say by tournament selection) as when parents are selected for crossover? Would judging fitness against the current leaders tend to improve evolvability or stifle it? This approach was attempted in run D but the results were inconclusive. On the other hand this approach is similar to that used in [Sims 1994] which produced good results.

Another approach to steady-state competitive fitness would be to use Angeline's technique of determining fitness

based on standings in a single elimination tournament tree, but to do it in small batches of newly created individuals. A batch of 16 or 32 new individuals would be created, then pitted against each other in a single elimination tournament tree. Their standings in competition with their own littermates would determine their fitness. This approach is similar to that used in [Smith 1994] but would allow fitness values to reflect competition with a wider range of opponents.

The delightful variety of strategies that appeared in these experiments demonstrate that there are many, many way to approach a given task. The road to successful behavior may lie along any of those variations. The most robust evolutionary systems are those that can pursue many approaches at once. Geographically distributed, deme-based systems promote diversity and parallelism. They seem well-suited to competitive evolution of behavior [Ngo 1993] because the quality of competition is better when diversity is preserved.

## Acknowledgments

This work is supported by Electronic Arts. The author wishes to thank Luc Barthelet, Vice President of Technology, for allowing research to coexist with product development. Additional thanks to Luc for help preparing the stimulus-response arrow diagrams. Thanks to John Koza and Pete Angeline for inspiration, encouragement, and critique. Thanks to James Rice for a detailed review. Special thanks to my wife Lisa and to our first child Eric, who was born at just about the same time as individual 15653 of run C.

## References

- [Angeline 1993] Angeline, P. J. (1993) and Pollack, J. B., Competitive Environments Evolve Better Solutions for Complex Tasks, in *Proceedings of the Fifth International Conference on Genetic Algorithms*, S. Forrest, Ed San Mateo, California: Morgan Kaufmann, pages 264-270.
- [Arkin 1987] Arkin, R. C. (1987) Motor Schema Based Navigation for a Mobile Robot: An Approach to Programming by Behavior", *Proceedings of the 1987 IEEE Conference on Robotics and Automation*, Raleigh, North Carolina, pages 264-271.
- [Axelrod 1984] Axelrod, R. (1984) *The Evolution of Cooperation*, Basic Books, New York.
- [Axelrod 1989] Axelrod, R. (1989) Evolution of Strategies in the Iterated Prisoner's Dilemma, in *Genetic Algorithms and Simulated Annealing*, L. Davis editor, Morgan Kaufmann.
- [Braitenberg 1984] Braitenberg, V. (1984) *Vehicles: Experiments in Synthetic Psychology*, MIT Press, Cambridge, Massachusetts.
- [D'haeseleer 1994] D'haeseleer, P (1994) and Bluming, J., Effects of Locality in Individual and Population Evolution, in *Advances in Genetic Programming*, K. E. Kinnear, Jr., Ed. Cambridge, Massachusetts: MIT Press.
- [Hillis 1992] Hillis, W. D. (1992) Co-Evolving Parasites Improve Simulated Evolution as an Optimization Procedure, in *Artificial Life II*, Santa Fe Institute Studies

- in the Sciences of Complexity, Proceedings Volume X, C. Langton, Ed., Addison-Wesley, Redwood City, California, pages 313-324.
- [Holland 1992] Holland, J. H. (1992) *Adaptation in Natural and Artificial Systems*, second edition, MIT Press, Cambridge, Massachusetts.
- [Isaacs 1965] Isaacs, R. (1965) *Differential Games*, John Wiley and Sons, New York.
- [Kauffman 1993] Kauffman, S. A. (1993) *The Origins of Order*, Oxford University Press, New York.
- [Kinnear 1994] Kinnear, K. E., Jr. (1994) Alternatives in Automatic Function Definition: A Comparison of Performance, in *Advances in Genetic Programming*, K. E. Kinnear, Jr., Ed. Cambridge, Massachusetts: MIT Press.
- [Koza 1992] Koza, J. R. (1992) *Genetic Programming: on the Programming of Computers by Means of Natural Selection*, ISBN 0-262-11170-5, MIT Press, Cambridge, Massachusetts.
- [Koza 1994] Koza, J. R. (1994) *Genetic Programming II: Automatic Discovery of Reusable Programs*, ISBN 0-262-11189-6, MIT Press, Cambridge, Massachusetts.
- [Lindgren 1992] Lindgren, K. (1992) Evolutionary Phenomena in Simple Dynamics, in *Artificial Life II*, Santa Fe Institute Studies in the Sciences of Complexity, Proceedings Volume X, C. Langton, Ed., Addison-Wesley, Redwood City, California, pages 295-312.
- [Lindgren 1994] Lindgren, K. (1994) and Nordahl, M., Artificial Food Webs, in *Artificial Life III*, Santa Fe Institute Studies in the Sciences of Complexity, Proceedings Volume XVI, C. Langton, Ed., ISBN 0-201-62494-X, Addison-Wesley, Redwood City, California, pages 73-103.
- [Merz 1971] Merz, A. W. (1971) *The Homicidal Chauffeur: A Differential Game*, (PhD Thesis) Stanford University Center for Systems Research, Report SUDAAR 418.
- [Menczer 1994] Menczer F (1994?) and Belew R. K., Latent Energy Environments, to appear in *Plastic Individuals in Evolving Populations*, Santa Fe Institute Studies in the Sciences of Complexity, Addison-Wesley (in press).
- [Miller 1989] Miller, J. H. (1989) The Co-evolution of Automata in the Repeated Prisoner's Dilemma, Santa Fe Institute Report 89-003.
- [Ngo 1993] Ngo, J. T. (1993) and Marks, J., Spacetime Constraints Revisited, Proceedings of SIGGRAPH 93 (Anaheim, California, August 1-6, 1993), in *Computer Graphics Proceedings*, Annual Conference Series, 1993, ACM SIGGRAPH, New York, pages 343-350.
- [Ray 1992] Ray, T. S. (1992) An Approach to the Synthesis of Life, in *Artificial Life II*, Santa Fe Institute Studies in the Sciences of Complexity, Proceedings Volume X, C. Langton, Ed., Addison-Wesley, Redwood City, California, pages 371-408.
- [Reynolds 1994a] Reynolds, C. W. (1994) An Evolved, Vision-Based Model of Obstacle Avoidance Behavior, in *Artificial Life III*, Santa Fe Institute Studies in the Sciences of Complexity, Proceedings Volume XVI, C. Langton, Ed., ISBN 0-201-62494-X, Addison-Wesley, Redwood City, California, pages 327-346.
- [Reynolds 1994b] Reynolds, C. W. (1994) Evolution of Obstacle Avoidance Behavior: Using Noise to Promote Robust Solutions, in *Advances in Genetic Programming*, K. E. Kinnear, Jr., Ed. Cambridge, Massachusetts: MIT Press.
- [Reynolds 1994c] Reynolds, C. W. (1994) Evolution of Corridor Following Behavior in a Noisy World, to appear in *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior* (SAB94), in press.
- [Sims 1994] Sims, K. (1994) *Evolving 3D Morphology and Behavior by Competition*, Artificial Life IV (in this volume), R. Brooks and Pattie Maes, Eds., MIT Press, Cambridge, Massachusetts.
- [Smith 1994] Smith, R. E. (1994) and Gray, B., Co-Adaptive Genetic Algorithms: An Example in Othello Strategy, to appear in *The Proceedings of The Florida Artificial Intelligence Research Symposium 1994*, and available as *TCGA Report Number 94002*, The University of Alabama, Tuscaloosa.
- [Syswerda 1991] Syswerda, G. (1991) A Study of Reproduction in Generational and Steady-State Genetic Algorithms, in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed. San Mateo, California: Morgan Kaufmann, pages 94-101.
- [Werner 1993] Werner, G. M. (1993) and Dyer, M. G., Evolution of Herding Behavior in Artificial Animals, in *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior* (SAB92), Meyer, Roitblat and Wilson editors, ISBN 0-262-63149-0, MIT Press, Cambridge, Massachusetts, pages 393-399.
- [Yeager 1994] Yeager, L. (1994) Computational Genetics, Physiology, Metabolism, Neural Systems, Learning, Vision, and Behavior or PolyWorld: Life in a New Context, in *Artificial Life III*, Santa Fe Institute Studies in the Sciences of Complexity, Proceedings Volume XVI, C. Langton, Ed., ISBN 0-201-62494-X, Addison-Wesley, Redwood City, California, pages 263-298.



## Artificial Animals in Realistic Virtual Worlds

*Demetri Terzopoulos*

University of Toronto & Intel Corporation



### Research Goals

#### *Computational model of animals*

- Model animals and their habitats realistically
- Essential considerations

{ physics  
locomotion  
perception  
behavior  
learning  
cognition

## Approach



### *Artificial animals*

- Lifelike autonomous agents
- Comprehensive modeling of animals
  - *morphology and appearance*
  - *function of body and brain*  
biomechanics, sensors,  
motor control, perception, behavior,  
learning, cognition, ...

## Artificial Fishes



## Artificial Fishes



## Artificial Fishes



## **State-of-the-Art of Production Computer Animation**



Pixar's "Toy Story"



ILM's "Jurassic Park"

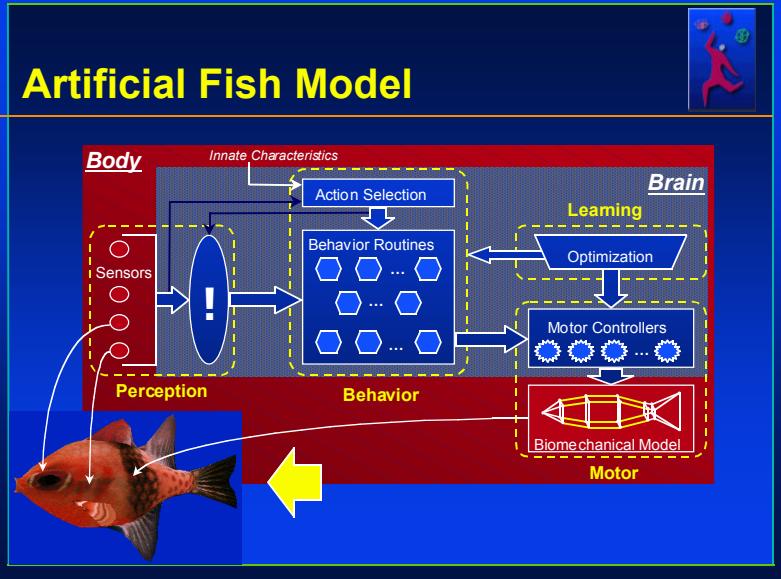
## **Artificial Animals and Animation**



***Computer Animation as artificial nature cinematography***

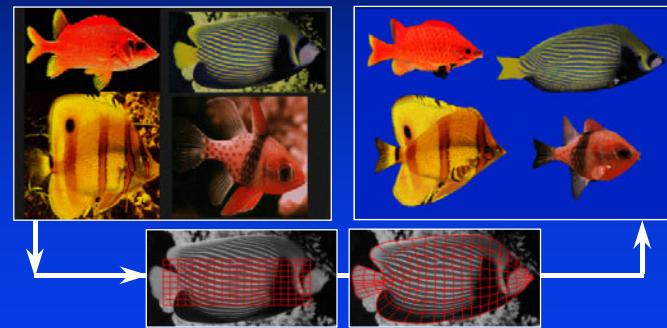


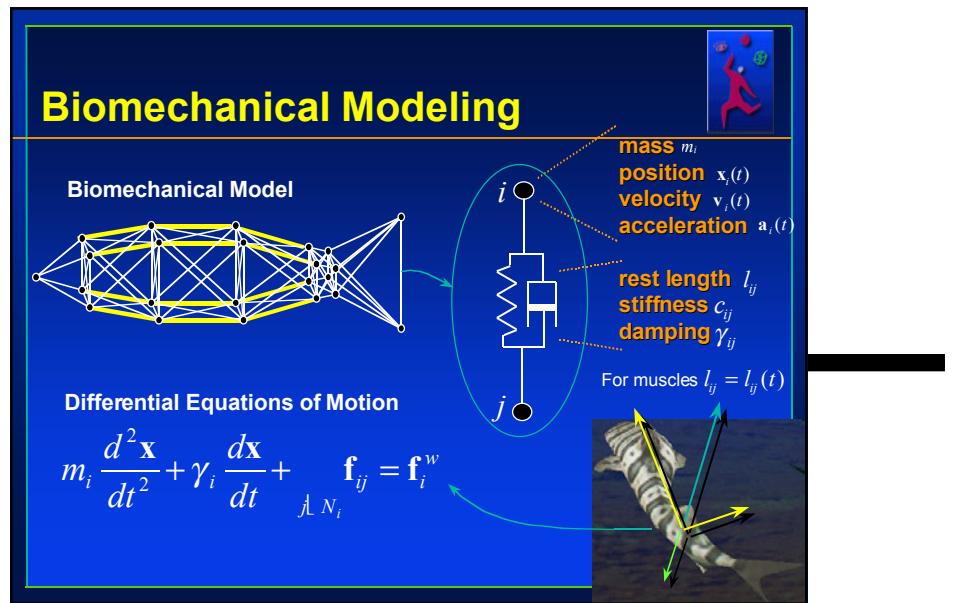
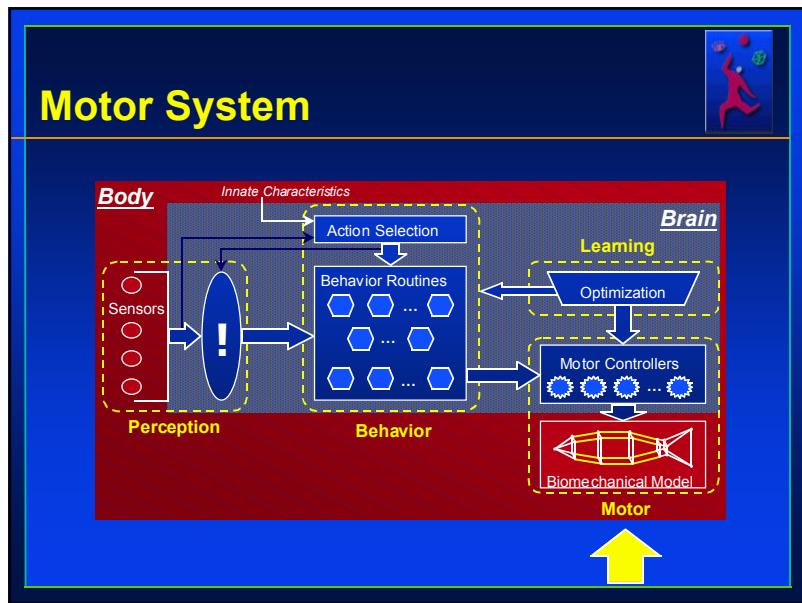
## Artificial Fish Model



## Capturing Form & Appearance

*From images... ...to 3D display models*





## Stable, Implicit Euler Time-Integration Method

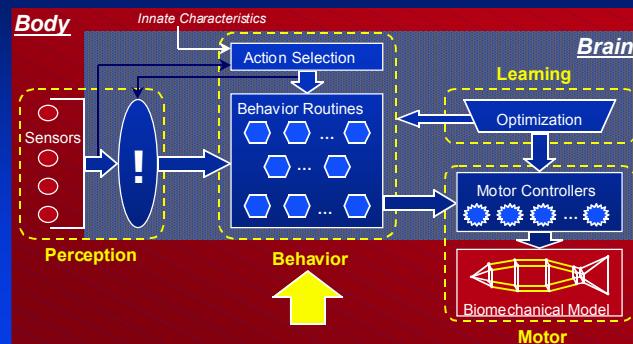


**Solve linear system at each time step**

$$\left\{ \begin{array}{l} \mathbf{A}^{(t)} \dot{\mathbf{x}}^{(t+\delta t)} = \dot{\mathbf{x}}^{(t)} + \mathbf{g}^{(t)} \\ \mathbf{x}^{(t+\delta t)} = dt \dot{\mathbf{x}}^{(t+\delta t)} + \mathbf{x}^{(t)} \end{array} \right.$$

- Efficient skyline storage of  $\mathbf{A}^{(t)}$
- LU factorization of  $\mathbf{A}^{(t)}$
- Forward / Back substitution solves for  $\dot{\mathbf{x}}^{(t+\delta t)}$

## Behavior Center



## Behavior



### Fixed & variable quantities

- Innate characteristics
  - gender, preferences, abilities
- Mental state
  - Hunger (appetite, ingestion, digestion)
  - Libido (sex drive)
  - Fear (predator proximity)

## Action Selection



### Decision tree intention generator

- Priority ordering of events based on danger
  - collision avoidance
  - predator avoidance (school, scatter, flee)
  - eating
  - mating (courtship dance, loop)
- Selected action dispatches behavior routine

## Artificial Fish Types

*Specializing the generic model*

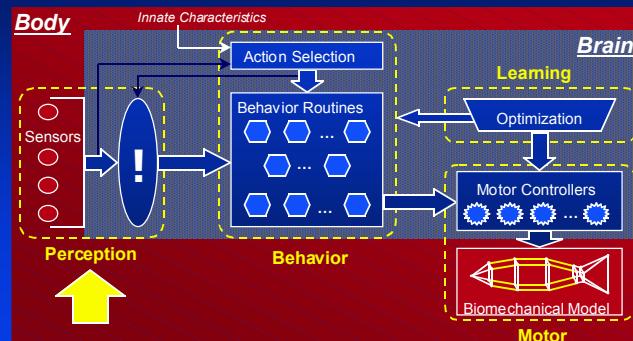
Pacifists



Predators & Prey

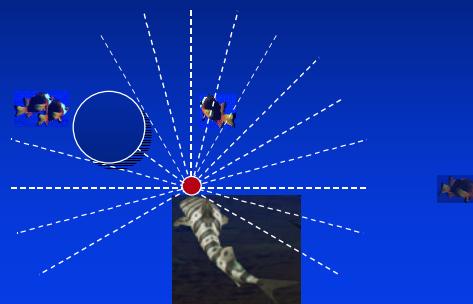


## Perception System



## Perception

**Model abilities \*and\* limitations**



## Iconic Perception

**Intrinsic images**



Retinal Image



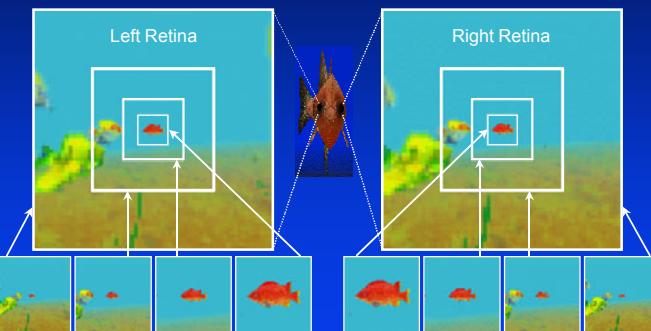
Range Image



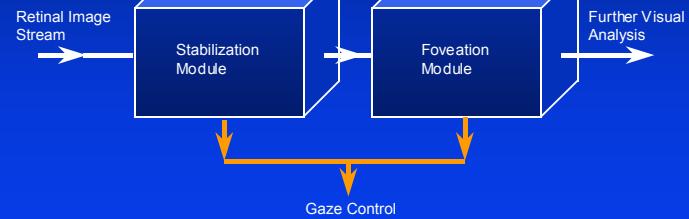
Object Identity Image

## Virtual Eyes

***Binocular, foveated retinal imaging***



## Active Vision System

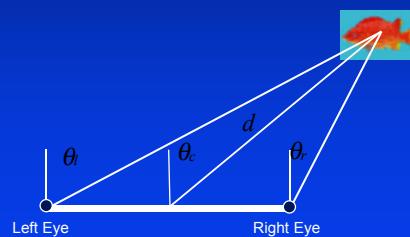


## Vision-Guided Navigation

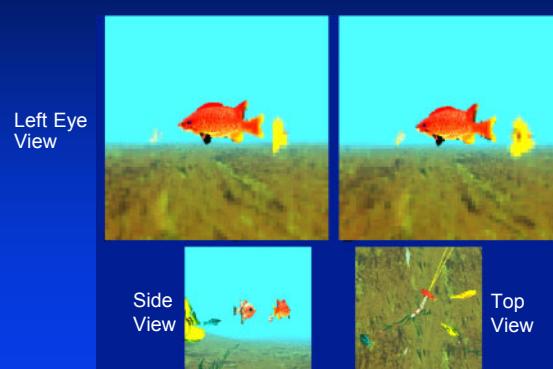


### Adaptive sensorimotor loop

- Navigation using gaze directions
  - *gaze angles and range-to-target geometry*



## Visual Pursuit of Moving Target (Demo)

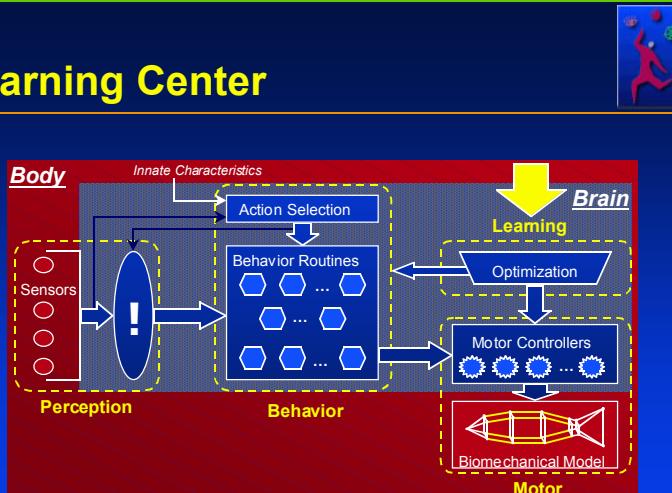


Right Eye View

Side View

Top View

## Learning Center



## Learning Locomotion

### Question:

Can an artificial animal learn to control its muscles to yield natural looking locomotion?

### Existence proof:

Real animals learn how to locomote

### Challenge:

Develop general, automatic locomotion learning algorithms for physics-based artificial animals

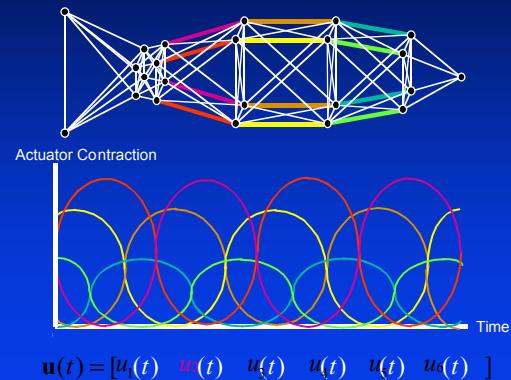
## Perception-Based Learning of Locomotion



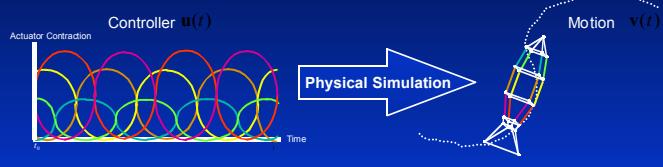
### First principles

- To locomote, animals exploit
  - *physical environment*
  - *biomechanics / muscles*
  - *perception*
- Natural locomotion is energetically efficient
- Formulation: Learning as optimization
  - *multilevel learning strategy*

## Controller



## Objective Function



$$E(u(t)) = \int_{t_1}^{t_0} \mu_1 E_u(u(t)) + \mu_2 E_v(v(t)) dt$$

## Other Biomechanical Models



Snake Model

Actuators: 20  
Point Masses: 46  
DOF: 158



Ray Model

Actuators: 16  
Point Masses: 21  
DOF: 79

## Composing Macro Controllers

### **Learn basic controllers (low-level)**

- Swim forward, turn left, turn right, dive,...

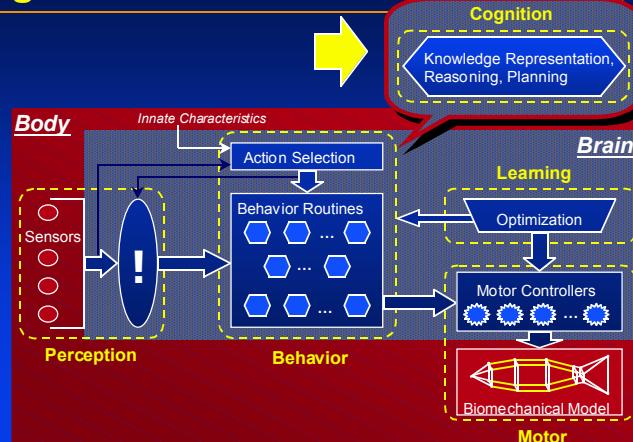
### **Abstract basic controllers**

- Dimensionality reduction

### **Combine basic controllers**

- Discover and refine macro controllers that solve complex motor tasks

## Cognition



## Cognition

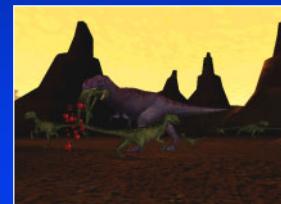


### ***Knowledge representation, reasoning, planning***

- Logic-based representation
  - *structure of physical world*
  - *causal relationships*
  - *the situation calculus*
- Cognitive agent vs purely reactive agent

## “Demosaurus Rex”

(Intel & Angel Studios)



## Interactive Artificial Marine World



### Silicon Graphics RealityStation

- MIPS R10000 CPU
- InfiniteReality graphics pipeline
- CrystalEyes stereoscopic display



## Synthetic Motion Capture



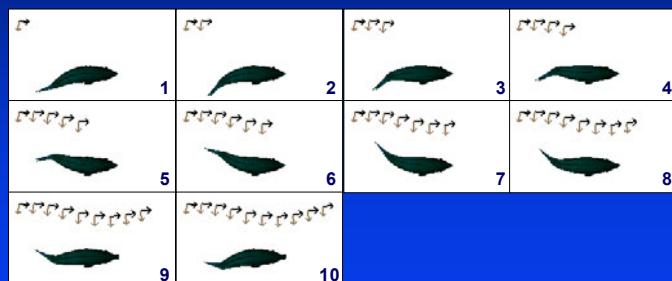
### *Compile action repertoires from biomechanically synthesized motions*

- Dramatic increase in animation rate for real-time virtual reality
- Accelerate display with level-of-detail geometric modeling and rendering
- Create interactive virtual marine experiences

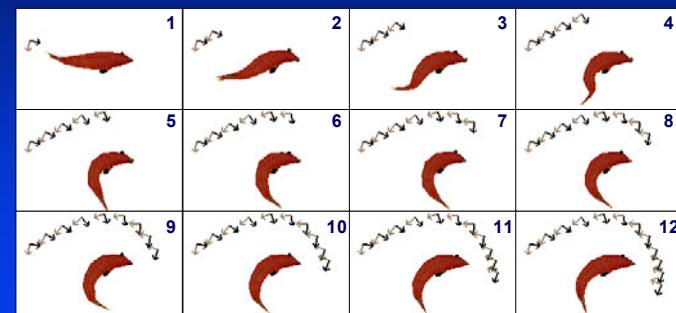
## Captured Forward Swim Action Segment



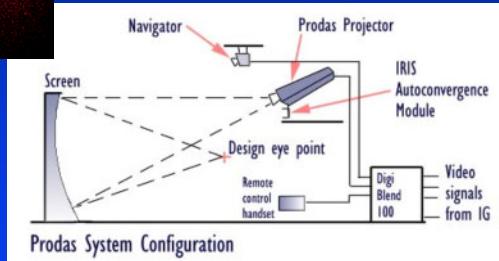
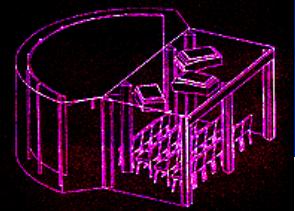
*Separate locomotion into rigid motion of local coordinate system & deformation*



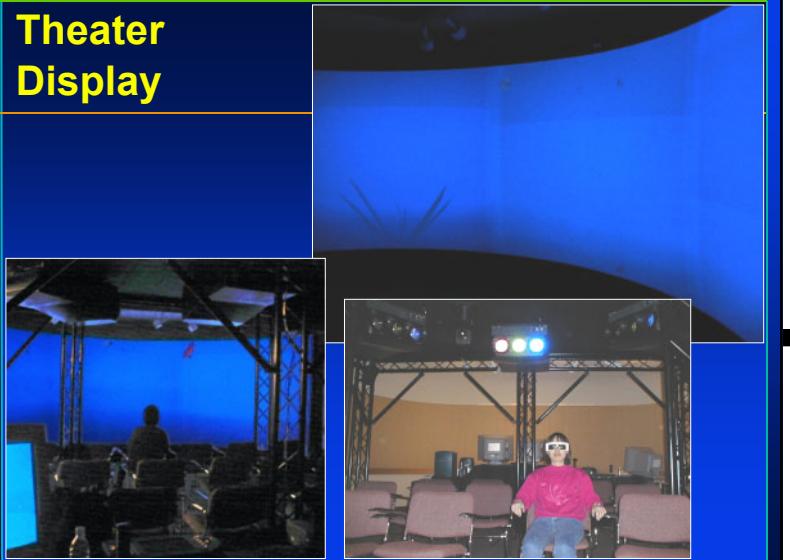
## Captured Right Turn Action Segment



## Portable Reality Theater (Trimension, Inc.)



## Theater Display



## VR Engine

### **Silicon Graphics Onyx2 System**

- 8 MIPS R10000 CPUs
- 3 InfiniteReality graphics pipelines
  - *Each pipeline feeds 1280 x 1024 pixel resolution graphics input to projector*



## Conclusion

### **Artificial animals**

- Comprehensive modeling of body and brain biomechanics, sensors, motor control, perception, behavior, learning, cognition, ...
- Apps: computational zoology, paleobiology
- Future work
  - *simulation of development*
  - *evolution of artificial animals*

## Acknowledgements



### ***Many thanks to:***

- |                    |       |                     |
|--------------------|-------|---------------------|
| • Xiaoyuan Tu      | Intel | (artificial fishes) |
| • John Funge       | Intel | (cognition)         |
| • Radek Grzeszczuk | Intel | (learning)          |
| • Tamer Rabie      | UofT  | (animat vision)     |
| • Qinxin Yu        | UofT  | (synthetic mo-cap)  |

### ***Additional info:***

<http://www.cs.toronto.edu/~dt>

## Realistic Facial Modeling

***Demetri Terzopoulos***

University of Toronto & Intel Corporation



**Scanned Data →**  
**Functional, Synthetic Head/Face**



**A Physics-Based Face Model**  
(Terzopoulos & Waters 1990)

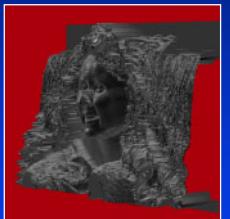


#### **Hierarchical structure**

- Expression: Facial action coding system (FACS)
- Control: Coordinated facial actuator commands
- Muscles: Contractile muscle fibers exert forces
- Physics: Muscle forces deform synthetic tissue
- Geometry: Expressive facial deformations
- Images: Rendering by graphics pipeline

## Raw Input Dataset (“Heidi”)

*From CyberWare 3D Color Digitizer*

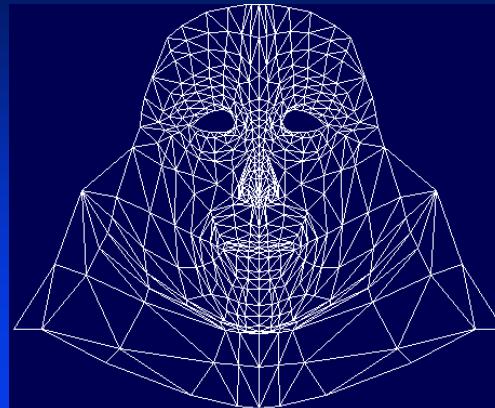


Range Image



RGB Texture Image

## Generic Facial Mesh



## Fitting the Generic Mesh



### *Feature-based image matching algorithm*

localizes facial  
features in:

Processed range image

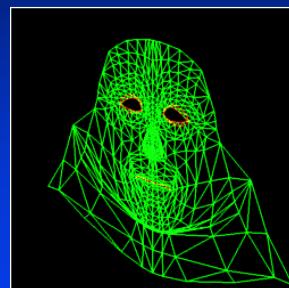
RGB texture image



## Sampling Facial Shape



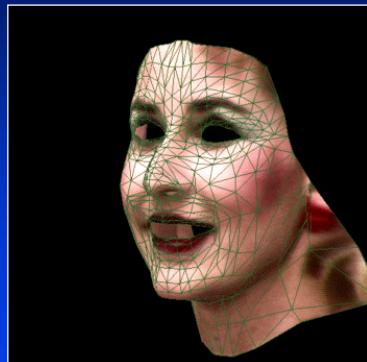
### *Fitted mesh nodes sample range data*



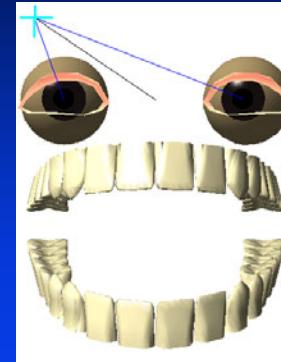
## Textured 3D Geometric Model

### *Texture map coordinates*

- Positions of fitted mesh nodes in RGB texture image



## Auxiliary Geometric Models



Eyelid Texture Interpolation



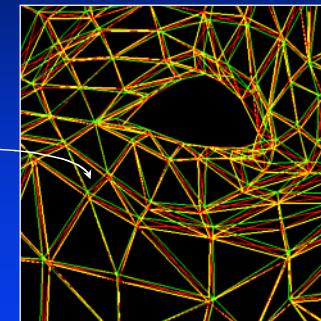
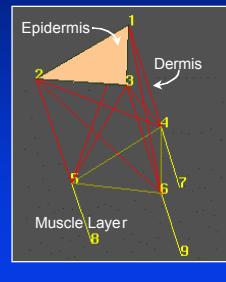
## Complete Geometric Model

*Neutral expression  
is estimated*



## Biomechanical Skin Model

*Deformable finite-element facial tissue*



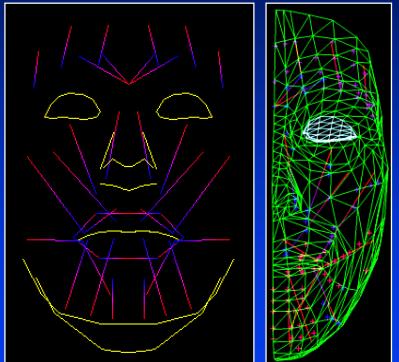
## Facial Muscle Model Structure

### 35 Muscles

- Levator Oculii
- Corrugators
- Naso-Labial
- Zygomatics
- Obicularis Oris

### plus

- Articulate Jaw
- Eyes/Eyelids



## Muscle-Actuated Expressions



## Muscle-Actuated Expressions



## Muscle-Actuated Expressions



## Muscle-Actuated Expressions



## Muscle-Actuated Expressions



## Raw CyberScans



Heidi



George



Giovanni



Mick



## Functional Head Models (Giovanni & Mick)



## Functional Model of George



## George in “Bureaucrat Too”



## Acknowledgements



### ***Many thanks to:***

- Yuancheng (Victor) Lee, UofT (realistic facial model)
- Keith Waters, Digital Cambridge Research Lab

### ***Additional info:***

<http://www.cs.toronto.edu/~dt>

# Artificial Fishes: Autonomous Locomotion, Perception, Behavior, and Learning in a Simulated Physical World

Demetri Terzopoulos, Xiaoyuan Tu, and Radek Grzeszczuk

Department of Computer Science, University of Toronto  
10 King's College Road, Toronto, Ontario, M5S 1A4, Canada  
e-mail: {dt|tu|radek}@cs.toronto.edu

Published in *Artificial Life*, 1(4):327–351, 1994.

## Abstract

*This paper develops artificial life patterned after animals as evolved as those in the superclass Pisces. It demonstrates a virtual marine world inhabited by realistic artificial fishes. Our algorithms emulate not only the appearance, movement, and behavior of individual animals, but also the complex group behaviors evident in many aquatic ecosystems. We model each animal holistically. An artificial fish is an autonomous agent situated in a simulated physical world. The agent has (i) a three-dimensional body with internal muscle actuators and functional fins, which deforms and locomotes in accordance with biomechanical and hydrodynamic principles, (ii) sensors, including eyes that can image the environment, and (iii) a brain with motor, perception, behavior, and learning centers. Artificial fishes exhibit a repertoire of piscine behaviors that rely on their perceptual awareness of their dynamic habitat. Individual and emergent collective behaviors include caudal and pectoral locomotion, collision avoidance, foraging, preying, schooling, and mating. Furthermore, artificial fishes can learn how to locomote through practice and sensory reinforcement. Their motor learning algorithms discover muscle controllers that produce efficient hydrodynamic locomotion. The learning algorithms also enable artificial fishes to train themselves to accomplish higher level, perceptually guided motor tasks, such as maneuvering to reach a visible target.*

**Keywords:** artificial life, autonomous agents, animats, artificial fishes, learning, behavior, perception, locomotion, physics-based modeling, computer graphics.

Published in *Artificial Life*, 1(4):327–351, 1994.

## 1 Introduction

Imagine a virtual marine world inhabited by a variety of realistic fishes.<sup>1</sup> In the presence of underwater currents, the fishes employ their muscles and fins to swim gracefully around immobile obstacles and among moving aquatic plants and other fishes. They autonomously explore their dynamic world in search of food. Large, hungry predator fishes stalk smaller prey fishes in the deceptively peaceful habitat. Prey fishes swim around contentedly until the sight of predators compels them to take evasive action. When a dangerous predator appears in the distance, similar species of prey form schools to improve their chances of survival. As the predator nears a school, the fishes scatter in terror. A chase ensues in which the predator selects victims and consumes them until sated. Some species of fishes seem untroubled by predators. They find comfortable niches and feed on floating plankton when they get hungry. Driven by healthy libidos, they perform elaborate courtship rituals to secure mates.

The computer emulation of the above scenario presents a formidable challenge in the field of artificial life. In this paper we propose a computational framework for creating *fully functional artificial animals*—in this instance, *artificial fishes*. Artificial fishes are autonomous agents with functional bodies controlled by brains. Their appearance, motivations, and complicated group interactions aspire to be as faithful as possible to nature's own. To simulate artificial worlds with the level of complexity of the one depicted above, we have taken a bottom-up, compositional approach. In our approach we model not just 3D form and appearance, but also the basic physics of the animal and its environment. Upon the simulated physics substrate, we can effectively model the animal's means of locomotion. This in turn positions us to model the animal's perceptual awareness of its world, its behavior, and its ability to learn. Our holistic approach to modeling the animal and its world is crucial to achieving realism.

The long-term goal of our research is a computational theory that can potentially account for the interplay of physics, locomotion, perception, behavior, and learning in higher animals. A good touchstone of such a theory is its ability to produce visually convincing results in the form of realistic computer graphics animation with little or no animator intervention. We have been able to achieve such results in two instances to date. Our animation “Go Fish!” [23] shows a colorful variety of artificial fishes foraging in translucent water. A sharp hook on a line descends towards the hungry fishes and attracts them. A hapless fish, the first to bite the bait, is caught and drawn to the surface. The color plates show stills from our 1994 animation “The Undersea World of Jack Cousto.” Plate 1a shows a variety of animated artificial fishes. The reddish fish are engaged in a mating ritual, the greenish fish is a predator hunting for small prey, the remaining fishes are feeding on plankton (white dots). Dynamic seaweeds grow from the ocean bed and sway in the current. In Plate 1b, the large male in the foreground is courtship dancing with the female (top). The prey fish in the background are engaging in schooling behavior, a common subterfuge for avoiding predators. Plate 1c shows a shark stalking the school. The detailed motions of the artificial fishes emulate the complexity and unpredictability of movement of their natural counterparts, and this enhances the visual beauty of the animations.

---

<sup>1</sup>“Fish” is both singular and plural; when plural, it refers to more than one fish *within* the same species. The plural “fishes” is used when two or more species are involved [25].



Plate 1a: Artificial fishes in their physics-based world.

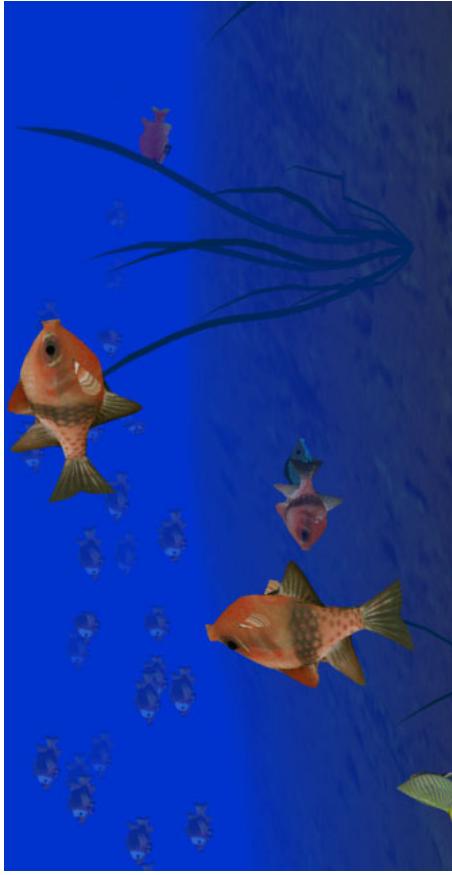


Plate 1b: Mating behaviors. Female (top) is courted by large male.



Plate 1c: Predator shark stalking school of prey fish.

### 1.1 Background

Our approach to developing artificial animals is consistent with the “animal” approach proposed by Wilson [26]. To render our computational model visually convincing, we attempt as well to capture with reasonable fidelity the appearance and physics of the animal and its world. Artificial fishes may be viewed as animals of unprecedented sophistication. They are autonomous virtual robots situated in a continuously dynamic 3D virtual world. Their functional design, including motor control, perceptual modeling, and behavioral simulation presents hurdles paralleling those encountered in building physical autonomous agents (see, e.g., the compilation [13]). Previously, the most complex animals were inspired by insects. Brooks (see [8]) describes a physical insect robot “Genghis”, bristling with sensors, that can locomote over irregular terrain, while Beer develops a virtual counterpart, a cockroach with simple behaviors in a 2D world [5] (see also [13]).

Our work tackles animals much more highly evolved and complex than insects. To deal with the broad behavioral repertoire of fishes, we exploit ideas from classical ethology [22, 12, 15, 1]. Tinbergen’s landmark studies of the three-spined stickleback highlight the great diversity of piscine behavior, even within a single species. We achieve the nontrivial patterns of behavior outlined in the introductory paragraph of this paper in stages. First, we implement primitive reflexive behaviors, such as obstacle avoidance, that directly couple perception to action [7]. Then we combine the primitive behaviors into motivational behaviors whose activation depends also on the artificial fish’s mental state, including hunger, libido, and fear.

Useful behavior is supported by perception of the environment as much as it is by action. Reynolds’ “boids” maintained flocking formations through perception of other nearby boids [20]. Recently Matarić has demonstrated similar flocking behaviors with physical robots [14]. Artificial fishes sense their world through simulated visual perception within a deliberately limited field of view. They can sense lighting patterns, determine distances to objects, and identify objects, subject to the natural limitations of occlusion. Furthermore, they are equipped with secondary nonvisual modalities, such as the ability to sense local water temperature.

At its lowest level, our work makes use of computational physics. We model the biomechanics of a broad class of fishes and their muscle-based locomotion abilities that exploit the physics of their liquid medium [6, 2]. The mechanical model that we develop was inspired by the simple but surprisingly effective computer graphics model of snake and worm dynamics proposed by Miller [17]. We have provided artificial fish with algorithms that enable them to learn automatically from first principles how to

achieve hydrodynamic locomotion by controlling their internal muscle actuators. The locomotion learning algorithm that we describe is more continuous and closer connected to actuation than most of the animat “behavior learning” algorithms surveyed in [16]. Our multilevel reinforcement learning procedure first performs a global search for actuator activation functions that produce efficient locomotion. The process then abstracts these activation functions into a highly compact representation. The representation emphasizes the natural periodicities of the derived muscle actions and makes explicit the coordination among multiple muscles that leads to effective locomotion. Finally, the artificial fish can put into practice the compact, efficient controllers that it has learned and train itself to accomplish higher level sensorimotor tasks.

## 1.2 Functional Overview of the Artificial Fish

Fig. 1 presents an overview of an artificial fish situated in its simulated physical world. The body of the fish harbors a brain or mind with motor, perception, behavior, and learning centers.

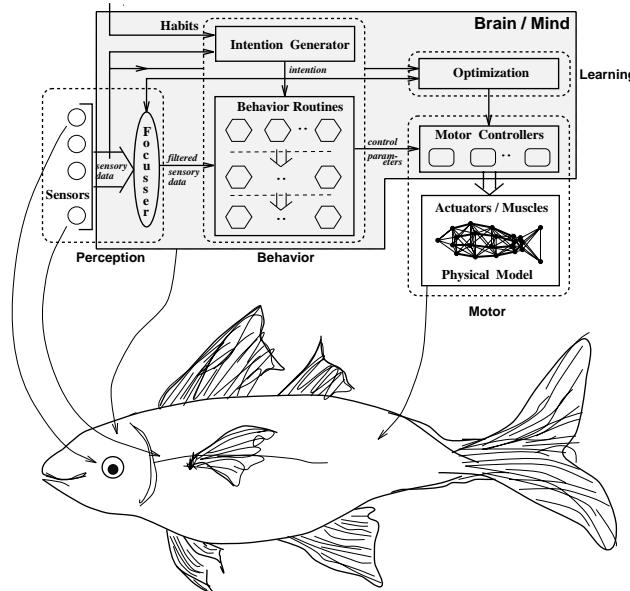


Figure 1: Control and information flow in artificial fish.

The motor system, comprising the actuators and a set of motor controllers (MCs), drives the dynamic model of the fish. We have crafted a mechanical body model that represents a good compromise between anatomical consistency, hence realism, and computational efficiency. Our model is rich enough so that we can build MCs by gleanning information from the fish biomechanics literature [6, 2]. The MCs are parameterized procedures. Each is dedicated to carrying out a specific motor function, such as “swim forward” or “turn left.” They translate natural control parameters such as the forward speed or angle of the turn into detailed muscle actions.

The perception system relies on a set of on-board virtual sensors to provide sensory information about the dynamic environment, including eyes that can produce time-varying retinal images of the environment. The brain’s perception center includes a perceptual attention mechanism which allows the artificial fish to train its sensors at the

world in a task-specific way, hence filtering out sensory information superfluous to its current behavioral needs. For example, the artificial fish attends to sensory information about nearby food sources when foraging.

The behavior center of the artificial fish’s mind mediates between its perception system and its motor system. An intention generator, the fish’s cognitive faculty, harnesses the dynamics of the perception-action cycle. The innate character of the fish is established by a set of habits that determine whether or not it is male or female, predator or prey, etc. The intention generator combines the habits and mental state with the incoming stream of sensory information to generate dynamic goals for the fish, such as to hunt and feed on prey. It ensures that goals have some persistence by exploiting a single-item memory. The intention generator also controls the perceptual attention mechanism. At every simulation time step, the intention generator activates behavior routines that attend to sensory information and compute the appropriate motor control parameters to carry the fish one step closer to fulfilling its current intention. Primitive behavior routines, such as obstacle avoidance, and more sophisticated motivational behavior routines, such as mating, implement the behavioral repertoire of the artificial fish.

The learning center of its mind enables the artificial fish to learn how to locomote through practice and sensory reinforcement. Through optimization, the motor learning algorithms discover muscle controllers that produce efficient locomotion. Their brain’s learning center also enable artificial fishes to train themselves to accomplish higher level sensorimotor tasks, such as maneuvering to reach a visible target.

## 2 Realistic Modeling of Form and Appearance

To achieve realism, our artificial fish model must first represent the form and appearance of real fishes with sufficient fidelity. To this end, we have perused photographs of real fishes, such as those shown in Fig. 2(a), and have built 3D geometric models of several different species using nonuniform rational B-spline (NURBS) surfaces (Fig. 2(b)).

The next step is to map realistic textures onto the geometric fish model (Fig. 2(e)). We extract natural textures from digital images of the fish photos, employing a “snake-grid” tool to determine appropriate texture map coordinates in the image. Snakes are interactive deformable contours that are subject to a force field derived from an image [11]. The force field attracts them towards interesting image features such as intensity edges. A collection of coupled snakes forms a deformable grid (Fig. 2(c-d)). The snake-grid floats freely over an image and it can be pulled into position using the mouse. When its border approaches the intensity edges that demarcate the fish from its background in the image, the border snakes lock on and adhere to these edges. The remaining snakes in the grid relax elastically to cover the imaged fish body with a nonuniform but smooth coordinate system (Fig. 2(d)). The snake crossing points serve as texture map image coordinates for the NURBS surfaces.

## 3 Physics-Based Fish Model and Locomotion

Studies into the dynamics of fish locomotion show that most fishes use their caudal fin as the primary motivator [24]. Caudal swimming normally uses posterior muscles on either side of the body, while turning normally uses anterior muscles. To synthesize realistic fish locomotion we have designed a dynamic fish model consisting of 23 nodal point masses and 91 springs. The spring arrangement maintains the structural stability of the body while allowing it to flex. Twelve of the springs running the length of the body also serve as simple muscles (Fig. 3).

### 3.1 Mechanics

The mechanics of the spring-mass model are specified as follows: Let node  $i$  have mass  $m_i$ , position  $\mathbf{x}_i(t) = [x_i(t), y_i(t), z_i(t)]$ , velocity  $\mathbf{v}_i(t) = d\mathbf{x}_i/dt$ , and acceleration  $\mathbf{a}_i(t) = d^2\mathbf{x}_i/dt^2$ . Let elastic spring  $S_{ij}$  connect node  $i$  to node  $j$  and denote its spring constant as  $c_{ij}$  and natural, rest length as  $l_{ij}$ . Its deformation is  $e_{ij}(t) = \|\mathbf{r}_{ij}\| - l_{ij}$ , where  $\mathbf{r}_{ij}(t) = \mathbf{x}_j - \mathbf{x}_i$ . The force  $S_{ij}$  exerts on node  $i$  is  $\mathbf{f}_{ij}^s(\mathbf{x}_i, \mathbf{x}_j) = c_{ij}e_{ij}\mathbf{r}_{ij}/\|\mathbf{r}_{ij}\|$  (note that  $\mathbf{f}_{ij}^s = -\mathbf{f}_{ji}^s$ ). The Lagrange equations of motion of the dynamic fish are:

$$m_i \frac{d^2\mathbf{x}_i}{dt^2} + \rho_i \frac{d\mathbf{x}_i}{dt} - \mathbf{w}_i = \mathbf{f}_i^w; \quad i = 0, \dots, 22, \quad (1)$$

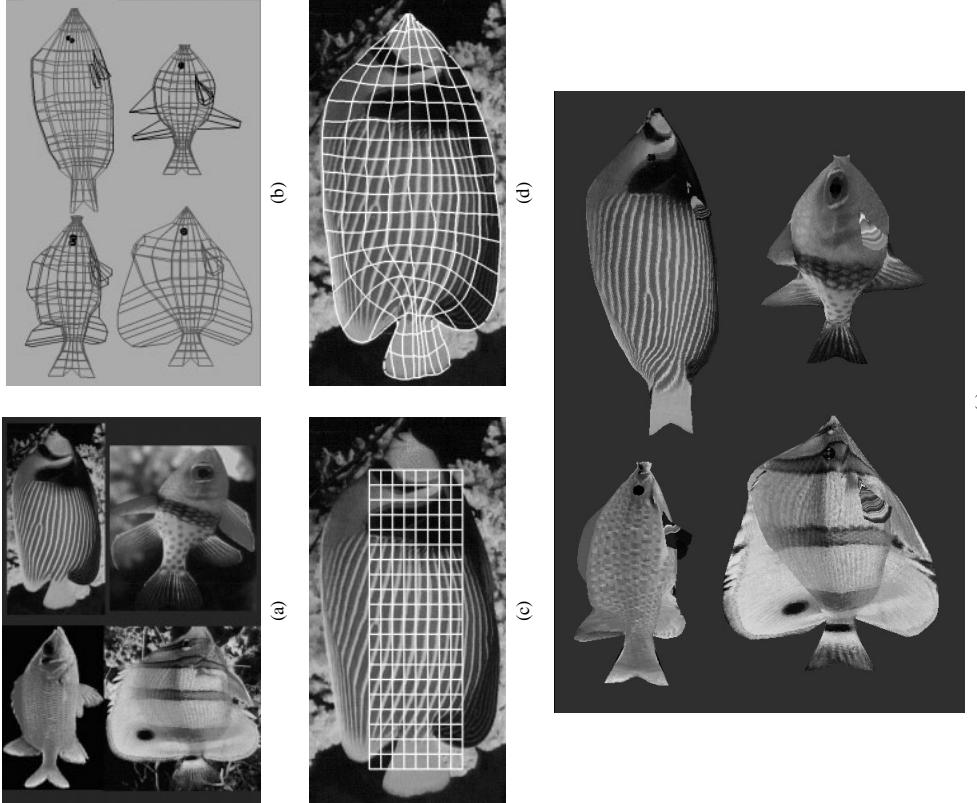


Figure 2: (a) Digitized images of fish photos. (b) 3D NURBS surface fish bodies. Initial (c) and final (d) snake-grid covering an imaged fish body. (e) Texture mapped 3D fish models.

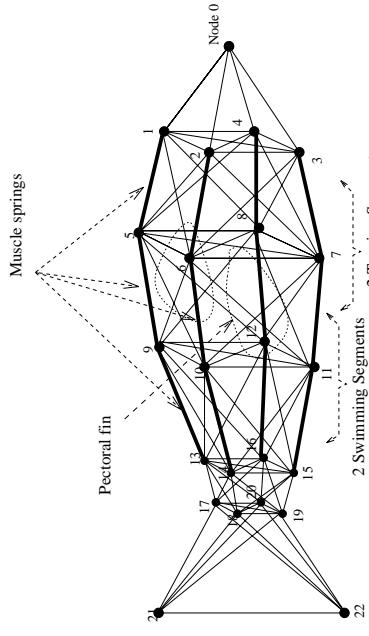


Figure 3: Dynamic fish model. Nodes are lumped masses. Lines are springs (shown at their natural lengths). Bold lines are muscle springs.

where  $\rho_i$  is the damping factor,  $\mathbf{w}_i(t) = \sum_{j \in N_i} \mathbf{f}_{ij}^s$  is the net internal force on node  $i$  due to springs connecting it to nodes  $j \in N_i$ , where  $N_i$  is the index set of neighboring nodes. Finally,  $\mathbf{f}_i^w(t)$  is the external (hydrodynamic) force on node  $i$ .

To integrate the differential equations of motion, we employ a numerically stable, implicit Euler time stepping method [19]. Since the elastic forces depend nonlinearly on the position variables  $\mathbf{x}_i$ , the method assembles the sparse stiffness matrix for the spring-mass system in efficient “skylne” storage format, then factorizes and solves the resulting system of algebraic equations at every time step to obtain the position increments.<sup>2</sup>

We couple the control points of the aforementioned texture mapped NURBS body model to the time-varying positions of the mass points (the nodes in Fig. 3), such that the fish body deforms in accordance with the simulated dynamics of the actuated spring-mass system.

### 3.2 Swimming Using Muscles and Hydrodynamics

The artificial fish moves as a real fish does, by contracting its muscles. If  $S_{ij}$  is a muscle spring, it is contracted by decreasing the rest length  $l_{ij}^r$ . For convenience, we assign a minimum contraction length  $l_{ij}^{\min}$  to the muscle spring and express the contraction factor as a number in the range  $[0, 1]$ . The brain of the fish controls the muscles continuously through time by specifying the vector of muscle actuation functions  $\mathbf{u}(t) = [u_1(t), \dots, u_{12}(t)]$ , whose components specify a time-varying contraction factor for each of the 12 muscles. The characteristic undulation of the fish's tail can be achieved by periodically contracting the swimming segment springs on one side of the body while relaxing their counterparts on the other side.

When the fish's tail beats, it sets in motion a volume of water. The inertia of the displaced water produces a reaction force normal to the fish's body proportional to the volume of water displaced per unit time, which propels the fish forward (Fig. 4). Under certain assumptions, the instantaneous force on the surface  $S$  of a body due to a viscous fluid is approximately proportional to  $-\int_S (\mathbf{n} \cdot \mathbf{v}) \mathbf{n} dS$ , where  $\mathbf{n}$  is the unit outward normal function over the surface and  $\mathbf{v}$  is the relative velocity function between the surface and the fluid. For efficiency, we triangulate the surface of the dynamic fish model between the nodes and approximate the force on each planar triangle as

<sup>2</sup>In our simulation:  $m_i = 1.1$  for  $i = 0$  and  $13 \leq i \leq 19$ ;  $m_i = 6.6$  for  $1 \leq i \leq 4$  and  $9 \leq i \leq 12$ ;  $m_i = 11.0$  for  $5 \leq i \leq 8$ , and  $m_i = 0.165$  for  $i = 21, 22$ . The cross springs ( $\mathbf{e}_{ij}, c_{ij}$ ) which resist shearing have spring constant  $c_{ij} = 38.0$ . The muscle springs (e.g.,  $c_{35}$ ) have spring constant  $c_{ij} = 28.0$  and  $c_{ij} = 30$  for the remaining springs. The damping factor  $\rho_i = 0.05$  and the time step used in the Euler time-integration procedure is 0.055.

$f = \min[0, -A(\mathbf{n} \cdot \mathbf{v})\mathbf{n}]$ , where  $A$  is the area of the triangle and  $\mathbf{v}$  is its velocity relative to the water. The  $\mathbf{f}_i^w$  variables at each of the three nodes defining the triangle are incremented by  $f/3$ .

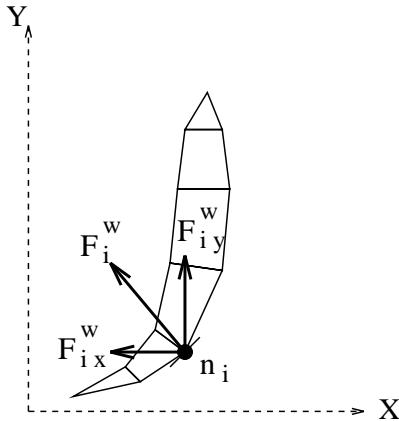


Figure 4: Hydrodynamic locomotion. With tail swinging towards positive  $X$  axis, reaction force  $\mathbf{F}_i^w$  at point  $n_i$  acts along the inward normal. Component  $\mathbf{F}_{ix}^w$  resists the lateral movement, while  $\mathbf{F}_{iy}^w$  is forward thrust. Aggregate thrust propels fish towards positive  $Y$  axis.

### 3.3 Motor Controllers

Currently the artificial fish has three MCs. The swim-MC produces straight swimming, while the left-turn-MC and right-turn-MC execute turns. The MCs prescribe muscle contractions to the mechanical model. The swim-MC controls the swimming segment muscles (see Fig. 3), while the turn MCs control the turning segment muscles.

According to [24], the swimming speed of most fishes is roughly proportional to the amplitude and frequency of the periodic lateral oscillation of the tail, below certain threshold values. Our experiments with the mechanical model agree well with these observations. Both the swimming speed and the turn angle of the fish model are approximately proportional to the contraction amplitudes and frequencies/rates of the muscle springs.

The swim-MC (swim-MC(speed)  $\mapsto \{r_1, s_1, r_2, s_2\}$ ) converts a swim speed parameter into contraction amplitude and frequency control parameters for the anterior ( $r_1, s_1$ ) and posterior ( $r_2, s_2$ ) swim segments. One pair of parameters suffices to control each of the two swim segments due to symmetry—the four muscle springs have identical rest lengths and minimum contraction lengths, identical spring constants, and the contractions of the muscle spring pairs on opposite sides are exactly out of phase. Moreover, the swim-MC produces periodic muscle contractions in the posterior swim segment which lag 180 degrees behind those of the anterior swim segment; hence the mechanical model displays a sinusoidal body shape as the fish swims (see [24]).

By experimenting, we have found a set of four maximal parameters,  $\hat{r}_1, \hat{s}_1, \hat{r}_2$  and  $\hat{s}_2$ , which produce the fastest swimming speed. The swim-MC generates slower swim speeds by specifying parameters that have values between 0 and the maximal parameters. For example,  $\{0.8\hat{r}_1, \hat{s}_1, 0.7\hat{r}_2, \hat{s}_2\}$  results in a slower-swimming fish.

As mentioned earlier, most fishes use their anterior muscles for turning, and the turn angle is approximately proportional to the degree and speed of the anterior bend, up to the limit of the fish's physical strength [24]. The artificial fish turns by contracting and expanding the springs of the turning segments (Fig. 3) in similar fashion. For example, a left turn is achieved by quickly contracting the left side springs of the segments and relaxing those on the right side. This deflects the fish's momentum and brings it into the desired orientation. Then the contracted springs

are restored to their rest lengths at a slower rate, so that the fish regains its original shape with minimal further change in orientation.

Similarly, the left and right turn MCs (turn-MC(angle)  $\mapsto \{r_0, s_0, r_1, s_1\}$ ) convert a turn angle to control parameters for the anterior and posterior turning segments to execute the turn (note that the posterior turning segment also serves as the anterior swim segment). Through experimentation, we established 4 sets of parameter values  $P_i = \{r_0^i, s_0^i, r_1^i, s_1^i\}$  which enable the fish to execute natural looking turns of approximately 30, 45, 60, and 90 degrees. By interpolating the key parameters, we define a steering map that allows the fish to generate turns of approximately any angle up to 90 degrees. Turns greater than 90 degrees are composed as sequential turns of lesser angles.

### 3.4 Pectoral Fins

On most fishes the pectoral fins control pitching (the up-and-down motion of the body) and yawing (the side-to-side motion). The pectorals can be held close to the body to increase speed by reducing drag or they can be extended to increase drag and serve as a brake [25]. Many reef fishes use a pectoral swimming style, keeping their bodies still and using their pectoral fins like oars to achieve fine motion control, including reverse motions, when foraging.

The artificial fish is neutrally buoyant in the virtual water and has a pair of pectoral fins which enable it to navigate freely in its 3D world. The pectoral fins function in a similar, albeit simplified, manner to those on real fishes. For our purposes the detailed movement of the pectoral fins is of lesser interest than the movement of the fish body. To simplify the fish model and its numerical solution, we do not simulate the elasticity and dynamics of the pectoral fins. However, we do approximate the dynamic forces that the pectoral fins exert on the body of the fish to control locomotion.

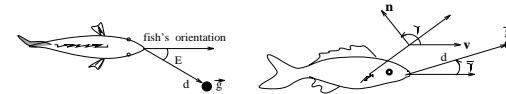


Figure 5: The pectoral fins

The pectoral fins (Fig. 5) work by applying reaction forces to nodes in the midsection, i.e. nodes  $1 \leq i \leq 12$  (see Fig. 3). The fins are analogous to the airfoils of an airplane. Pitch, yaw, and roll control stems from changing their orientations relative to the body; i.e., the angle  $\pi/4 \leq \gamma \leq \pi$ . Assuming that a fin has an area  $A$ , surface normal  $\mathbf{n}$  and the fish has a velocity  $\mathbf{v}$  relative to the water (Fig. 5), the fin force is  $F_f = -A(\mathbf{n} \cdot \mathbf{v})\mathbf{n} = -A(\|\mathbf{v}\| \cos \gamma)\mathbf{n}$  which is distributed equally to the 12 midsection nodes. When the leading edge of a fin is elevated, a lift force is imparted on the body and the fish ascends, and when it is depressed a downward force is exerted and the fish descends. When the fin angles differ the fish yaws and rolls. The artificial fish can produce a braking effect by angling its fins to decrease its forward speed (i.e.  $\gamma = \pi$ ). This motion control is useful in maintaining schooling patterns, for instance.

## 4 Learning Muscle-Based Locomotion

We have discussed how locomotion controllers may be carefully hand crafted using knowledge gleaned from the piscine biomechanics literature and long hours of experimentation with our mechanical fish model. In this section, we consider the following general question: Given a physics-based model of an animal with internal muscle actuators capable of producing locomotion, such as the fish model of Fig. 3, is it possible for the model to learn from first principles how to control its actuators in order to locomote in a natural fashion? Furthermore, can it employ the controllers that it has learned in order to accomplish higher level tasks guided by sensory perception? We demonstrate affirmative answers to both questions by developing a learning center in the mind of the artificial fish that applies a form of reinforcement learning.

#### 4.1 Learning Strategy

We formulate a two-phase, bottom-up strategy for learning muscle controllers. The artificial fish has a fully functional body, but at first it is “brain-dead” and does not know how to use its muscles to locomote. In phase one, it repeatedly practices a variety of muscle activation functions and remembers activation patterns that improve its locomotion, thus learning how to locomote with increasingly better efficiency. Repeated improvements eventually lead to natural looking locomotion patterns that are optimally efficient.

When an adequate degree of optimization has been achieved in the low-level learning phase, the learning algorithm enters phase two, where it abstracts the activation functions into a highly compact representation. The representation drastically reduces the dimensionality of the learning problem by using basis functions that make explicit the natural periodicities of the derived muscle actions and the pattern of coordination among multiple muscles that yields effective locomotion. The artificial fish trains itself with routine locomotion drills and associates the low-level muscle activation functions that it has learned with specific higher-level tasks that it needs to perform. Finally, it can put to use the compact, efficient controllers that it has learned, to accomplish higher level sensorimotor tasks—for example, it can locomote and execute turning maneuvers to reach a visible target. The learned controllers can subsequently be employed during advanced behaviors such as hunting.

#### 4.2 Low-Level Learning

At the foundation of our approach lies the notion that natural motion patterns are energetically efficient. This allows us to reduce the problem of learning realistic locomotion into a problem of optimizing an objective function, for which various solution techniques are available.

Fig. 6 illustrates the learning algorithm. The objective function takes the form

$$E(\mathbf{u}(t)) = \int_{t_0}^{t_1} (\mu_1 E_u(\mathbf{u}(t)) + \mu_2 E_v(\mathbf{v}(t))) dt, \quad (2)$$

a weighted sum, with weighting variables  $\mu_1, \mu_2$ , of a term  $E_u$  that evaluates the vector of muscle actuator control functions  $\mathbf{u}(t)$ , which dictate the lengths  $l_{ij}$  of muscle springs in the dynamic model (see Sec. 3.2), and a term  $E_v$  that evaluates the resulting trajectory  $\mathbf{v}(t)$  of the artificial fish. Note that to compute  $\mathbf{v}(t)$  and hence  $E$ , we must perform a forward simulation of the dynamic model over a time interval  $t_0 \leq t \leq t_1$  with the actuation function inputs  $\mathbf{u}(t)$ .

The term  $E_u$  guides the optimization by discouraging large, rapid fluctuations of  $\mathbf{u}$ . The rationale is that chaotic muscle actuations usually produce energy-inefficient body motions. We encourage smoothness through the function

$$E_u = -\frac{1}{2} \left( \nu_1 \left| \frac{d\mathbf{u}}{dt} \right|^2 + \nu_2 \left| \frac{d^2\mathbf{u}}{dt^2} \right|^2 \right), \quad (3)$$

with weighting factors  $\nu_1$  and  $\nu_2$ . The two terms are potential energy densities of linear and cubic splines in time, respectively. The former penalizes muscle effort more than the latter.

The criterion  $E_v$  for a good trajectory that we used most often in our learning experiments was the final distance of the fish from a target location. Depending on the task, other possible criteria are: the closeness to a specified speed, the closeness to a specified trajectory, etc.

Learning low level control involves the application of simulated annealing to optimize (2) [19]. Simulated annealing is applied after discretizing the actuator control functions  $\mathbf{u}(t)$  by sampling them in the time interval under consideration to obtain the set of discrete samples  $\mathbf{u}_i = \mathbf{u}(t_i)$  for  $1 \leq i \leq N$  (typically, we set  $N$  to 15 time samples, and recover the continuous  $\mathbf{u}(t)$  through linear or cubic spline interpolation of the  $\mathbf{u}_i$ ). The annealing algorithm repeatedly perturbs the  $\mathbf{u}_i$  to modify the actuator activation functions that control the muscles in the fish. It retains those perturbations that produce increasingly better locomotion as measured by the objective function  $E$ , and sometimes accepts those that don’t escape local minima. Note that after performing a forward simulation using  $\mathbf{u}(t)$ , the artificial fish can evaluate  $E$  with its on-board sensors. Hence, learning proceeds autonomously.

Fig. 7 shows six artificial leopard sharks in a race. The furthest shark has completed only 90 annealing steps,

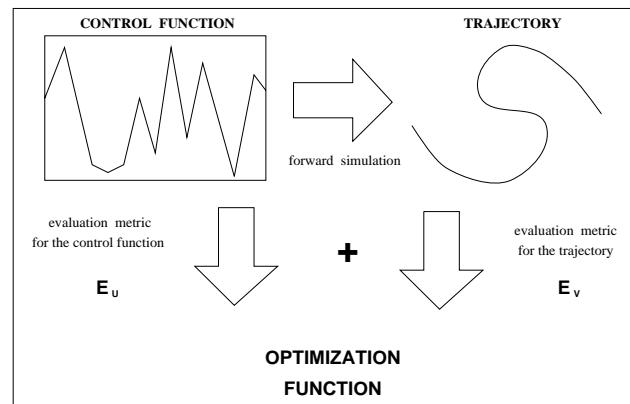


Figure 6: Low-level algorithm for learning locomotion. Muscle control inputs actuate body of artificial animal to produce a locomotion trajectory through forward physical simulation. A fitness function measures the quality of input control and output trajectory.

which results in muscle control functions that are essentially random and achieve very poor locomotion. Nearer sharks have learned for progressively longer periods of time (1350 more annealing steps). After learning for 6840 annealing steps, the nearest shark locomotes very efficiently and it wins the race decisively.

#### 4.3 Abstraction of High-Level Controllers

In the second stage of the learning process abstracts higher-level muscle controllers. This is a dimensionality reducing change of representation that compresses the information content of the many control points to a compact form in terms of a few global basis functions. Specifically, it tries to represent the control functions as accurately as possible in the form  $\mathbf{u}(t) \approx \sum_{i=0}^M \alpha_i \mathbf{B}_i(t)$ , where  $\mathbf{B}_i$  are basis functions,  $\alpha_i$  are scalar quantities, and  $M$  is a small number.

Since natural locomotion patterns are generally periodic [18], the Fourier basis is a reasonable choice. We employ the short time FFT to perform the change in basis (the wavelet transform may also be applied). If the Fourier space is a suitable representation that captures the temporal structure of the control functions, the dimensionality reduction can be achieved easily by eliminating all basis functions whose coefficients  $\alpha_i$  in the above approximation formula are negligible. This will result in a small set of  $M$  significant basis functions, usually 1 or 2, with associated coefficients that define the abstracted muscle controller.

The artificial fish can now train itself to perform several routine locomotion tasks, such as swimming forward at different speeds and executing turns of different radii. Fig. 8 illustrates the control abstraction procedure and its results after straight swim and left turn training sessions. After it has abstracted controllers for these tasks, the artificial fish can construct a forward speed and steering map by interpolating across amplitudes, frequencies, and phases of the sinusoidal basis functions in the set of abstracted controllers. Finally, it can put these learned abstractions to use to accomplish higher level tasks, such as target tracking. Fig. 9 shows a shark model that has trained itself to swim between targets (spherical buoys); having swum from the left to the far target, it has now turned and is proceeding to the near target.

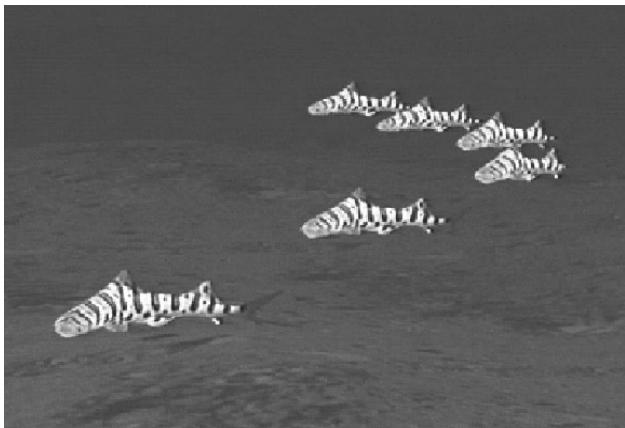


Figure 7: Race between sharks that have learned to locomote for progressively longer times.

## 5 Sensory Perception

The perception system of the artificial fish, illustrated in Fig. 1, comprises a set of virtual on-board sensors and a perceptual focuser. Currently the artificial fish is equipped with two sensors that provide information about the dynamic environment—a temperature sensor that measures the ambient (virtual) water temperature at the center of the artificial fish’s body and a much more elaborate cyclopean vision sensor.

### 5.1 Vision Sensor

We have not attempted to emulate the vision system of real fishes. Instead, we have incorporated a basic cyclopean vision sensor into the fish model. It is crucial to model the basic limitations of animal vision systems, otherwise the perceptually driven behaviors will not be natural. The cyclopean vision sensor has a 300 degree spherical field of view extending frontally and laterally to an effective radius  $V_r$  appropriate to the visibility of the translucent water (Fig. 10(a)). An object is “seen” only if some part of it enters this view volume and it is not fully occluded behind some other opaque object (Fig. 10(b)).

The artificial fish’s vision sensor has access to the geometry, material property, and illumination information that is available to the graphics pipeline for rendering purposes. In addition, the vision sensor can interrogate the world model database to identify nearby objects and interrogate the physical simulation to obtain information such as the instantaneous positions and velocities of objects of interest. In this way, the vision sensor extracts from the 3D virtual world only some of the most useful information that piscine visual processes can provide real fishes about their world, such as overall brightness, and the colors, sizes, distances, and identities of visible objects.

A more biologically plausible emulation of piscine visual processes would involve the application of various computer vision algorithms [10] to extract information from “retinal” images of the 3D world rendered from the vantage point of the artificial fish’s cyclopean vision sensor. Intrinsic images are a useful paradigm for this purpose [4]. The rendering pipelines of 3D graphics workstations can readily synthesize retinal images, associated z-buffers, and object identity maps for artificial fish vision. Currently, the fish determines only the overall brightness of its environment by computing the mean intensity of the retinal image. Fig. 11 shows examples of retinal images acquired by a fish “witnessing” another fish being baited by a fishing line.

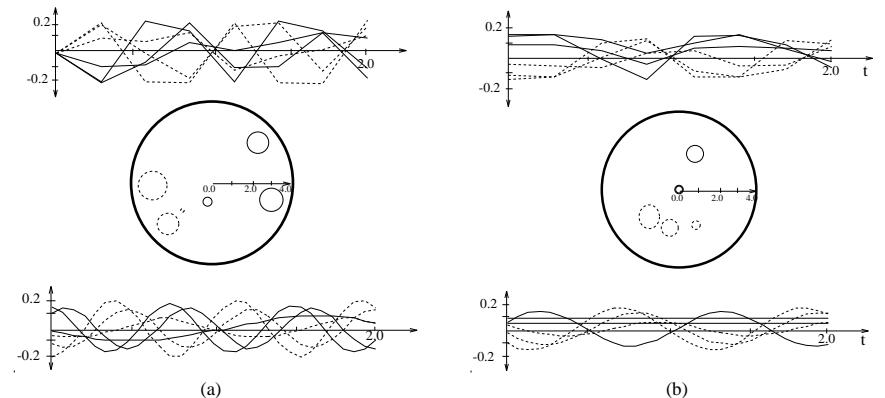


Figure 8: Abstraction of muscle controllers in straight swim (a) and left turn (b) training sessions. Upper plots: muscle controls gleaned by low-level learning algorithm. Center plots: primary modes. Lower plots: compact muscle controls abstracted from primary modes.

## 6 Behavioral Modeling

The artificial fish’s behavior system runs continuously within the simulation loop. At each time step the intention generator issues an intention based on the fish’s habits, mental state, and incoming sensory information. It then chooses and executes a behavior routine which in turn runs the appropriate motor controllers. It is important to note that the behavior routines are incremental by design. Their job is to bring the artificial fish one step closer to fulfilling the intention during the current time step. The intention generator employs a memory mechanism to avoid dithering.

### 6.1 Habits and Mental State

The innate character of the fish is determined by a set of habit parameters that determine whether or not it likes brightness, darkness, cold, warmth, schooling, or is a male/female, etc.

The artificial fish has three mental state variables, hunger  $H$ , libido  $L$ , and fear  $F$ . The range of each variable is  $[0, 1]$ , with higher values indicating a stronger urge to eat, mate and avoid danger, respectively. The variables are calculated as follows:

$$\begin{aligned} H(t) &= \min[1 - n^e(t)R(\Delta t^H)/\alpha, 1], \\ L(t) &= \min[s(\Delta t^L)(1 - H(t)), 1], \\ F(t) &= \min\left[\sum_i F^i, 1\right], \text{ where } F^i = \min[D_0/d^i(t), 1]; \end{aligned}$$

where  $t$  is time,  $n^e(t)$  is the amount of food consumed as measured by the number of food particles or prey fishes eaten,  $R(x) = 1 - p_0x$  with constant  $p_0$  is the digestion rate,  $\Delta t^H$  is the time since the last meal,  $\alpha$  is a constant that dictates the appetite of the fish (bigger fishes have a larger  $\alpha$ ),  $s(x) = p_1x$  with constant  $p_1$  is the libido function,  $\Delta t^L$  is the time since the last mating,  $D_0 = 100$  is a constant, and  $F^i$  and  $d^i$  are, respectively, the fear of and distance to sighted predator  $i$ . Nominal constants are  $p_0 = 0.00067$  and  $p_1 = 0.0025$ . Certain choices can result in ravenous fishes (e.g.,  $p_0 = 0.005$ ) or sexual mania (e.g.,  $p_1 = 0.01$ ).

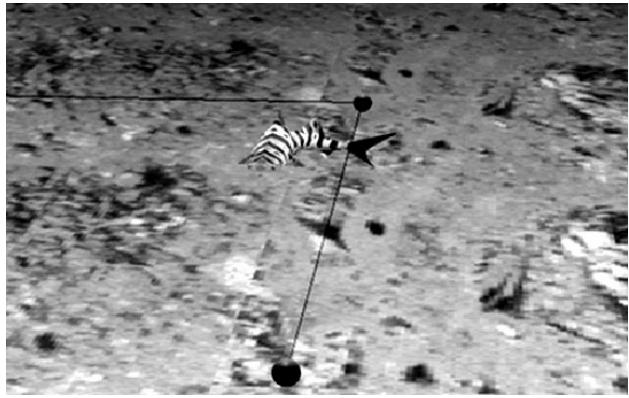


Figure 9: Trained shark swimming between targets.

## 6.2 Intention Generator

Fig. 12 illustrates the generic intention generator which is responsible for the goal-directed behavior of the artificial fish in its dynamic world.

The intention generator first checks the sensory information stream to see if there is any immediate danger of collision. If any object penetrates the fish's collision sensitivity region (a bounding box) then the intention  $I$  generated is to *avoid* collision. A large sensitivity region results in a 'timid' fish that takes evasive action to avoid a potential collision well in advance, while a tight sensitivity region yields a 'courageous' fish that takes evasive action only at the last second.

If there is no immediate danger of collision, the neighborhood is searched for predators, the fear state variable  $F$  and the most dangerous predator  $m$  for which  $F^m \geq F^i$  are calculated. If the total fear  $F > f_0$  (where  $0.1 \leq f_0 \leq 0.5$  is a threshold value) evasive action is to be taken. If the most dangerous predator is not too threatening (i.e.  $F^m < f_1$  where  $f_1 > f_0$ ) and the fish has a schooling habit, then the *school* intention is generated, otherwise the *escape* intention is generated.

If fear is below threshold, the hunger and libido mental state variables  $H$  and  $L$  are calculated. If the greater of the two exceeds a threshold  $0 < r < 0.5$ , the intention generated will be to *eat* or *mate* accordingly.

If the above test fails, the intention generator accesses the ambient light and temperature information from the perception system. If the fish's habits dictate contentment with the ambient conditions, the intention generated will be to *wander* about, otherwise it will be to *leave* the vicinity.

Note that after the intention generator chooses an intention, it invokes the perceptual focus mechanism. For example, when the *avoid* intention is generated, the perception focuser is activated to locate the positions of the obstacles, paying special attention to the most dangerous one, generally the closest. Then the intention generator computes motor preferences (qualitative constraints, such as *another fish to the left  $\Rightarrow$  no left turn*). The focuser passes only the position of the most dangerous obstacle along with these constraints to the behavior routines. When the intention of a male fish is to *mate*, the focuser targets the most desirable female fish; when the intention is to *escape* from predators, only the information about the most threatening predator is passed to the next layer; etc.

## 6.3 Behavior Memory and Persistence

In a complex dynamic world, the artificial fish should have some persistence in its intentions, otherwise it will tend to dither, perpetually switching goals. If the current behavior is interrupted by a high priority event, the intention generator stores in a single-item short term memory the current intention and some associated information that may

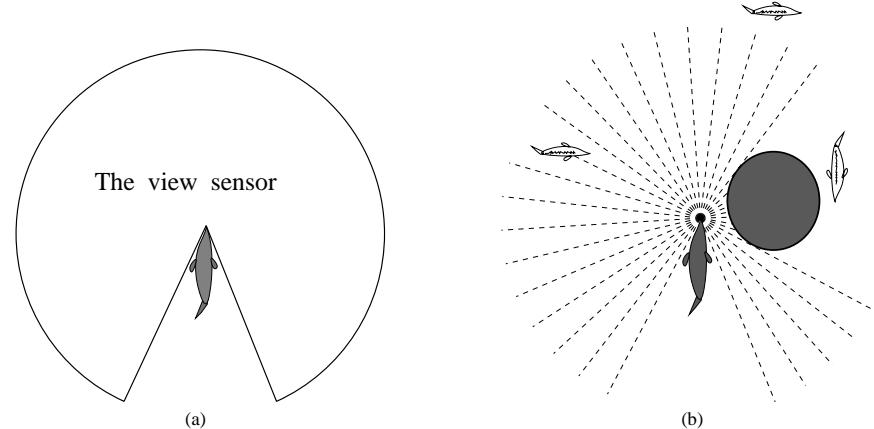


Figure 10: Artificial fish vision sensor. (a) Visual perception is limited to 300 degree solid angle. (b) Occlusion and distance limits the perception of objects (only the fish towards the left is visible).

be used to resume the interrupted behavior. Persistence is particularly important in making long duration behaviors such as foraging, schooling, and mating more robust. Suppose, for example, that the current behavior is mating and an imminent collision is detected with another fish. This causes an *avoid* intention and the storage of the *mate* intention (we refer to the stored intention as  $I^*$ ) along with the identity of the mating partner. After the obstacle has been cleared, the intention generator commands the focuser to generate up-to-date heading and range information about the mating partner, assuming it is still in viewing range.

Our design of the intention generator and focuser simplifies the modification of existing personalities and behaviors and the addition of new ones. For example, we can create artificial fishes with different persistencies by augmenting the focuser with a new positive threshold. Suppose the current intention of a predator fish is to *eat* and let the distance to some currently targeted prey be  $l_c$  and the distance to some other prey be  $l_n$ . If  $l_c - l_n$  is greater than the threshold, the fish will target the new prey. Varying the threshold will vary the fish's level of persistence. The same heuristic can be applied to mates when the fish is trying to *mate*. One can make the fish 'fickle' by setting the value of the threshold close to zero or make it 'devoted' by setting a large value.

## 6.4 Behavior Routines

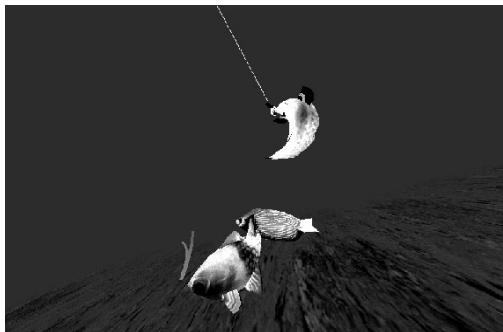
Once the intention generator selects an intention it attempts to satisfy the intention by passing control to a behavior routine along with the data from the perception focuser. The artificial fish currently includes eight behavior routines: *avoiding-static-obstacle*, *avoiding-fish*, *eating-food*, *mating*, *leaving*, *wandering*, *escaping*, and *schooling* which serve the obvious purposes. The behavior routine uses the focused perceptual data to select an MC and provide it with the proper motor control parameters. We now briefly describe the function of the routines.

The *avoiding-static-obstacle* and *avoiding-fish* routines operate in similar fashion. Given the relative position of the obstacle, an appropriate MC (e.g. *left-turn-MC*) is chosen and the proper control parameters are calculated subject to the motor preferences imposed by other surrounding obstacles. For efficiency the *avoid-fish* routine treats the dynamic obstacle as a rectangular bounding box moving in a certain direction. Although collisions between fishes cannot always be avoided, bounding boxes can be easily adjusted such that they almost always are, and the method is very efficient. An enhancement would be to add collision resolution.

The *eating-food* routine tests the distance  $d$  from the fish's mouth to the food (see Fig. 5). If  $d$  is greater than some



(a)



(b)

Figure 11: Fisheye view of the world showing fishing line (a) and hooked fish (b).

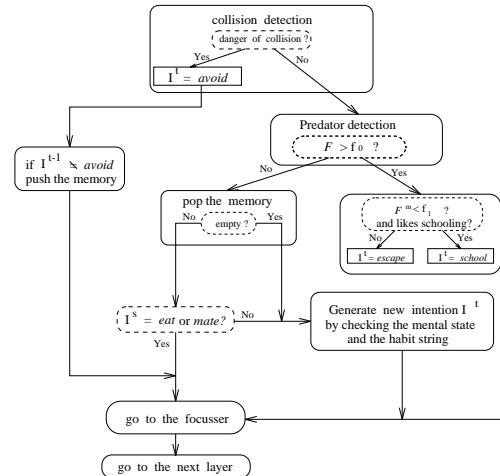
threshold value, the subroutine *chasing-target* is invoked.<sup>3</sup> When  $d$  is less than the threshold value the subroutine *suck-in* is activated where a “vacuum” force (to be explained in Sec. 6.1) is calculated and then exerted on the food.

The *mating* routine invokes four subroutines: *looping*, *circling*, *ascending* and *nuzzling* (see Sec. 7.3 for details). The *wandering* routine sets the fish swimming at a certain speed by invoking the swim-MC, while sending random turn angles to the turn-MCs. The *leaving* routine is similar to the *wandering* routine. The *escaping* routine chooses a suitable MC according to the relative position, orientation of the predator to the fish. The *schooling* routine will be discussed in Sec. 7.2.

## 7 Artificial Fish Types

In the introductory paragraph of the paper we described the behavior of three types of artificial fishes—predators, prey, and pacifists. This section presents their implementation details.

<sup>3</sup>The *chasing-target* subroutine guides a fish as it swims towards a goal. It plays a crucial role in several behavior routines.

Figure 12: Generic intention generator (simplified). Set of intentions: { *avoid*, *escape*, *school*, *eat*, *mate*, *leave*, *wander* }.  $f_0$  and  $f_1$  are thresholds with  $f_0 < f_1$ .

### 7.1 Predators

Fig. 13 is a schematic of the intention generator of a predator, which is a specialized version of Fig. 12. For simplicity, predators currently are not preyed upon by other predators, so they perform no predator detection, and *escape*, *school*, and *mate* intentions are disabled ( $F = 0$ ,  $L = 0$ ). Since predators cruise perpetually, the *leave* intention is also disabled.

Generally prey is in less danger of being hunted when it is far away from the predator, or is in a school, or is behind the predator. A predator chases prey  $k$  if the cost  $C_k = d_k(1 + \beta_1 S_k + \beta_2 E_k/\pi)$  of reaching it is minimal. Here,  $d_k$  is the distance between the mouth of the predator and the center of prey  $k$ 's body,  $S_k = 1$  if prey  $k$  is in a school of fishes, otherwise  $S_k = 0$ , and the angle  $E_k \in [0, \pi]$  (Fig. 5) measures the turning cost.  $\beta_1$  and  $\beta_2$  are parameters that tune the contributions of  $S_k$  and  $E_k$ . We use  $\beta_1 = 0.5$  and  $\beta_2 = 0.2$  in our implementation of the focuser. Plate 1c shows a shark predator stalking a school of prey fish.

Most teleost fishes do not bite on their victims like sharks do. When a fish is about to eat it swims close to the victim and extends its protrusile jaw, thus creating a hollow space within the mouth. The pressure difference between the inside and the outside of the mouth produces a vacuum force that sucks the victim and anything else in the nearby water into the mouth. The predator closes its mouth, expels the water through the gills, and grinds the food with pharyngeal jaws [25]. We simulate this process by enabling the artificial fish to open and close its mouth kinematically. To suck in prey, it opens its mouth and, while the mouth is open, exerts vacuum forces on fishes (the forces are added to external nodal forces  $f_i$  in equation (1) and other dynamic particles in the vicinity of the open mouth, drawing them in (Fig. 14).

### 7.2 Prey

The intention generator of a prey fish is given by specializing the generic intention generator of Fig. 12, as shown in Fig. 15.

Schooling and evading predators are the two distinct behaviors of prey. We briefly describe the implementation of the *schooling* behavior. Schooling is a complex behavior where all the fishes swim in generally the same direction.

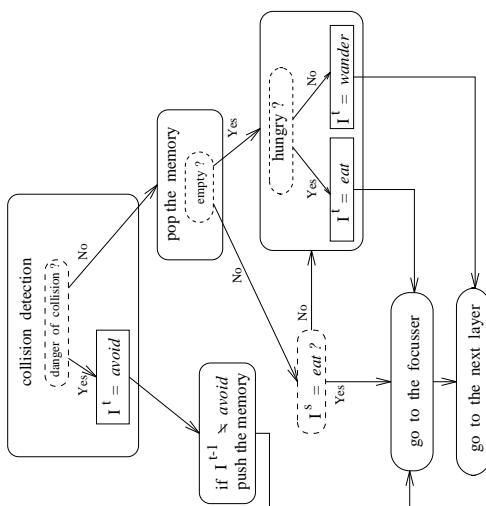


Figure 13: The intention generator of a Predator

Each fish constantly adjusts its speed and direction to match those of other members of the school. They establish a certain distance from one another, roughly one body length from neighbors, on average [25]. Each member of a school of artificial fish acts autonomously, and the schooling behavior is achieved through sensory perception and locomotion. An inceptive school is formed when a few fish swim towards a lead fish. Once a fish is in some proximity to some other schooling fish, the *schooling* behavior routine outlined in Fig. 16 is invoked.

The intention generator prevents schooling fish from getting too close together, because the *avoid* collision intention has highest precedence. To create more compact schools, the collision sensitivity region of a schooling fish is decreased, once it gets into formation. When a large school encounters an obstacle, the autonomous behavior of individual fishes trying to avoid the obstacle may cause the school to split into two groups and rejoin once the obstacle is cleared and the *schooling* behavior routine regains control (Fig. 17).

### 7.3 Pacifists

The intention generator of a pacifist differs from that of prey in that intention *mate* is activated and *escape* and *school* are deactivated. Piscine mating behaviors show great interspecies and intraspecies diversity [21]. However, two behaviors are prevalent: (i) nuzzling, where typically the male approaches the female from underneath and nudges her abdomen repeatedly until she is ready to spawn, and (ii) spawning ascent, where in its simplest form, the female rapidly swims towards the surface pursued by the male and releases gametes at the peak of her ascent. Moreover, courtship dancing is common in many species, albeit with substantial variation. Two frequently observed patterns are looping, in which the male and female circle, seemingly chasing each other's tail.

We have implemented a reasonably elaborate *courtship* behavior routine which simulates courtship dancing, circling, spawning ascent, and nuzzling behavior patterns in sequence (Plate 1b). A male fish selects a mating partner based on the following criteria: a female of the same species is more attractive than one of different species, and closer females are more attractive than ones further away. A female selects a partner similarly, but shows preference over the size of the male fish (stronger, more protective) rather than its distance.



Figure 14: Predator ingesting prey.

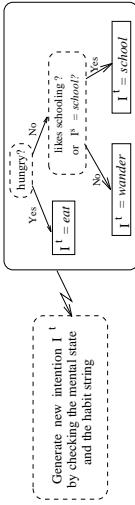


Figure 15: Modified portion of intention generator for prey.

Once fish  $i$  has selected a potential partner  $j$  based on the above criteria, it sends a signal to fish  $j$ , and there are three possibilities: Case 1: If fish  $j$ 's intention is not to *mate*, fish  $i$  approaches  $j$  and follows it around using *chasing-target* with the center of  $j$ 's body as target. Case 2: If fish  $j$ 's intention is to *mate* but its intended partner is not fish  $i$ . In this case, if  $i$  is male it will perform a *looping* behavior in front of  $j$  for a certain amount of time. If  $j$  is impressed and selects  $i$  during this time limit, then the courtship sequence continues, otherwise  $i$  will discontinue *looping* and leave  $j$  to find a new potential partner. Otherwise, if  $i$  is female it will choose another potential male. Case 3: If fish  $j$ 's intention is to *mate* and its intended partner is fish  $i$ , the *courtship* behavior starts with the male looping in front of the female while she hovers and bobs her head. Looping is simulated by invoking *chasing-target* at a point in front of the female's head which moves up and down at a certain frequency. The female's hovering and head bobbing is accomplished through motor control of her pectoral fins (i.e., parameter  $\gamma$  in Fig. 5).

The male counts the number of times his mouth reaches the vicinity of the moving point, and when the count exceeds a set threshold (currently 6) he makes a transition from *looping* to *circling* behavior. Although the threshold count is fixed, the actual motions and duration of looping is highly unpredictable for any number of reasons, including the fact that looping may be temporarily interrupted to handle high priority events such as potential collisions between the pair or with other fishes that may pass by. Before the transition to *circling*, the female fish may reject her initial partner and turn to a new larger male fish if the latter joins in the *looping* display. At this point the initially engaged male turns away as in case 2 described above. *Circling* is achieved when the fishes use *chasing-target* to chase each other's tail. The *circling* routine ends and the *spawning ascending* routine begins after the female has made a fixed number of turns during *circling*. The female fish ascends quickly through fast swimming followed by hovering. The male

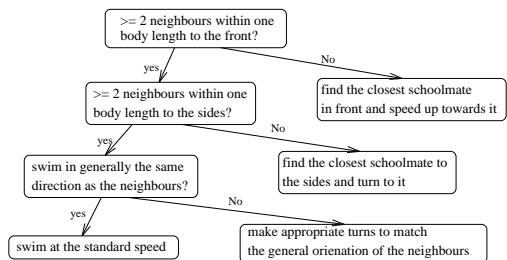


Figure 16: Schooling behavior routine.

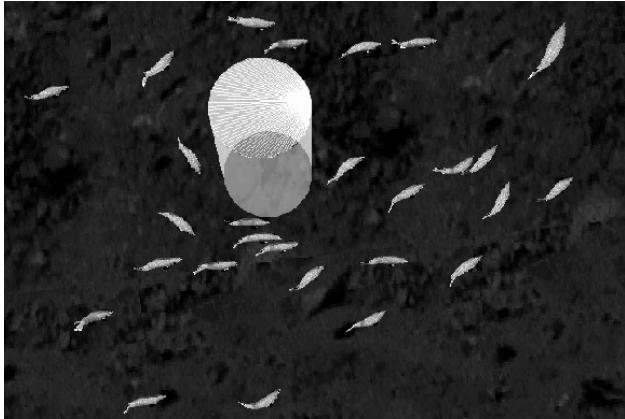


Figure 17: School of fish swimming past cylindrical obstacle (from lower left to upper right). Schooling behavior is interrupted by collision avoidance behavior and then resumed.

fish uses *chasing-target* to follow the abdomen of the female. The *nuzzling* routine requires the male to approach her abdomen from below. Once his mouth touches her abdomen, the male backs off for a number of time steps. This procedure repeats, until the male successfully touches the female a predetermined number of times (e.g., 3). To permit the mating pair to come close together, the regions of sensitivity are set very tightly to their bodies. It is intriguing to watch some of the male artificial fish's nuzzling attempts fail because of an inappropriate approach angle to the female which triggers the *avoiding-fish* response. The male turns away to avoid the collision and tries again.

## 8 Conclusion and Research Directions

This paper has presented the results of research spanning the fields of artificial life and computer graphics, with anticipated future implications for computer vision. We have developed a physics-based, virtual marine world inhabited by astonishingly lifelike artificial life forms that emulate the appearance, motion, and behavior of fishes in their natural habitats. Each artificial fish is an autonomous agent with a deformable body actuated by internal muscles, eyes, and a mind that includes learning, behavior, perception, and motor centers. Through controlled muscle

actions, artificial fishes are able to swim through simulated water in accordance with simplified hydrodynamics. Their functional fins enable them to locomote, maintain balance, and maneuver in the water. Though rudimentary compared to real animals, their minds are nonetheless able to learn some basic motor functions and carry out perceptually guided motor tasks. In accordance with their perceptual awareness of the virtual world, their minds arbitrate a repertoire of piscine behaviors, including collision avoidance, foraging, preying, schooling, and mating. The easy extensibility of our approach is suggested most evidently by the complex patterns of mating behavior that we have been able to implement in artificial fishes.

Our model achieves a good compromise between realism and computational efficiency. To give an example simulation rate, our implementation can simulate a scenario with 10 fishes, 15 food particles, and 5 static obstacles at about 4 frames/sec, including wireframe rendering time, on a Silicon Graphics R4400 Indigo<sup>2</sup> Extreme workstation. More complex scenarios with large schools of fish, dynamic plants, and full color texture mapped rendering at video resolution can take 5 seconds or more per frame.

Our work opens up many avenues of research. Clearly the artificial fish is a virtual robot that offers a much broader range of perceptual and animate capabilities, lower cost, and higher reliability than can be expected from present-day physical robots like those described in [13]. For at least these reasons, artificial fishes in their dynamic world can serve as a proving ground for theories that profess competence at effectively linking perception to action [3]. A different research direction would address the goals of researchers interested in evolving artificial life. We may be within reach of computational models that can imitate the spawning behaviors of the female fish and fertilization by the male. Through simulated sexual reproduction in a competitive world, gametes representing artificial fish genotypes can be fused to evolve new varieties of artificial fishes. Interestingly, Pokhilko, Pajitnov, *et al.*, have already demonstrated the simulated breeding of fish models much simpler than ours using genetic algorithms, and this idea has resulted in the fascinating computer game "El-Fish" [9].

## Acknowledgements

We thank Eugene Fiume, Michael McCool, Michiel van de Panne, Sarah Peebles, and John Funge for discussions and for assistance with the computer animations. This research has been made possible by a grant from the Natural Sciences and Engineering Research Council of Canada and the support of the Canadian Institute for Advanced Research.

## References

- [1] H. E. Adler. *Fish Behavior: Why Fishes do What They Do*. T.F.H Publications, Neptune City, NJ, 1975.
- [2] R.M. Alexander. *Exploring Biomechanics*. Scientific American Library, New York, 1992.
- [3] D. Ballard. Animate vision. *Artificial Intelligence*, 48:57–86, 1991.
- [4] H. G. Barrow and J. M. Tenenbaum. Recovering intrinsic scene characteristics from images. In E. Riseman and A. Hanson, editors, *Computer Vision Systems*, pages 3–26. Academic Press, NY, 1978.
- [5] R. Beer. *Intelligence as Adaptive Behavior*. Academic press, NY, 1990.
- [6] R. W. Blake. *Fish Locomotion*. Cambridge University Press, Cambridge, England, 1983.
- [7] V. Braintenberg. *Vehicles, Experiments in Synthetic Psychology*. MIT Press, Cambridge, MA, 1984.
- [8] R. A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6(1):3–15, 1990.
- [9] E. Corcoran. One fish, two fish: How to raise a school of tempting software toys. *Scientific American*, July 1992.
- [10] B. K. P. Horn. *Robot Vision*. MIT Press, Cambridge, MA, 1986.

- [11] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *Int. J. of Computer Vision*, 1(4):321–331, 1987.
- [12] K. Lorenz. *Foundations of Ethology*. Springer-Verlag, New York, 1973.
- [13] P. Maes, editor. *Designing Autonomous Agents*. MIT Press, Cambridge, MA, 1991.
- [14] M. J. Matarić. *Interaction and Intelligent Behavior*. PhD thesis, Dept. of EECS, MIT, Cambridge, MA, May 1994.
- [15] D. McFarland. *Animal Behaviour*. Pitman, 1985.
- [16] J.-A. Meyer and A. Guillot. Simulation of adaptive behavior in animats: Review and prospect. In J.-A. Meyer and S. Wilson, editors, *From Animals to Animats*, pages 2–14. MIT Press, Cambridge, MA, 1991.
- [17] G. S. P. Miller. The motion dynamics of snakes and worms. *Computer Graphics*, 22(4):169–177, 1988.
- [18] K. G. Pearson. Sensory elements in pattern-generating networks. In *Making Them Move*, pages 111–127. Morgan Kaufmann, San Mateo, California, 1991.
- [19] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, Cambridge, England, 1986.
- [20] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.
- [21] R. E. Thresher. *Reproduction in Reef Fishes*. TFH Publications, Neptune City, NJ, 1984.
- [22] N. Tinbergen. *The Study of Instinct*. Clarendon Press, Oxford, England, 1950.
- [23] X. Tu, D. Terzopoulos, and E. Fiume. Go Fish! ACM SIGGRAPH Video Review Issue 91: SIGGRAPH'93 Electronic Theater, 1993.
- [24] P. W. Webb. Form and function in fish swimming. *Scientific American*, 251(1), 1989.
- [25] R. Wilson and J. Q. Wilson. *Watching Fishes*. Harper and Row, New York, 1985.
- [26] S. W. Wilson. The animat path to AI. In J.-A. Meyer and S. Wilson, editors, *From Animals to Animats*, pages 15–21. MIT Press, Cambridge, MA, 1991.

# Perception and Learning in Artificial Animals

**Demetri Terzopoulos, Tamer Rabie and Radek Grzeszczuk**

Department of Computer Science, University of Toronto  
6 King's College Road, Toronto, ON M5S 3H5, Canada

e-mail: {dt | tamer | radek}@cs.toronto.edu

## Abstract

We employ a virtual marine world inhabited by realistic artificial animals as an ALife laboratory for developing and evaluating zoomimetic perception and learning algorithms. In particular, we propose active perception strategies that enable artificial marine animals to navigate purposefully through their world by using computer vision algorithms to analyze the foveated retinal image streams acquired by their eyes. We also demonstrate learning algorithms that enable artificial marine animals to acquire complex motor skills similar to those displayed by trained marine mammals at aquatic theme parks.

## 1 Introduction

A recent result of artificial life research is a virtual world inhabited by artificial animals and plants that emulate some of the fauna and flora of natural marine environments [1]. In this paper, we employ this highly realistic virtual world as an artificial zoological laboratory. The laboratory facilitates the investigation of open problems related to biological information processing in animals, and it has enabled us to develop and evaluate zoomimetic perception and learning algorithms.

The psychologist J.J. Gibson studied (in pre-computational terms) the perceptual problems faced by an active observer situated in the dynamic environment [2].<sup>1</sup> We present a prototype active perception system that enables artificial marine animals to navigate purposefully through their world by analyzing the retinal image streams acquired by their eyes. Retinal image analysis is carried out using computer vision algorithms. We equip our artificial animals with directable, virtual eyes capable of foveal vision. This aspect of our work is related to that of Cliff and Bullock [5], but our realistic animal models have enabled us to progress a great deal further.<sup>2</sup> Our

<sup>1</sup>Computational versions of Gibson's paradigm were developed in computer vision by Bajcsy [3] and Ballard [4] under the names of "active perception" and "animate vision", respectively.

<sup>2</sup>Cliff and Bullock [5] were concerned with the evolution of simple visually guided behaviors using Wilson's animat in a discrete 2D grid world.



Figure 1: *Artificial fishes swimming among aquatic plants in a physics-based virtual marine environment.*

goal is to engineer general-purpose vision systems for artificial animals possessing zoomimetic eyes that image continuous 3D photorealistic worlds. We assemble a suite of vision algorithms that support foveation, retinal image stabilization, color object recognition, and perceptually-guided navigation. These perceptual capabilities allow our artificial fishes to pursue moving targets, such as fellow fishes. They do so by cascading their eyes to maintain foveation on targets as they control their muscle-actuated bodies to locomote in the direction of their gaze.

We also demonstrate motor learning algorithms that enable artificial marine animals to acquire some nontrivial motor skills through practice. In particular, these algorithms enable an artificial dolphin to learn to execute stunts not unlike those performed by trained marine mammals to the delight of spectators at aquatic theme parks. This research builds upon the low-level motor learning algorithms described in our prior work [1]. It reinforces our earlier claim that biomechanical models of animals situated in physics-based worlds are fertile ground for learning novel sensorimotor control strategies.

## 2 Review of Artificial Fishes

Artificial fishes are autonomous agents inhabiting a realistic, physics-based virtual marine world (Fig. 1). Each agent has a

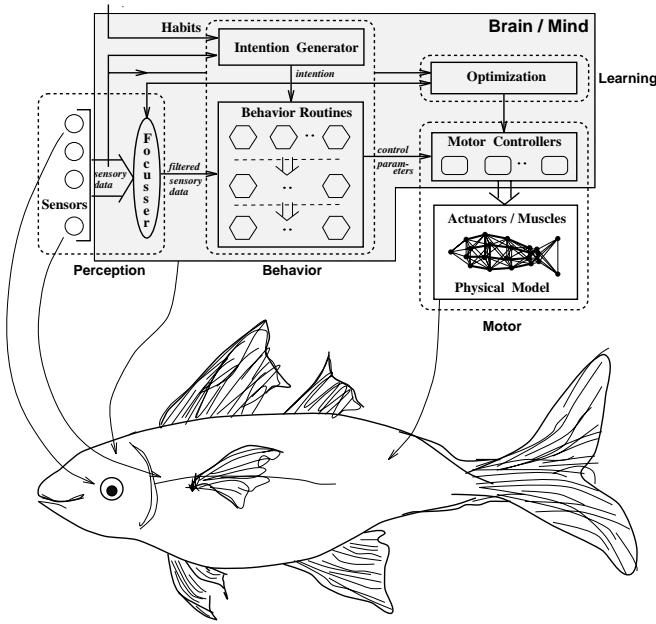


Figure 2: Artificial fish model (from [1]).

deformable body actuated by internal muscles. The body also harbors eyes and a brain with motor, perception, behavior, and learning centers (Fig. 2). Through controlled muscle actions, artificial fishes are able to swim through simulated water in accordance with hydrodynamic principles. Their functional fins enable them to locomote, maintain balance, and maneuver in the water. Thus the model captures not just the form and appearance of the animal, but also the basic physics of the animal in its environment. Although rudimentary compared to those of real animals, the brains of artificial fishes are nonetheless able to learn some basic motor functions and carry out perceptually guided motor tasks. The behavior center of the artificial fish's brain mediates between its perception system and its motor system, harnessing the dynamics of the perception-action cycle. The innate character of the fish is determined by fixed habits. Its dynamic mental state is represented by a set of mental variables—hunger, libido, and fear. An intention generator serves as the fish's cognitive faculty, arbitrating the artificial fish's behavioral repertoire in accordance with its perceptual awareness of the virtual world. The behavioral repertoire includes primitive, reflexive behavior routines, such as collision avoidance, as well as more sophisticated motivational behavior routines such as foraging, preying, schooling, and mating.

The details of the artificial fish model are presented in the paper [1] (or see an earlier version in the ALIFE IV Proceedings). The remainder of this section covers details about the motor system which are necessary to understand the learning and vision algorithms to follow.

The motor system comprises the dynamic model of the fish including its muscle actuators and a set of motor controllers (MCs). Fig. 3 illustrates the biomechanical body model which

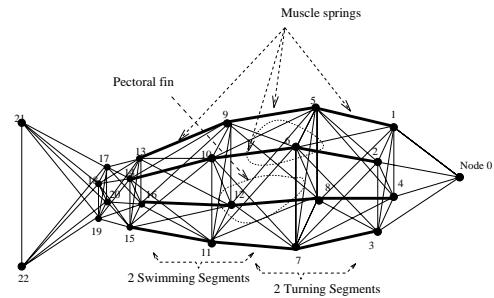


Figure 3: Biomechanical fish model. Nodes denote lumped masses. Lines indicate uniaxial elastic elements (shown at natural length). Bold lines indicate muscle elements.

produces realistic piscine locomotion using only 23 lumped masses and 91 elastic elements. These mechanical components are interconnected so as to maintain the structural integrity of the body as it flexes due to the action of its 12 contractile muscles.

Artificial fishes locomote like real fishes, by autonomously contracting their muscles. As the body flexes it displaces virtual fluid which induces local reaction forces normal to the body. These hydrodynamic forces generate thrust that propels the fish forward. The model mechanics are governed by Lagrange equations of motion driven by the hydrodynamic forces. The system of coupled second-order ordinary differential equations are continually integrated through time by a numerical simulator.<sup>3</sup>

The model is sufficiently rich to enable the design of motor controllers by gleaning information from the fish biomechanics literature. The motor controllers coordinate muscle actions to carry out specific motor functions, such as swimming forward (swim-MC), turning left (left-turn-MC), and turning right (right-turn-MC). They translate natural control parameters such as the forward speed or angle of the turn into detailed muscle actions that execute the function. The artificial fish is neutrally buoyant in the virtual water and has a pair of pectoral fins that enable it to navigate freely in its 3D aquatic world by pitching, rolling, and yawing its body. Additional motor controllers coordinate the fin actions.

### 3 Perception

This section describes a vision system for artificial fish which is based solely on retinal image analysis via computer vision algorithms [6].<sup>4</sup> We have developed a prototype active

<sup>3</sup>The artificial fish model achieves a good compromise between realism and computational efficiency. For example, the implementation can simulate a scenario with 10 fishes, 15 food particles, and 5 static obstacles at about 4 frames/sec (with wireframe rendering) on a Silicon Graphics R4400 Indigo<sup>2</sup> Extreme workstation. More complex scenarios with large schools of fish, dynamic plants, and full color texture mapped GL rendering at video resolution can take 5 seconds or more per frame.

<sup>4</sup>By contrast, in our prior work [1] the artificial fishes rely on simulated perception—a “perceptual oracle” which satisfies the fish’s sensory needs by directly interrogating the 3D world model; i.e., the

vision system for our artificial animals. The system is designed for extensibility, so that it can eventually support the broad repertoire of individual and group behaviors of artificial fishes. However, our approach to perception applies to any animal, not just fishes. In fact, we view artificial fishes as virtual piscine robots, and we do not restrict ourselves to modeling the perceptual mechanisms of real fishes [7]. Indeed, it will soon become evident that our piscine robots sense their world through virtual eyes that are patterned after those of primates!

### 3.1 Active Vision System

The basic functionality of the active vision system starts with binocular perspective projection of the color 3D world onto the 2D retinas of the artificial fish. Retinal imaging is accomplished by photorealistic graphics rendering of the world from the animal's point of view. This projection respects occlusion relationships among objects. It forms spatially nonuniform visual fields with high resolution foveas and low resolution peripheries. Based on an analysis of the incoming color retinal image stream, the perception center of the artificial fish's brain supplies saccade control signals to its eyes and stabilize the visual fields during locomotion, to attend to interesting targets based on color, and to keep targets fixated. The artificial fish is thus able to approach and track other artificial fishes using sensorimotor control.

Fig. 4 is a block diagram of the active vision system showing two main modules that control foveation of the eyes and retinal image stabilization.

**Eyes and Foveated Retinal Imaging** The artificial fish is capable of binocular vision and possesses an ocular motor system that controls eye movements [8]. The movements of each eye are controlled through two gaze angles ( $\theta, \phi$ ) which specify the horizontal and vertical rotation of the eyeball, respectively, with respect to the head coordinate frame (when  $\theta = \phi = 0^\circ$ , the eye looks straight ahead).

Each eye is implemented as four coaxial virtual cameras to approximate the spatially nonuniform, foveal/peripheral imaging capabilities typical of biological eyes. Fig. 5(a) shows an example of the  $64 \times 64$  images that are rendered (using the GL library and SGI graphics pipeline) by the four coaxial cameras of the left and right eye. The level  $l = 0$  camera has the widest field of view (about  $120^\circ$ ). The field of view decreases with increasing  $l$ . The highest resolution image at level  $l = 3$  is the fovea and the other images form the visual periphery. Fig. 5(b) shows the  $512 \times 512$  binocular retinal images composited from the coaxial images at the top of the figure (the component images are expanded by factors  $2^{l-3}$ ). To reveal the retinal image structure in the figure, we have placed a white border around each magnified component image. Significant computational efficiency accrues from processing four  $64 \times 64$  component images rather than

autonomous agents were permitted direct access to the geometric and photometric information available to the graphics rendering engine, as well as object identity and dynamic state information about the physics-based world model.

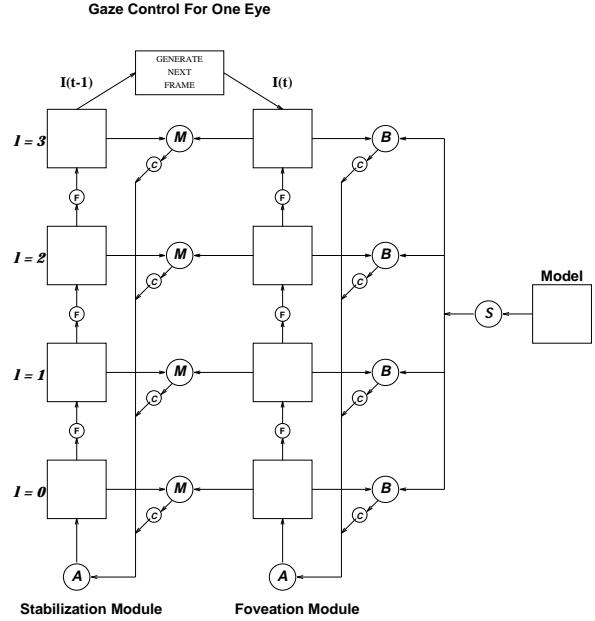


Figure 4: The active vision system. The flow of the algorithm is from right to left. A: Update gaze angles ( $\theta, \phi$ ) and saccade using these angles, B: Search current level for model target and if found localize it, else search lower level, C: Select level to be processed (see text), F: Reduce field of view for next level and render, M: Compute a general translational displacement vector ( $u, v$ ) between images  $I(t - 1)$  and  $I(t)$ , S: Scale the color histogram of the model for use by the current level.

a uniform  $512 \times 512$  retinal image.

**Foveation by Color Object Detection** The brain of the fish stores a set of color models of objects that are of interest to it. For instance, if the fish is a predator, it would possess mental models of prey fish. The models are stored as a list of  $64 \times 64$  RGB color images in the fish's visual memory.

To detect and localize any target that may be imaged in the low resolution periphery of its retinas, the active vision system of the fish employs an improved version of a color indexing algorithm proposed by Swain [9]. Since each model object has a unique color histogram signature, it can be detected in the retinal image by histogram intersection and localized by histogram backprojection. Our algorithms are explained more fully in [10].

**Saccadic Eye Movements** When a target is detected in the visual periphery, the eyes will saccade to the angular offset of the object to bring it within the fovea. With the object in the high resolution fovea, a more accurate foveation is obtained by a second pass of histogram backprojection. A second saccade typically centers the object accurately in both left and right foveas, thus achieving vergence.

Module A in Fig. 4 performs the saccades by incrementing the gaze angles ( $\theta, \phi$ ) in order to rotate the eyes to achieve the required gaze direction.

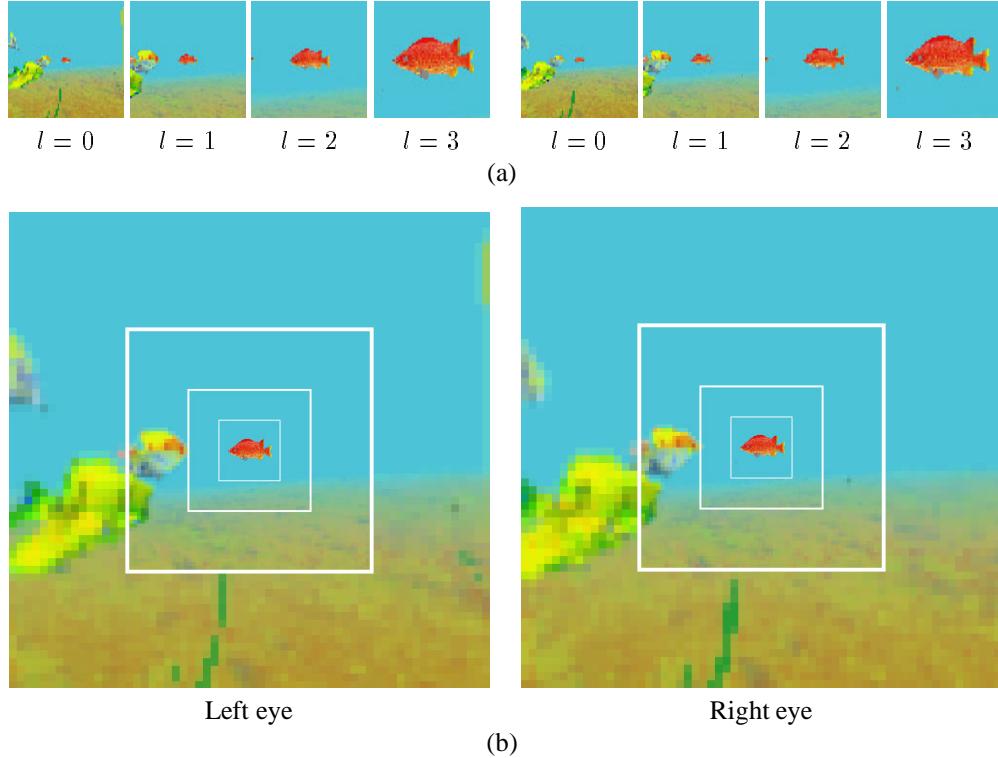


Figure 5: *Binocular retinal imaging (monochrome versions of original color images). (a)* 4 component images;  $l = 0, 1, 2$ , are peripheral images;  $l = 3$  is foveal image. *(b)* Composited retinal images (borders of composited component images are shown in white).

**Visual Field Stabilization using Optical Flow** It is necessary to stabilize the visual field of the artificial fish because its body undulates as it swims. Once a target is verged in both foveas, the stabilization process (Fig. 4) assumes the task of keeping the target foveated as the fish locomotes. Thus, it emulates the optokinetic reflex in animals.

Stabilization is achieved by computing the overall translational displacement  $(u, v)$  of light patterns between the current foveal image and that from the previous time instant, and updating the gaze angles to compensate. The displacement is computed as a translational offset in the retinotopic coordinate system by a least squares minimization of the optical flow between image frames at times  $t$  and  $t - 1$  [6].

The optical flow stabilization method is robust only for small displacements between frames. Consequently, when the displacement of the target between frames is large enough that the method is likely to produce bad estimates, the foveation module is invoked to re-detect and re-foveate the target as described earlier.

Each eye is controlled independently during foveation and stabilization of a target. Hence, the two retinal images must be correlated to keep them verged accurately on the target. Referring to Fig. 6, the vergence angle is  $\theta_V = (\theta_R - \theta_L)$  and its magnitude increases as the fish comes closer to the target. Therefore, once the eyes are verged on a target, it is straightforward for the fish vision system to estimate the range to the

target by triangulation using the gaze angles.

### 3.2 Vision-Guided Navigation

The fish can use the gaze direction for the purposes of navigation in its world. In particular, it is natural to use the gaze angles as the eyes are fixated on a target to navigate towards the target. The  $\theta$  angles are used to compute the left/right turn angle  $\theta_P$  shown in Fig. 6, and the  $\phi$  angles are similarly used to compute an up/down turn angle  $\phi_P$ . The fish's turn motor controllers (see Section 2) are invoked to execute a left/right turn—left-turn-MC for an above-threshold positive  $\theta_P$  and right-turn-MC for negative  $\theta_P$ —with  $|\theta_P|$  as parameter. Up/down turn motor commands are issued to the fish's pectoral fins, with an above-threshold positive  $\phi_P$  interpreted as “up” and negative as “down”.

The problem of pursuing a moving target that has been fixated in the foveas of the fish's eyes is simplified by the gaze control mechanism described above. The fish can robustly track a target in its fovea and locomote to follow it around the environment by using the turn angles  $(\theta_P, \phi_P)$  computed from the gaze angles that are continuously updated by the foveation/stabilization algorithms.

We have carried out numerous experiments in which the moving target is a reddish prey fish whose color histogram model is stored in the memory of a predator fish equipped with the active vision system. Fig. 7 shows plots of the

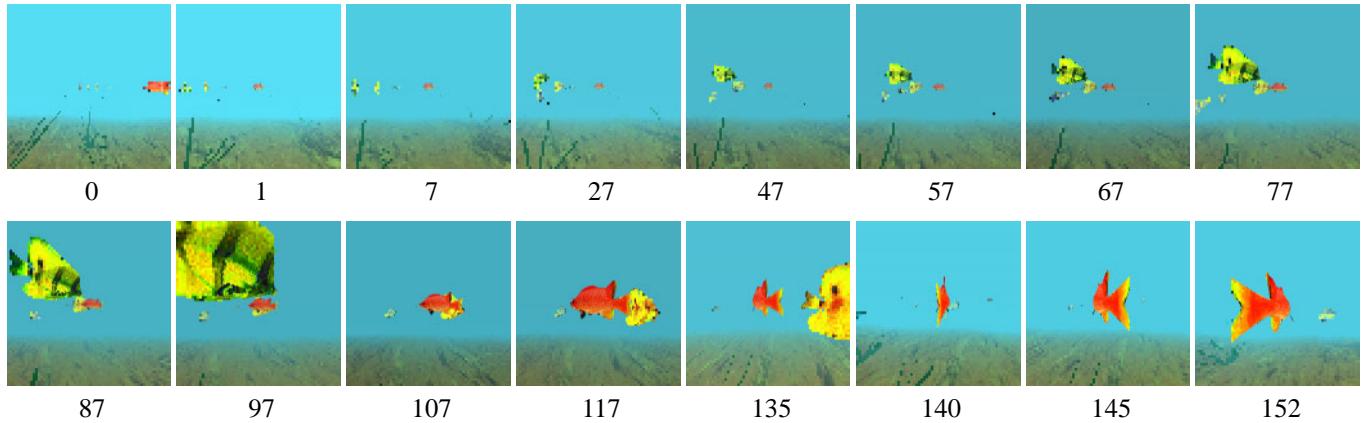


Figure 8: *Retinal image sequence from the left eye of the active vision fish as it detects and foveates on a reddish fish target and swims in pursuit of the target (monochrome versions of original color images). The target appears in the periphery (middle right) in frame 0 and is foveated in frame 1. The target remains fixated in the center of the fovea as the fish uses the gaze direction to swim towards it (frames 7–117). The target fish turns and swims away with the observer fish in visually guided pursuit (frames 135–152).*

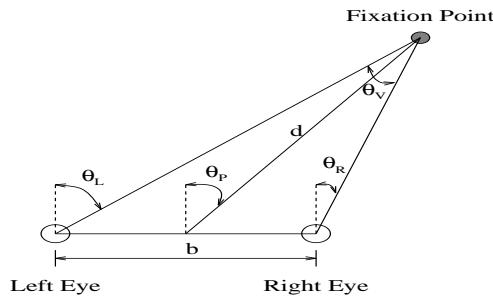


Figure 6: *Gaze angles and range to target geometry.*

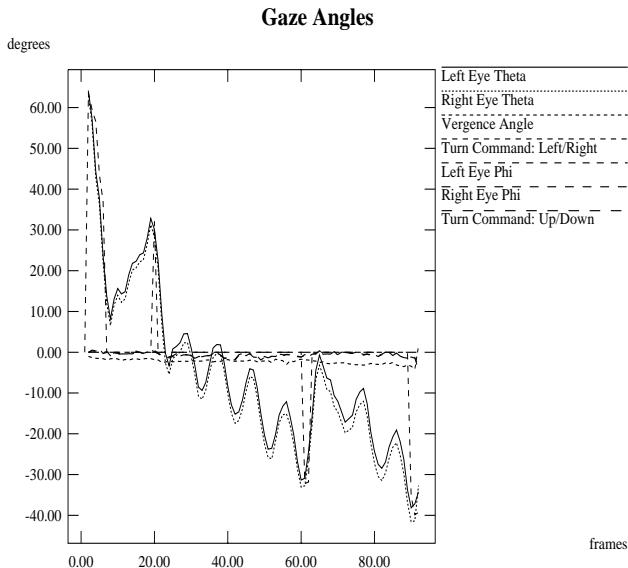


Figure 7: *Gaze angles resulting from the pursuit of a target by the AV fish.*

gaze angles and the turn angles obtained over the course of 100 frames in a typical experiment as the predator is fixated upon and actively pursuing a prey target. Fig. 8 shows a sequence of image frames acquired by the fish during its navigation (monochrome versions of only the left retinal images are shown). Frame 0 shows the target visible in the low resolution periphery of the fish's eyes (middle right). Frame 1 shows the view after the target has been detected and the eyes have performed a saccade to foveate the target (the scale difference of the target after foveation is due to perspective distortion). The subsequent frames show the target remaining fixated in the fovea despite the side-to-side motion of the fish's body as it swims towards the target.

The saccade signals that keep the predator's eyes fixated on its prey as both are swimming are reflected by the undulatory responses of the gaze angles in Fig. 7. The figure also shows that the vergence angle increases as the predator approaches its target (near frame 100). In comparison to the  $\theta$  angles, the  $\phi$  angles show little variation, because the fish does not undulate vertically very much as it swims forward. It is apparent from the graphs that the gaze directions of the two eyes are well correlated.

Note that in frames 87–117 of Fig. 8, a yellow fish whose size is similar to the target fish passes behind the target. In this experiment the predator was programmed to be totally disinterested in and not bother to foveate any non-reddish objects. Because of the color difference, the yellowish object does not distract the fish's gaze from its reddish target. This demonstrates the robustness of the color-based fixation algorithm.

#### 4 Learning

The learning center of its brain (see Fig. 2) enables the artificial fish to acquire effective locomotion skills through practice and sensory reinforcement. Our second challenge has been to enhance the algorithms comprising the artificial fish's learn-

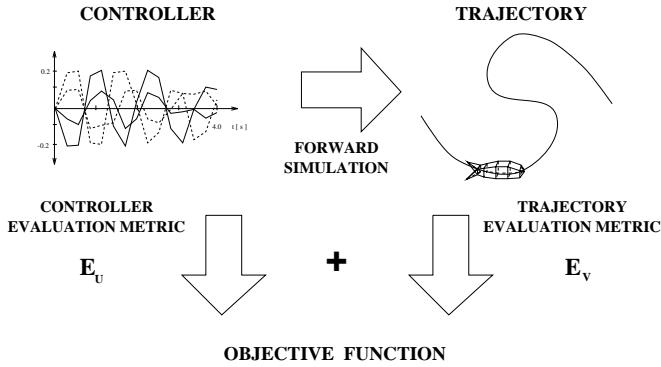


Figure 9: The objective function that guides the learning process is a weighted sum of terms that evaluate the controller and the trajectory.

ing center so that it can learn more complex motor skills than those we demonstrated in reference [1].

#### 4.1 Low-Level Motor Learning

Recall that some of the deformable elements in the biomechanical model (Fig. 3) play the role of contractile *muscles* whose natural length decreases under the autonomous control of the motor center of the artificial animal's brain. To dynamically contract a muscle, the brain must supply an *activation function*  $a(t)$  to the muscle. This continuous time function has range  $[0, 1]$ , with 0 corresponding to a fully relaxed muscle and 1 to a fully contracted muscle. Typically, individual muscles form muscle groups, called *actuators*, that are activated in unison. Referring to Fig. 3, the artificial fish has 12 muscles which are grouped pairwise in each segment to form 3 left actuators and 3 right actuators. Each actuator  $i$  is activated by a scalar *actuation function*  $u_i(t)$ , whose range is again normalized to  $[0, 1]$ , thus translating straightforwardly into activation functions for each muscle in the actuator. Thus, to control the fish's body we must specify the actuation functions  $\mathbf{u}(t) = [u_1(t), \dots, u_i(t), \dots, u_N(t)]'$ , where  $N = 6$ . The continuous vector-valued function of time  $\mathbf{u}(t)$  is called the *controller* and its job is to produce locomotion. Learned controllers may be stored within the artificial animal's motor control center.

A continuous *objective functional*  $E$  provides a quantitative measure of the progress of the locomotion learning process. The functional is the weighted sum of a term  $E_u$  that evaluates the controller  $\mathbf{u}(t)$  and a term  $E_v$  that evaluates the motion  $\mathbf{v}(t)$  that the controller produces in a time interval  $t_0 \leq t \leq t_1$ , with smaller values of  $E$  indicating better controllers  $\mathbf{u}$ . Mathematically,

$$E(\mathbf{u}(t)) = \int_{t_0}^{t_1} (\mu_1 E_u(\mathbf{u}(t)) + \mu_2 E_v(\mathbf{v}(t))) dt, \quad (1)$$

where  $\mu_1$  and  $\mu_2$  are scalar weights. Fig. 9 illustrates this schematically.

It is important to note that the complexity of our models precludes the closed-form evaluation of  $E$ . As Fig. 9 indicates, to

compute  $E$ , the artificial animal must first invoke a controller  $\mathbf{u}(t)$  to produce a motion  $\mathbf{v}(t)$  with its body (in order to evaluate term  $E_v$ ). This is done through forward simulation of the biomechanical model over the time interval  $t_0 \leq t \leq t_1$  with controller  $\mathbf{u}(t)$ .

We may want to promote a preference for controllers with certain qualities via the controller evaluation term  $E_u$ . For example, we can guide the optimization of  $E$  by discouraging large, rapid fluctuations of  $\mathbf{u}$ , since chaotic actuations are usually energetically inefficient. We encourage lower amplitude, smoother controllers through the function  $E_u = (\nu_1 |d\mathbf{u}/dt|^2 + \nu_2 |d^2\mathbf{u}/dt^2|^2)/2$ , where the weighting factors  $\nu_1$  and  $\nu_2$  penalize actuation amplitudes and actuation variation, respectively. The distinction between good and bad controllers also depends on the goals that the animal must accomplish. In our learning experiments we used trajectory criteria  $E_v$  such as the final distance to the goal, the deviation from a desired speed, etc. These and other criteria will be discussed shortly in conjunction with specific experiments.

The low level motor learning problem optimizes the objective functional (1). This cannot be done analytically. We convert the continuous optimization problem to an algebraic parameter optimization problem [11] by parameterizing the controller through discretization using basis functions. Mathematically, we express  $u_i(t) = \sum_{j=1}^M u_i^j B^j(t)$ , where the  $u_i^j$  are scalar parameters and the  $B^j(t)$ ,  $1 \leq j \leq M$  are (vector-valued) temporal basis functions. The simplest case is when the  $u_i^j$  are evenly distributed in the time interval and the  $B^j(t)$  are tent functions centered on the nodes with support extending to nearest neighbor nodes, so that  $\mathbf{u}(t)$  is the linear interpolation of the nodal variables.

Since  $\mathbf{u}(t)$  has  $N$  basis functions, the discretized controller is represented using  $NM$  parameters. Substituting the above equation into the continuous objective functional (1), we approximate it by the discrete *objective function*  $E([u_1^1, \dots, u_N^M]')$ . Learning low level motor control amounts to using an optimization algorithm to iteratively update the parameters so as to optimize the discrete objective function and produce increasingly better locomotion.

We use the simulated annealing method to optimize the objective function [12]. Simulated annealing has three features that make it particularly suitable for our application. First, it is applicable to problems with a large number of variables yielding search spaces large enough to make exhaustive search prohibitive. Second, it does not require gradient information about the objective function. Analytic gradients are not directly attainable in our situation since evaluating  $E$  requires a forward dynamic simulation. Third, it avoids getting trapped in local suboptima of  $E$ . In fact, given a sufficiently slow annealing schedule, it will find a global optimum of the objective functional. Robustness against local suboptima can be important in obtaining muscle control functions that produce realistic motion.

In summary, the motor learning algorithms discover muscle controllers that produce efficient locomotion through op-

timization. Muscle contractions that produce forward movements are “remembered”. These partial successes then form the basis for the fish’s subsequent improvement in its swimming technique. Their brain’s learning center also enable these artificial animals to train themselves to accomplish higher level sensorimotor tasks, such as maneuvering to reach a visible target (see [1] for the details).

## 4.2 Learning Complex Skills

**Abstracting Controllers** It is time consuming to learn a good solution for a low level controller because of the high dimensionality of the problem (large  $N M$ ), the lack of gradient information to accelerate the optimization of the objective functional, and the presence of suboptimal traps that must be avoided. For tractability, the learning procedure must be able to abstract compact higher level controllers from the low level controllers that have been learned, retain the abstracted controllers, and apply them to future locomotion tasks.

The process of abstraction takes the form of a dimensionality reducing change of representation. More specifically, it seeks to compress the many parameters of the discrete controllers to a compact form in terms of a handful of basis functions. Natural, steady-state locomotion patterns tend to be quasi-periodic and they can be abstracted very effectively without substantial loss. A natural approach to abstracting low-level motor controllers is to apply the fast Fourier transform (FFT) [12] to the parameters of the controller and then suppress the below-threshold amplitudes.

Typically, our artificial animals are put through a “basic training” regimen of primitive motor tasks that it must learn, such as locomoting at different speeds and executing turns of different radii. They learn effective low level controllers for each task and retain compact representations of these controllers through controller abstraction. The animals subsequently put the abstractions that they have learned into practice to accomplish higher level tasks, such as target tracking or leaping through the air. To this end, abstracted controllers are concatenated in sequence, with each controller slightly overlapping the next. To eliminate discontinuities, temporally adjacent controllers are smoothly blended together by linearly fading and summing them over a small, fixed region of overlap, approximately 5% of each controller (Fig. 10).

**Composing Macro Controllers** Next the learning process discovers composite abstracted controllers that can accomplish complex locomotion tasks. Consider the spectacular stunts performed by marine mammals that elicit applause at theme parks like “SeaWorld”. We can treat a leap through the air as a complex task that can be achieved using simpler tasks; e.g., diving deep beneath a suitable leap point, surfacing vigorously to gain momentum, maintaining balance during the ballistic flight through the air, and splashing down dramatically with a belly flop.

We have developed an automatic learning technique that constructs a macro jump controller of this sort as an optimized sequence of basic abstracted controllers. The optimization

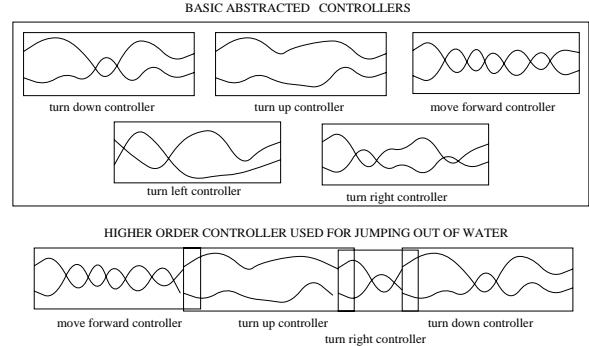


Figure 10: *Higher level controller for jumping out of water is constructed from a set of abstracted basic controllers.*

process is, in principle, similar to the one in low level learning. It uses simulated annealing for optimization, but rather than optimizing over nodal parameters or frequency parameters, it optimizes over the selection, ordering, and duration of abstracted controllers. Thus the artificial animal applying this method learns effective macro controllers of the type shown at the bottom of Fig. 10 by optimizing over a learned repertoire of basic abstracted controllers illustrated at the top of the figure.

We have trained an artificial dolphin to learn effective controllers for 5 basic motor tasks: turn-down, turn-up, turn-left, turn-right, and move-forward. We then give it the task of performing a stunt like the one described above and the dolphin discovers a combination of controllers that accomplishes the stunt. In particular, it discovers that it must build up momentum by thrusting from deep in the virtual pool of water up towards the surface and it must exploit this momentum to leap out of the water. Fig. 11(a) shows a frame as the dolphin exits the water. The dolphin can also learn to perform tricks while in the air. Fig. 11(b) shows it using its nose to bounce a large beach-ball off a support. The dolphin can learn to control the angular momentum of its body while exiting the water and while in ballistic flight so that it can perform aerial spins and somersaults. Fig. 11(c) shows it in the midst of a somersault in which it has just bounced the ball with its tail instead of its nose. Fig. 11(d) shows the dolphin right after splashdown. In this instance it has made a dramatic bellyflop splash.

## 5 Conclusion

We have demonstrated that the artificial fishes model that we developed in our prior work may be effectively employed to devise sophisticated algorithms for perception and learning. We have successfully implemented within the framework of the artificial fish a set of active vision algorithms for foveation and vergence of interesting targets, for retinal image stabilization, and for pursuit of moving targets through visually-guided navigation. Note that these vision algorithms confront synthetic retinal images that are by no means easy to analyze (compared to the sorts of images encountered in physical robotics). We have also demonstrated enhanced learning

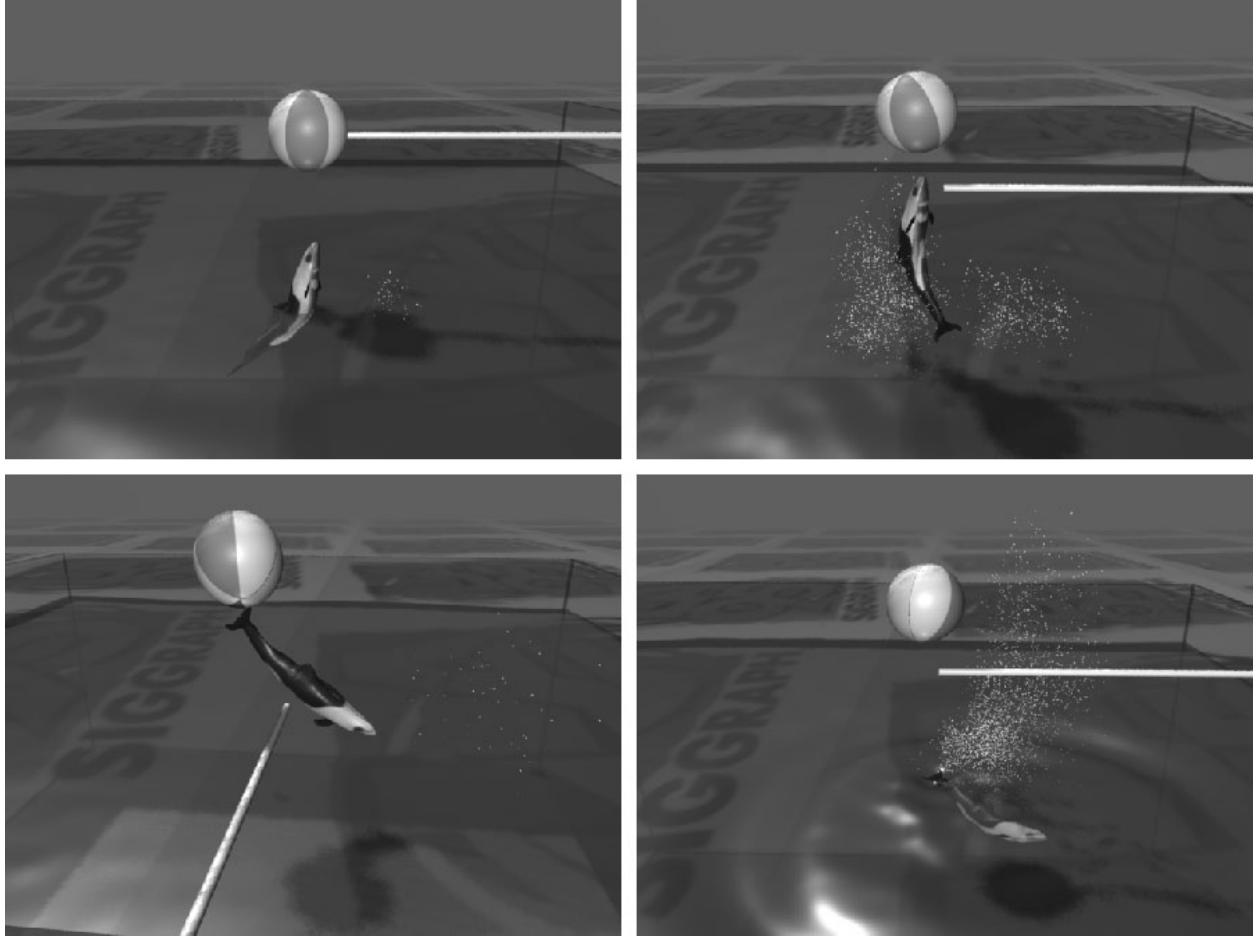


Figure 11: “*SeaWorld*” skills learned by an artificial dolphin.

algorithms that can enable an artificial marine mammal to acquire complex motor skills. These skills necessitate locomotion through water, ballistic flight through air, and a graceful style of execution. The use of highly realistic, physics-based virtual worlds inhabited by biomimetic autonomous agents appears to be a fruitful strategy for exploring difficult open problems in biological information processing and control. Our approach has value beyond artificial life to related disciplines such as vision, robotics, and virtual reality.

### Acknowledgements

We would like to thank Xiaoyuan Tu for making the research described herein possible by developing and implementing the artificial fish model. This work was supported by grants from the Natural Sciences and Engineering Research Council of Canada.

### References

- [1] D. Terzopoulos, X. Tu, and R. Grzeszczuk. Artificial fishes: Autonomous locomotion, perception, behavior, and learning in a simulated physical world. *Artificial Life*, 1(4):327–351, 1994.
- [2] J. J. Gibson. *The Ecological Approach to Visual Perception*. Houghton Mifflin, Boston, MA, 1979.
- [3] R. Bajcsy. Active perception. *Proceedings of the IEEE*, 76(8):996–1005, 1988.
- [4] D. Ballard. Animate vision. *Artificial Intelligence*, 48:57–86, 1991.
- [5] D. Cliff and S. Bullock. Adding “foveal vision” to Wilson’s animat. *Adaptive Behavior*, 2(1):49–72, 1993.
- [6] B. K. P. Horn. *Robot Vision*. MIT Press, Cambridge, MA, 1986.
- [7] R. D. Fernald. Vision. In D. H. Evans, editor, *The Physiology of Fishes*, chapter 6, pages 161–189. CRC Press, Boca Raton, FL, 1993.
- [8] M. E. Goldberg, H. M. Eggers, and P. Gouras. The ocular motor system. In E. R. Kandel, J. H. Schwartz, and T. M. Jessel, editors, *Principles of Neural Science*, pages 660–678. Elsevier, New York, NY, 1991.
- [9] M. Swain and D. Ballard. Color indexing. *Int. J. Computer Vision*, 7:11–32, 1991.
- [10] D. Terzopoulos and T.F. Rabie. Animat vision. In *Proc. Fifth International Conference on Computer Vision*, Cambridge, MA, June 1995. IEEE Computer Society Press.
- [11] C. J. Goh and K. L. Teo. Control parameterization: A unified approach to optimal control problems with general constraints. *Automatica*, 24:3–18, 1988.
- [12] W. H. Press, B. Flannery, et al. *Numerical Recipes: The Art of Scientific Computing, Second Edition*. Cambridge University Press, 1992.

# Realistic Modeling for Facial Animation

Yuencheng Lee<sup>1</sup>, Demetri Terzopoulos<sup>1</sup>, and Keith Waters<sup>2</sup>  
University of Toronto<sup>1</sup> and Digital Equipment Corporation<sup>2</sup>

## Abstract

A major unsolved problem in computer graphics is the construction and animation of realistic human facial models. Traditionally, facial models have been built painstakingly by manual digitization and animated by ad hoc parametrically controlled facial mesh deformations or kinematic approximation of muscle actions. Fortunately, animators are now able to digitize facial geometries through the use of scanning range sensors and animate them through the dynamic simulation of facial tissues and muscles. However, these techniques require considerable user input to construct facial models of individuals suitable for animation. In this paper, we present a methodology for automating this challenging task. Starting with a structured facial mesh, we develop algorithms that automatically construct functional models of the heads of human subjects from laser-scanned range and reflectance data. These algorithms automatically insert contractile muscles at anatomically correct positions within a dynamic skin model and root them in an estimated skull structure with a hinged jaw. They also synthesize functional eyes, eyelids, teeth, and a neck and fit them to the final model. The constructed face may be animated via muscle actuations. In this way, we create the most authentic and functional facial models of individuals available to date and demonstrate their use in facial animation.

**CR Categories:** I.3.5 [Computer Graphics]: Physically based modeling; I.3.7 [Computer Graphics]: Animation.

**Additional Keywords:** Physics-based Facial Modeling, Facial Animation, RGB/Range Scanners, Feature-Based Facial Adaptation, Texture Mapping, Discrete Deformable Models.

## 1 Introduction

Two decades have passed since Parke's pioneering work in animating faces [13]. In the span of time, significant effort has been devoted to the development of computational models of the human face for applications in such diverse areas as entertainment, low bandwidth teleconferencing, surgical facial planning, and virtual reality. However, the task of accurately modeling the expressive human face by computer remains a major challenge.

Traditionally, computer facial animation follows three basic procedures: (1) design a 3D facial mesh, (2) digitize the 3D mesh, and (3) animate the 3D mesh in a controlled fashion to simulate facial actions.

In procedure (1), it is desirable to have a refined topological mesh that captures the facial geometry. Often this entails digitizing

<sup>1</sup>Department of Computer Science, 10 King's College Road, Toronto, ON, Canada, M5S 1A4. {vlee | dt}@cs.toronto.edu

<sup>2</sup>Cambridge Research Lab., One Kendall Square, Cambridge, MA 02139. waters@crl.dec.com

Published in the Proceedings of SIGGRAPH 95 (Los Angeles, CA, August, 1995). In *Computer Graphics Proceedings, Annual Conference Series*, 1995, ACM SIGGRAPH, pp. 55–62.

as many nodes as possible. Care must be taken not to oversample the surface because there is a trade-off between the number of nodes and the computational cost of the model. Consequently, meshes developed to date capture the salient features of the face with as few nodes as possible (see [17, 14, 21, 9, 23] for several different mesh designs).

In procedure (2), a general 3D digitization technique uses photogrammetry of several images of the face taken from different angles. A common technique is to place markers on the face that can be seen from two or more cameras. An alternative technique is to manually digitize a plaster cast of the face using manual 3D digitization devices such as orthogonal magnetic fields sound captors [9], or one to two photographs [9, 7, 1]. More recently, automated laser range finders can digitize on the order of  $10^5$  3D points from a solid object such as a person's head and shoulders in just a few seconds [23].

In procedure (3), an animator must decide which mesh nodes to articulate and how much they should be displaced in order to produce a specific facial expression. Various approaches have been proposed for deforming a facial mesh to produce facial expressions; for example, parameterized models [14, 15], control-point models [12, 7], kinematic muscle models [21, 9], a texture-map-assembly model [25], a spline model [11], feature-tracking models [24, 16], a finite element model [6], and dynamic muscle models [17, 20, 8, 3].

## 1.1 Our Approach

The goal of our work is to automate the challenging task of creating realistic facial models of individuals suitable for animation. We develop an algorithm that begins with cylindrical range and reflectance data acquired by a Cyberware scanner and automatically constructs an efficient and fully functional model of the subject's head, as shown in Plate 1. The algorithm is applicable to various individuals (Plate 2 shows the raw scans of several individuals). It proceeds in two steps:

In step 1, the algorithm adapts a well-structured face mesh from [21] to the range and reflectance data acquired by scanning the subject, thereby capturing the shape of the subject's face. This approach has significant advantages because it avoids repeated manual modification of control parameters to compensate for geometric variations in the facial features from person to person. More specifically, it allows the automatic placement of facial muscles and enables the use of a single control process across different facial models.

The generic face mesh is adapted automatically through an image analysis technique that searches for salient local minima and maxima in the range image of the subject. The search is directed according to the known relative positions of the nose, eyes, chin, ears, and other facial features with respect to the generic mesh. Facial muscle emergence and attachment points are also known relative to the generic mesh and are adapted automatically as the mesh is conformed to the scanned data.

In step 2, the algorithm elaborates the geometric model constructed in step 1 into a functional, physics-based model of the subject's face which is capable of facial expression, as shown in the lower portion of Plate 1.

We follow the physics-based facial modeling approach proposed

by Terzopoulos and Waters [20]. Its basic features are that it animates facial expressions by contracting synthetic muscles embedded in an anatomically motivated model of skin composed of three spring-mass layers. The physical simulation propagates the muscle forces through the physics-based synthetic skin thereby deforming the skin to produce facial expressions. Among the advantages of the physics-based approach are that it greatly enhances the degree of realism over purely geometric facial modeling approaches, while reducing the amount of work that must be done by the animator. It can be computationally efficient. It is also amenable to improvement, with an increase in computational expense, through the use of more sophisticated biomechanical models and more accurate numerical simulation methods.

We propose a more accurate biomechanical model for facial animation compared to previous models. We develop a new biomechanical facial skin model which is simpler and better than the one proposed in [20]. Furthermore, we argue that the skull is an important biomechanical structure with regard to facial expression [22]. To date, the skin-skull interface has been underemphasized in facial animation despite its importance in the vicinity of the articulate jaw; therefore we improve upon previous facial models by developing an algorithm to estimate the skull structure from the acquired range data, and prevent the synthesized facial skin from penetrating the skull.

Finally, our algorithm includes an articulated neck and synthesizes subsidiary organs, including eyes, eyelids, and teeth, which cannot be adequately imaged or resolved in the scanned data, but which are nonetheless crucial for realistic facial animation.

## 2 Generic Face Mesh and Mesh Adaptation

The first step of our approach to constructing functional facial models of individuals is to scan a subject using a Cyberware Color Digitizer™. The scanner rotates 360 degrees around the subject, who sits motionless on a stool as a laser stripe is projected onto the head and shoulders. Once the scan is complete, the device has acquired two registered images of the subject: a range image (Figure 1) — a topographic map that records the distance from the sensor to points on the facial surface, and a reflectance (RGB) image (Figure 2) — which registers the color of the surface at those points. The images are in cylindrical coordinates, with longitude (0–360) degrees along the x axis and vertical height along the y axis. The resolution of the images is typically  $512 \times 256$  pixels (cf. Plate 1)

The remainder of this section describes an algorithm which reduces the acquired geometric and photometric data to an efficient geometric model of the subject’s head. The algorithm is a two-part process which repairs defects in the acquired images and conforms a generic facial mesh to the processed images using a feature-based matching scheme. The resulting mesh captures the facial geometry as a polygonal surface that can be texture mapped with the full resolution reflectance image, thereby maintaining a realistic facsimile of the subject’s face.

### 2.1 Image Processing

One of the problems of range data digitization is illustrated in Figure 1(a). In the hair area, in the chin area, nostril area, and even in the pupils, laser beams tend to disperse and the sensor observes no range value for these corresponding 3D surface points. We must correct for missing range and texture information.

We use a *relaxation method* to interpolate the range data. In particular, we apply a membrane interpolation method described in [18]. The relaxation interpolates values for the missing points so as to bring them into successively closer agreement with surrounding points by repeatedly indexing nearest neighbor values. Intuitively, it stretches an elastic membrane over the gaps in the surface. The images interpolated through relaxation are shown in Figure 1(b) and



Figure 1: (a) Range data of “Grace” from a Cyberware scanner. (b) Recovered plain data.

illustrate improvements in the hair area and chin area. Relaxation works effectively when the range surface is smooth, and particularly in the case of human head range data, the smoothness requirement of the solutions is satisfied quite effectively.

Figure 2(a) shows two  $512 \times 256$  reflectance (RGB) texture maps as monochrome images. Each reflectance value represents the surface color of the object in cylindrical coordinates with corresponding longitude (0–360 degrees) and latitude. Like range images, the acquired reflectance images are lacking color information at certain points. This situation is especially obvious in the hair area and the shoulder area (see Figure 2(a)). We employ the membrane relaxation approach to interpolate the texture image by repeated averaging of neighboring known colors. The original texture image in Figure 2(a) can be compared with the interpolated texture image in Figure 2(b).



Figure 2: (a) Texture data of “George” with void points displayed in white and (b) texture image interpolated using relaxation method.

The method is somewhat problematic in the hair area where range variations may be large and there is a relatively high percentage of missing surface points. A thin-plate relaxation algorithm [18] may be more effective in these regions because it would fill in the larger gaps with less “flattening” than a membrane [10].

Although the head structure in the cylindrical laser range data is distorted along the longitudinal direction, important features such as the slope changes of the nose, forehead, chin, and the contours of the mouth, eyes, and nose are still discernible. In order to locate the contours of those facial features for use in adaptation (see below), we use a modified Laplacian operator (applied to the discrete image through local pixel differencing) to detect edges from the range map shown in Figure 3(a) and produce the field function in Fig. 3(b). For details about the operator, see [8]. The field function highlights important features of interest. For example, the local maxima of the modified Laplacian reveals the boundaries of the lips, eyes, and chin.

### 2.2 Generic Face Mesh and Mesh Adaptation

The next step is to reduce the large arrays of data acquired by the scanner into a parsimonious geometric model of the face that can eventually be animated efficiently. Motivated by the adaptive meshing techniques [19] that were employed in [23], we significantly



Figure 3: (a) Original range map. (b) Modified Laplacian field function of (a).

improved the technique by adapting a generic face mesh to the data. Figure 4 shows the planar generic mesh which we obtain through a cylindrical projection of the 3D face mesh from [21]. One of the advantages of the generic mesh is that it has well-defined features which form the basis for accurate feature based adaptation to the scanned data and automatic scaling and positioning of facial muscles as the mesh is deformed to fit the images. Another advantage is that it automatically produces an efficient triangulation, with finer triangles over the highly curved and/or highly articulate regions of the face, such as the eyes and mouth, and larger triangles elsewhere.

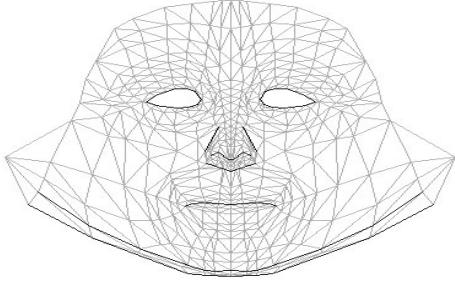


Figure 4: Facial portion of generic mesh in 2D cylindrical coordinates. Dark lines are features for adaptation.

We label all facial feature nodes in the generic face prior to the adaptation step. The feature nodes include eye contours, nose contours, mouth contours, and chin contours.

For any specific range image and its positive Laplacian field function (Figure 3), the generic mesh adaptation procedure performs the following steps to locate feature points in the range data (see [8] for details):

#### Mesh Adaptation Procedures

1. Locate nose tip
2. Locate chin tip
3. Locate mouth contour
4. Locate chin contour
5. Locate ears
6. Locate eyes
7. Activate spring forces
8. Adapt hair mesh
9. Adapt body mesh
10. Store texture coordinates

Once the mesh has been fitted by the above feature based matching technique (see Plate 3), the algorithm samples the range image at the location of the nodes of the face mesh to capture the facial geometry, as is illustrated in Figure 5.

The node positions also provide texture map coordinates that are used to map the full resolution color image onto the triangles (see Plate 3).

### 2.3 Estimation of Relaxed Face Model

Ideally, the subject's face should be in a neutral, relaxed expression when he or she is being scanned. However, the scanned woman in

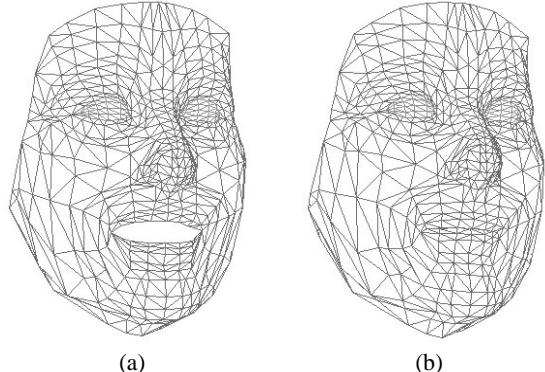


Figure 5: (a) Generic geometric model conformed to Cyberware scan of "Heidi". (b) Same as (a). Note that "Heidi's" mouth is now closed, subsequent to estimation of the relaxed face geometry.

the "Heidi" dataset is smiling and her mouth is open (see Plate 2). We have made our algorithm tolerant of these situations. To construct a functional model, it is important to first estimate the relaxed geometry. That is, we must infer what the "Heidi" subject would look like had her face been in a relaxed pose while she was being scanned. We therefore estimate the range values of the closed mouth contour from the range values of the open mouth contour by the following steps:

1. Perform adaptation procedures in Sec. 2.2 without step 3.
2. Store nodal longitude/latitude into adapted face model.
3. Perform lip adaptation in step 3 in sec. 2.2
4. Store nodal range values into adapted face model.

As a result, the final reconstructed face model in Figure 5(b) will have a relaxed mouth because the longitude and latitude recorded is the default shape of our closed mouth model (see Figure 4). Moreover, the shape of the final reconstructed face is still faithful to the head data because the range value at each facial nodal point is obtained correctly after the lip adaptation procedure has been performed. Relaxing the face shown in Figure 5(a) results in the image in Figure 5(b) (with eyelids inserted — see below).

## 3 The Dynamic Skin and Muscle Model

This section describes how our system proceeds with the construction of a fully functional model of the subject's face from the facial mesh produced by the adaptation algorithm described in the previous section. To this end, we automatically create a dynamic model of facial tissue, estimate a skull surface, and insert the major muscles of facial expression into the model. The following sections describe each of these components. We also describe our high-performance parallel, numerical simulation of the dynamic facial tissue model.

### 3.1 Layered Synthetic Tissue Model

The skull is covered by deformable tissue which has five distinct layers [4]. Four layers—epidermis, dermis, sub-cutaneous connective tissue, and fascia—comprise the skin, and the fifth consists of the muscles of facial expression. Following [20], and in accordance with the structure of real skin [5], we have designed a new, synthetic tissue model (Figure 6(a)).

The tissue model is composed of triangular prism elements (see Figure 6(a)) which match the triangles in the adapted facial mesh. The epidermal surface is defined by nodes 1, 2, and 3, which are connected by epidermal springs. The epidermis nodes are also connected by dermal-fatty layer springs to nodes 4, 5, and 6, which define the fascia surface. Fascia nodes are interconnected by fascia

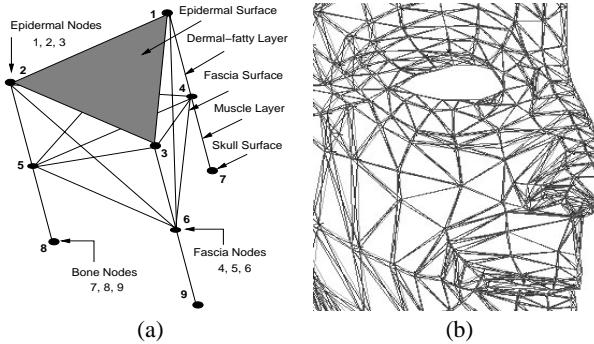


Figure 6: (a) Triangular skin tissue prism element. (b) Close-up view of right side of an individual with conformed elements.

springs. They are also connected by muscle layer springs to skull surface nodes 7, 8, 9.

Figure 9(b) shows 684 such skin elements assembled into an extended skin patch. Several synthetic muscles are embedded into the muscle layer of the skin patch and the figure shows the skin deformation due to muscle contraction. Muscles are fixed in an estimated bony subsurface at their point of emergence and are attached to fascia nodes as they run through several tissue elements. Figure 6(b) shows a close-up view of the right half of the facial tissue model adapted to an individual's face which consists of 432 elements.

### 3.2 Discrete Deformable Models (DDMs)

A discrete deformable model has a node-spring-node structure, which is a uniaxial finite element. The data structure for the node consists of the nodal mass  $m_i$ , position  $\mathbf{x}_i(t) = [x_i(t), y_i(t), z_i(t)]'$ , velocity  $\mathbf{v}_i = dx_i/dt$ , acceleration  $\mathbf{a}_i = d^2\mathbf{x}_i/dt^2$ , and net nodal forces  $\mathbf{f}_i^n(t)$ . The data structure for the spring in this DDM consists of pointers to the head node  $i$  and the tail node  $j$  which the spring interconnects, the natural or rest length  $l_k$  of the spring, and the spring stiffness  $c_k$ .

### 3.3 Tissue Model Spring Forces

By assembling the discrete deformable model according to histological knowledge of skin (see Figure 6(a)), we are able to construct an anatomically consistent, albeit simplified, tissue model. Figure 6(b) shows a close-up view of the tissue model around its eye and nose parts of a face which is automatically assembled by following the above approach.

- The force spring  $j$  exerts on node  $i$  is

$$g_j = c_j(l_j - l_j^r)\mathbf{s}_j$$

- each layer has its own stress-strain relationship  $c_j$  and the dermal-fatty layer uses biphasic springs (non-constant  $c_j$ ) [20]
- $l_j^r$  and  $l_j = \|\mathbf{x}_j - \mathbf{x}_i\|$  are the rest and current lengths for spring  $j$
- $\mathbf{s}_j = (\mathbf{x}_j - \mathbf{x}_i)/l_j$  is the spring direction vector for spring  $j$

### 3.4 Linear Muscle Forces

The muscles of facial expression, or the muscular plate, spreads out below the facial tissue. The facial musculature is attached to the skin tissue by short elastic tendons at many places in the fascia, but is fixed to the facial skeleton only at a few points. Contractions of the facial muscles cause movement of the facial tissue. We model

28 of the primary facial muscles, including the zygomatic major and minor, frontalis, nasii, corrugator, mentalis, buccinator, and anguli depressor groups. Plate 4 illustrates the effects of automatic scaling and positioning of facial muscle vectors as the generic mesh adapts to different faces.

To better emulate the facial muscle attachments to the fascia layer in our model, a group of fascia nodes situated along the muscle path—i.e., within a predetermined distance from a central muscle vector, in accordance with the muscle width—experience forces from the contraction of the muscle. The face construction algorithm determines the nodes affected by each muscle in a precomputation step.

To apply muscle forces to the fascia nodes, we calculate a force for each node by multiplying the muscle vector with a force length scaling factor and a force width scaling factor (see Figure 7(a)). Function  $\Theta_1$  (Figure 8(a)) scales the muscle force according to the length ratio  $\epsilon_{j,i}$ , while  $\Theta_2$  (Figure 8(b)) scales it according to the width  $\omega_{j,i}$  at node  $i$  of muscle  $j$ :

$$\begin{aligned} \epsilon_{j,i} &= ((\mathbf{m}_j^F - \mathbf{x}_i) \cdot \mathbf{m}_j) / (\|\mathbf{m}_j^A - \mathbf{m}_j^F\|) \\ \omega_{j,i} &= \|\mathbf{p}_i - (\mathbf{p}_i \cdot \mathbf{n}_j)\mathbf{n}_j\| \end{aligned}$$

- The force muscle  $j$  exerts on node  $i$  is

$$\mathbf{f}_i^j = \Theta_1(\epsilon_{j,i})\Theta_2(\omega_{j,i})\mathbf{m}_j$$

- $\Theta_1$  scales the force according to the distance ratio  $\epsilon_{j,i}$ , where  $\epsilon_{j,i} = \rho_{j,i}/d_j$ , with  $d_j$  the muscle  $j$  length.
- $\Theta_2$  scales the force according to the width ratio  $\omega_{j,i}/w_j$ , with  $w_j$  the muscle  $j$  width.
- $\mathbf{m}_j$  is the normalized muscle vector for muscle  $j$

Note that the muscle force is scaled to zero at the root of the muscle fiber in the bone and reaches its full strength near the end of the muscle fiber. Figure 9(b) shows an example of the effect of muscle forces applied to a synthetic skin patch.

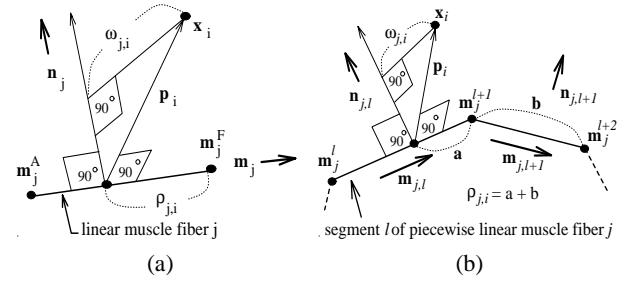


Figure 7: (a) Linear muscle fiber. (b) Piecewise linear muscle fiber.

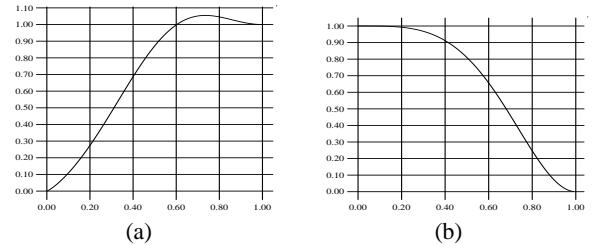


Figure 8: (a) Muscle force scaling function  $\Theta_1$  wrt  $\epsilon_{j,i}$ , (b) Muscle force scaling function  $\Theta_2$  wrt  $\omega_{j,i}/w_j$

### 3.5 Piecewise Linear Muscle Forces

In addition to using linear muscle fibers in section 3.4 to simulate sheet facial muscles like the frontalis and the zygomatics, we also model sphincter muscles, such as the orbicularis oris circling the mouth, by generalizing the linear muscle fibers to be piecewise

linear and allowing them to attach to fascia at each end of the segments. Figure 7(b) illustrates two segments of an  $N$ -segment piecewise linear muscle  $j$  showing three nodes  $\mathbf{m}_j^l$ ,  $\mathbf{m}_j^{l+1}$ , and  $\mathbf{m}_j^{l+2}$ . The unit vectors  $\mathbf{m}_{j,l}$ ,  $\mathbf{m}_{j,l+1}$  and  $\mathbf{n}_{j,l}$ ,  $\mathbf{n}_{j,l+1}$  are parallel and normal to the segments, respectively. The figure indicates fascia node  $i$  at  $\mathbf{x}_i$ , as well as the distance  $\rho_{j,i} = a + b$ , the width  $\omega_{j,i}$ , and the perpendicular vector  $\mathbf{p}_i$  from fascia node  $i$  to the nearest segment of the muscle. The length ratio  $\varepsilon_{j,i}$  for fascia node  $i$  in muscle fiber  $j$  is

$$\varepsilon_{j,i} = \frac{(\mathbf{m}_j^{l+1} - \mathbf{x}_i) \cdot \mathbf{m}_{j,l} + \sum_{k=l+1}^N \|\mathbf{m}_j^{k+1} - \mathbf{m}_j^k\|}{\sum_{k=1}^N \|\mathbf{m}_j^{k+1} - \mathbf{m}_j^k\|}$$

The width  $\omega_{j,i}$  calculation is the same as for linear muscles. The remaining muscle force computations are the same as in section 3.4. Plate 4 shows all the linear muscles and the piecewise linear sphincter muscles around the mouth.

### 3.6 Volume Preservation Forces

In order to faithfully exhibit the incompressibility [2] of real human skin in our model, a volume constraint force based on the change of volume (see Figure 9(a)) and displacements of nodes is calculated and applied to nodes. In Figure 9(b) the expected effect of volume preservation is demonstrated. For example, near the origin of the muscle fiber, the epidermal skin is bulging out, and near the end of the muscle fiber, the epidermal skin is depressed.

- The volume preservation force element  $e$  exerts on nodes  $i$  in element  $e$  is

$$\mathbf{q}_i^e = k_1(V^e - \tilde{V}^e)\mathbf{n}_i^e + k_2(\mathbf{p}_i^e - \tilde{\mathbf{p}}_i^e)$$

- $\tilde{V}^e$  and  $V^e$  are the rest and current volumes for  $e$
- $\mathbf{n}_i^e$  is the epidermal normal for epidermal node  $i$
- $\tilde{\mathbf{p}}_i^e$  and  $\mathbf{p}_i^e$  are the rest and current nodal coordinates for node  $i$  with respect to the center of mass of  $e$
- $k_1, k_2$  are force scaling constants

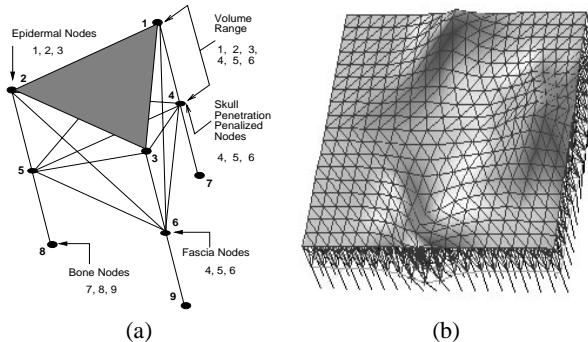


Figure 9: (a) Volume preservation and skull nonpenetration element. (b) Assembled layered tissue elements under multiple muscle forces.

### 3.7 Skull Penetration Constraint Forces

Because of the underlying impenetrable skull of a human head, the facial tissue during a facial expression will slide over the underlying bony structure. With this in mind, for each individual's face model reconstructed from the laser range data, we estimate the skull surface normals to be the surface normals in the range data image. The skull is then computed as an offset surface. To prevent nodes from penetrating the estimated skull (see Figure 9(a)), we apply a skull non-penetration constraint to cancel out the force component on the fascia node which points into the skull; therefore, the resulting force will make the nodes slide over the skull.

- The force to penalize fascia node  $i$  during motion is:

$$\mathbf{s}_i = \begin{cases} -(\mathbf{f}_i^n \cdot \mathbf{n}_i)\mathbf{n}_i & \text{when } \mathbf{f}_i^n \cdot \mathbf{n}_i < 0 \\ \mathbf{0} & \text{otherwise} \end{cases}$$

- $\mathbf{f}_i^n$  is the net force on fascia node  $i$
- $\mathbf{n}_i$  is the nodal normal of node  $i$

### 3.8 Equations of Motion for Tissue Model

Newton's law of motion governs the response of the tissue model to forces. This leads to a system of coupled second order ODEs that relate the node positions, velocities, and accelerations to the nodal forces. The equation for node  $i$  is

$$m_i \frac{d^2\mathbf{x}_i}{dt^2} + \gamma_i \frac{d\mathbf{x}_i}{dt} + \tilde{\mathbf{g}}_i + \tilde{\mathbf{q}}_i + \tilde{\mathbf{s}}_i + \tilde{\mathbf{h}}_i = \tilde{\mathbf{f}}_i$$

- $m_i$  is the nodal mass,
- $\gamma_i$  is the damping coefficient,
- $\tilde{\mathbf{g}}_i$  is the total spring force at node  $i$ ,
- $\tilde{\mathbf{q}}_i$  is the total volume preservation force at node  $i$ ,
- $\tilde{\mathbf{s}}_i$  is the total skull penetration force at node  $i$ ,
- $\tilde{\mathbf{h}}_i$  is the total nodal restoration force at node  $i$ ,
- $\tilde{\mathbf{f}}_i$  is the total applied muscle force at node  $i$ ,

### 3.9 Numerical Simulation

The solution to the above system of ODEs is approximated by using the well-known, explicit Euler method. At each iteration, the nodal acceleration at time  $t$  is computed by dividing the net force by nodal mass. The nodal velocity is then calculated by integrating once, and another integration is done to compute the nodal positions at the next time step  $t + \Delta t$ , as follows:

$$\begin{aligned} \mathbf{a}_i^t &= \frac{1}{m_i}(\tilde{\mathbf{f}}_i^t - \gamma_i \mathbf{v}_i^t - \tilde{\mathbf{g}}_i^t - \tilde{\mathbf{q}}_i^t - \tilde{\mathbf{s}}_i^t - \tilde{\mathbf{h}}_i^t) \\ \mathbf{v}_i^{t+\Delta t} &= \mathbf{v}_i^t + \Delta t \mathbf{a}_i^t \\ \mathbf{x}_i^{t+\Delta t} &= \mathbf{x}_i^t + \Delta t \mathbf{v}_i^{t+\Delta t} \end{aligned}$$

### 3.10 Default Parameters

The default parameters for the physical/numerical simulation and the spring stiffness values of different layers are as follows:

	Mass ( $m$ )	Time step ( $\Delta t$ )	Damping ( $\gamma$ )
	0.5	0.01	30

	Epid	Derm-fat 1	Derm-fat 2	Fascia	Muscle
$c$	60	30	70	80	10

### 3.11 Parallel Processing for Facial Animation

The explicit Euler method allows us to easily carry out the numerical simulation of the dynamic skin/muscle model in parallel. This is because at each time step all the calculations are based on the results from the previous time step. Therefore, parallelization is achieved by evenly distributing calculations at each time step to all available processors. This parallel approach increases the animation speed to allow us to simulate facial expressions at interactive rates on our Silicon Graphics multiprocessor workstation.

## 4 Geometry Models for Other Head Components

To complete our physics-based face model, additional geometric models are combined along with the skin/muscle/skull models developed in the previous section. These include the eyes, eyelids, teeth, neck, hair, and bust (Figure 10). See Plate 5 for an example of a complete model.

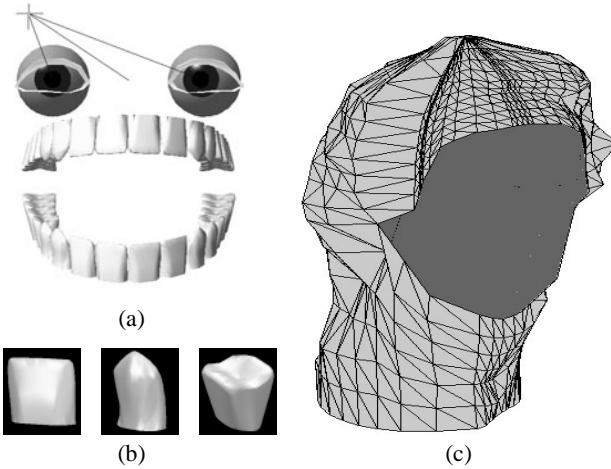


Figure 10: (a) Geometric models of eyes, eyelids, and teeth (b) Incisor, canine, and molar teeth. (c) hair and neck.

#### 4.1 Eyes

Eyes are constructed from spheres with adjustable irises and adjustable pupils (Figure 10(a)). The eyes are automatically scaled to fit the facial model and are positioned into it. The eyes rotate kinematically in a coordinated fashion so that they will always converge on a specified fixation point in three-dimensional space that defines the field of view. Through a simple illumination computation, the eyes can automatically dilate and contract the pupil size in accordance with the amount of light entering the eye.

#### 4.2 Eyelids

The eyelids are polygonal models which can blink kinematically during animation (see Figure 10(a)). Note that the eyelids are open in Figure 10(a).

If the subject is scanned with open eyes, the sensor will not observe the eyelid texture. An eyelid texture is synthesized by a relaxation based interpolation algorithm similar to the one described in section 2.1. The relaxation algorithm interpolates a suitable eyelid texture from the immediately surrounding texture map. Figure 11 shows the results of the eyelid texture interpolation.

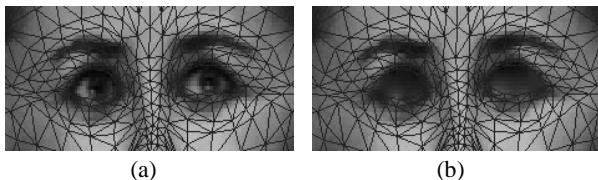


Figure 11: (a) Face texture image with adapted mesh before eyelid texture synthesis (b) after eyelid texture synthesis.

#### 4.3 Teeth

We have constructed a full set of generic teeth based on dental images. Each tooth is a NURBS surfaces of degree 2. Three different teeth shapes, the incisor, canine, and molar, are modeled (Figure 10(b)). We use different orientations and scalings of these basic shapes to model the full set of upper and lower teeth shown in Figure 10(a). The dentures are automatically scaled to fit in length, curvature, etc., and are positioned behind the mouth of the facial model.

#### 4.4 Hair, Neck, and Bust Geometry

The hair and bust are both rigid polygonal models (see Figure 10(c)). They are modeled from the range data directly, by extending the

facial mesh in a predetermined fashion to the boundaries of the range and reflectance data, and sampling the images as before.

The neck can be twisted, bent and rotated with three degrees of freedom. See Figure 12 for illustrations of the possible neck articulations.

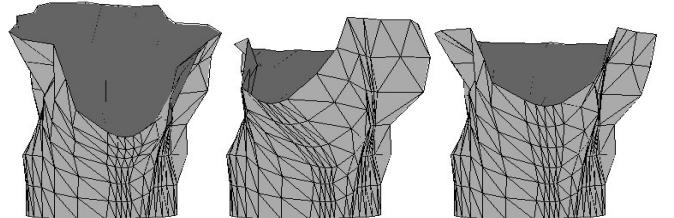


Figure 12: articulation of neck.

### 5 Animation Examples

Plate 1 illustrates several examples of animating the physics-based face model after conformation to the “Heidi” scanned data (see Plate 2).

- The *surprise* expression results from contraction of the outer frontalis, major frontalis, inner frontalis, zygomatics major, zygomatics minor, depressor labii, and mentalis, and rotation of the jaw.
- The *anger* expression results from contraction of the corrugator, lateral corrugator, levator labii, levator labii nasi, anguli depressor, depressor labii, and mentalis.
- The *quizzical look* results from an asymmetric contraction of the major frontalis, outer frontalis, corrugator, lateral corrugator, levator labii, and buccinator.
- The *sadness* expression results from a contraction of the inner frontalis, corrugator, lateral corrugator, anguli depressor, and depressor labii.

Plate 6 demonstrates the performance of our face model construction algorithm on two male individuals (“Giovanni” and “Mick”). Note that the algorithm is tolerant of some amount of facial hair.

Plate 7 shows a third individual “George.” Note the image at the lower left, which shows two additional expression effects—cheek puffing, and lip puckering—that combine to simulate the vigorous blowing of air through the lips. The cheek puffing was created by applying outwardly directed radial forces to “inflate” the deformable cheeks. The puckered lips were created by applying radial pursing forces and forward protruding forces to simulate the action of the orbicularis oris sphincter muscle which circles the mouth.

Finally, Plate 8 shows several frames from a two-minute animation “*Bureaucrat Too*” (a second-generation version of the 1990 “*Bureaucrat*” which was animated using the generic facial model in [20]). Here “George” tries to read landmark papers on facial modeling and deformable models in the SIGGRAPH ’87 proceedings, only to realize that he doesn’t yet have a brain!

### 6 Conclusion and Future Work

The human face consists of a biological tissue layer with nonlinear deformation properties, a muscle layer knit together under the skin, and an impenetrable skull structure beneath the muscle layer. We have presented a physics-based model of the face which takes all of these structures into account. Furthermore, we have demonstrated a new technique for automatically constructing face models of this sort and conforming them to individuals by exploiting high-resolution laser scanner data. The conformation process is carried out by a feature matching algorithm based on a reusable generic

mesh. The conformation process, efficiently captures facial geometry and photometry, positions and scales facial muscles, and also estimates the skull structure over which the new synthetic facial tissue model can slide. Our facial modeling approach achieves an unprecedented level of realism and fidelity to any specific individual. It also achieves a good compromise between the complete emulation of the complex biomechanical structures and functionality of the human face and real-time simulation performance on state-of-the-art computer graphics and animation hardware.

Although we formulate the synthetic facial skin as a layered tissue model, our work does not yet exploit knowledge of the variable thickness of the layers in different areas of the face. This issue will in all likelihood be addressed in the future by incorporating additional input data about the subject acquired using noninvasive medical scanners such as CT or MR.

## Acknowledgments

The authors thank Lisa White and Jim Randall for developing the piecewise linear muscle model used to model the mouth. Range/RGB facial data were provided courtesy of Cyberware, Inc., Monterey, CA. The first two authors thank the Natural Science and Engineering Research Council of Canada for financial support. DT is a fellow of the Canadian Institute for Advanced Research.

## References

- [1] T. Akimoto, Y. Suenaga, and R. Wallace. Automatic creation of 3D facial models. *IEEE Computer Graphics and Applications*, 13(5):16–22, September 1993.
- [2] James Doyle and James Philips. *Manual on Experimental Stress Analysis*. Society for Experimental Mechanics, fifth edition, 1989.
- [3] Irfan A. Essa. *Visual Interpretation of Facial Expressions using Dynamic Modeling*. PhD thesis, MIT, 1994.
- [4] Frick and Hans. *Human Anatomy*, volume 1. Thieme Medical Publishers, Stuttgart, 1991.
- [5] H. Gray. *Anatomy of the Human Body*. Lea & Febiber, Philadelphia, PA, 29th edition, 1985.
- [6] Brian Guenter. A system for simulating human facial expression. In *State of the Art in Computer Animation*, pages 191–202. Springer-Verlag, 1992.
- [7] T. Kurihara and K. Arai. A transformation method for modeling and animation of the human face from photographs. In *State of the Art in Computer Animation*, pages 45–57. Springer-Verlag, 1991.
- [8] Y.C. Lee, D. Terzopoulos, and K. Waters. Constructing physics-based facial models of individuals. In *Proceedings of Graphics Interface '93*, pages 1–8, Toronto, May 1993.
- [9] N. Magneneat-Thalmann, H. Minh, M. Angelis, and D. Thalmann. Design, transformation and animation of human faces. *Visual Computer*, 5:32–39, 1989.
- [10] D. Metaxas and E. Milios. Reconstruction of a color image from nonuniformly distributed sparse and noisy data. *Computer Vision, Graphics, and Image Processing*, 54(2):103–111, March 1992.
- [11] M. Nahas, H. Hutric, M. Rioux, and J. Domey. Facial image synthesis using skin texture recording. *Visual Computer*, 6(6):337–343, 1990.
- [12] M. Oka, K. Tsutsui, A. Ohba, Y. Kurauchi, and T. Tago. Real-time manipulation of texture-mapped surfaces. In *SIGGRAPH 21*, pages 181–188. ACM Computer Graphics, 1987.
- [13] F. Parke. Computer generated animation of faces. In *ACM National Conference*, pages 451–457. ACM, 1972.
- [14] F. Parke. Parameterized models for facial animation. *IEEE Computer Graphics and Applications*, 2(9):61–68, November 1982.
- [15] F. Parke. Parameterized models for facial animation revisited. In *SIGGRAPH Facial Animation Tutorial Notes*, pages 43–56. ACM SIGGRAPH, 1989.
- [16] Elizabeth C. Patterson, Peter C. Litwinowicz, and N. Greene. Facial animation by spatial mapping. In *State of the Art in Computer Animation*, pages 31–44. Springer-Verlag, 1991.
- [17] S. Platt and N. Badler. Animating facial expression. *Computer Graphics*, 15(3):245–252, August 1981.
- [18] D. Terzopoulos. The computation of visible-surface representations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-10(4):417–438, 1988.
- [19] D. Terzopoulos and M. Vasilescu. Sampling and reconstruction with adaptive meshes. In *Proceedings of Computer Vision and Pattern Recognition Conference*, pages 70–75. IEEE, June 1991.
- [20] D. Terzopoulos and K. Waters. Physically-based facial modeling, analysis, and animation. *Visualization and Computer Animation*, 1:73–80, 1990.
- [21] K. Waters. A muscle model for animating three-dimensional facial expression. *Computer Graphics*, 22(4):17–24, 1987.
- [22] K. Waters. A physical model of facial tissue and muscle articulation derived from computer tomography data. In *Visualization in Biomedical Computing*, pages 574–583. SPIE, Vol. 1808, 1992.
- [23] K. Waters and D. Terzopoulos. Modeling and animating faces using scanned data. *Visualization and Computer Animation*, 2:123–128, 1991.
- [24] L. Williams. Performance-driven facial animation. In *SIGGRAPH 24*, pages 235–242. ACM Computer Graphics, 1990.
- [25] J. Yau and N. Duffy. 3-D facial animation using image samples. In *New Trends in Computer Graphics*, pages 64–73. Springer-Verlag, 1988.



Plate 1: Objective. *Input*: Range map in 3D and texture map (top). *Output*: Functional face model for animation.



Plate 2: Raw 512 × 256 digitized data for Heidi (top left), George (top right), Giovanni (bottom left), Mick (bottom right).

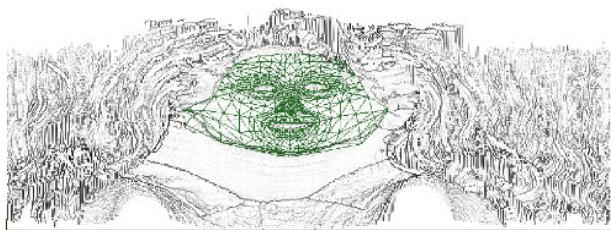


Plate 3: Adapted face mesh overlaying texture map and Laplacian filtered range map of Heidi.



Plate 6: Animation examples of Giovanni and Mick.

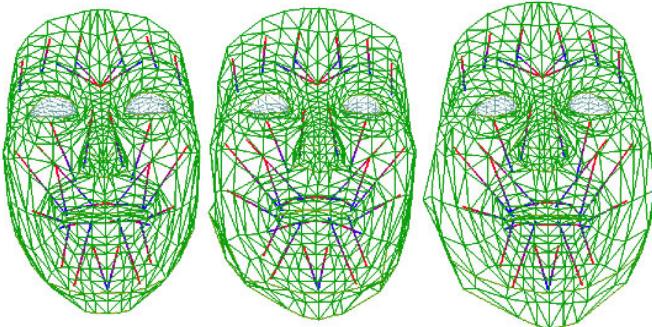


Plate 4: Muscle fiber vector embedded in generic face model and two adapted faces of Heidi and George.

Plate 7: Animation example of George.



Plate 5: Complete, functional head model of Heidi with physics-based face and geometric eyes, teeth, hair, neck, and shoulders (in Monument Valley).



Plate 8: George in four scenes from “Bureaucrat Too”.

# The Artificial Life of Virtual Humans



**Daniel Thalmann**  
**Computer Graphics Lab**  
**EPFL**  
**CH 1015 Lausanne**  
**[thalmann@lig.di.epfl.ch](mailto:thalmann@lig.di.epfl.ch)**  
**<http://ligwww.epfl.ch>**





## Real-time body deformations

- direct contour deformation
- avoid contours computation from implicit surfaces



## Avatars

- Representation of User
- Animation correlated to Actual body
- Real-Time Performance Animation
- Requires High-End VR devices



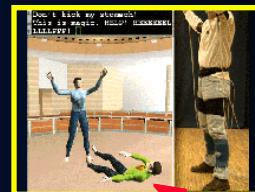
## User-guided Actors

- Second type of avatar
- User controls motion
- Simple devices
- Realistic motion with joints



## Autonomous Actors

- Autonomous: without external intervention
- Its own goals, rules
  - Increases real-time interaction with environment
  - Hence, increases presence of participants



## Interactive Perceptive Virtual Humans

- react to environment
- own behavior
- take decision based on perception, memory, and reasoning
- communicate with other virtual humans and real people



```
behavioral loop (on time)
initialize animation environment
while (not terminated) {
    update scene
    for each actor
        realize perception of environment
        select actions based on sensorial
        input, actual state, specific behavior
    for each actor
        execute selected actions }
```



## **Actions**

- **Strongly related to Motion Control Methods**
- **Dependent on the situation of actor(s)**
- **single actor,**
- **in an environment,**
- **interacting with other actors,**
- **interacting with the user**



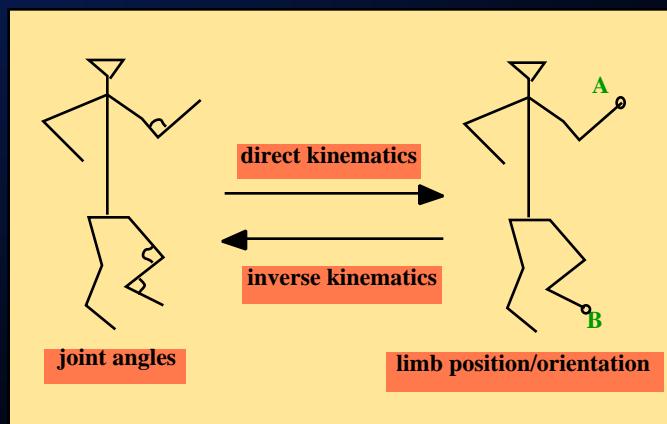
## Motion Capture



- Flock of birds
- Calibration retaining motion realism
- Process of perturbation introduced by soft tissue and muscle displacement

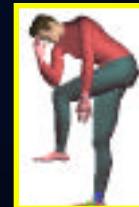


## Direct and Inverse Kinematics



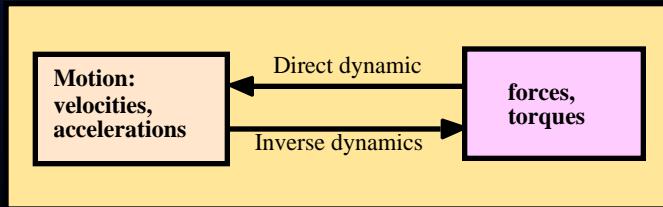
## Inverse Kinetics

- control center of mass position control
- combines information of articulated structure kinematics with mass distribution.
- concept of "Influence Tree" to represent fraction of body supported by given supporting site



## Dynamics

use forces,  
torques,  
constraints and  
mass properties of  
objects



## walking



- locomotion with body's center of gravity moves alternately on right and left side.
- At all times at least one foot in contact with floor and during brief phase both feet in contact with floor

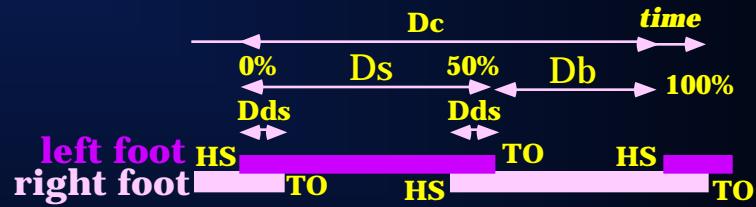


## Our Walking Model (Boulic et al. 90)

- parameterization coming from biomechanical data
- produces at any time values of spatial, temporal and joint parameters of free walking with average characteristics.
- spatial values normalized by height of thigh



## Temporal structure of walking cycle



D<sub>c</sub>: Duration of cycle     HS: Heel Strike

TO: Toe Off

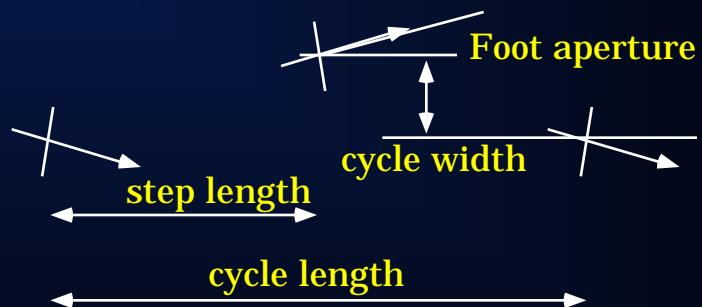
D<sub>s</sub>: Duration of support (contact with floor)

D<sub>b</sub>: Duration of balance (non-contact with floor)

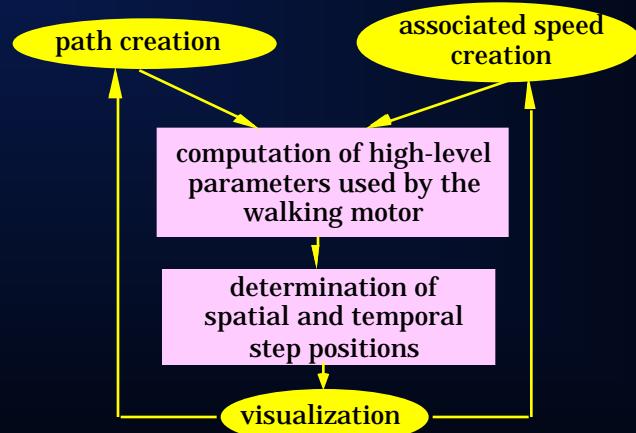
D<sub>ds</sub>: duration of double support



## Spatial structure of walking cycle



## Walking on curved path



## Grasping



3 steps:

- **Heuristic grasping decision**
- **Inverse kinematics to find final arm posture**
- **Multi-sensors**



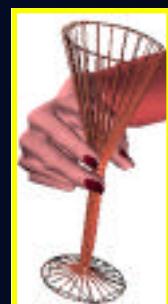
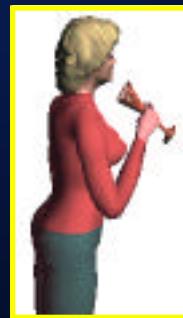
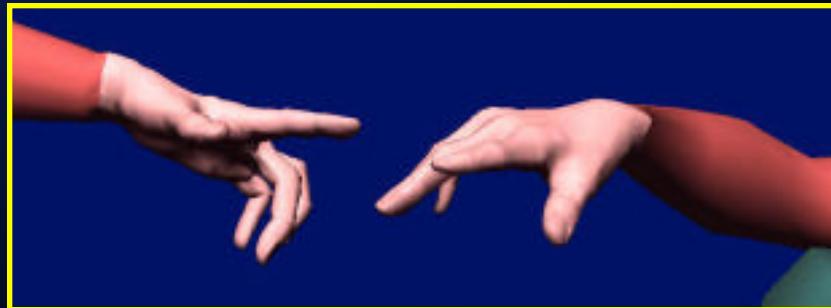


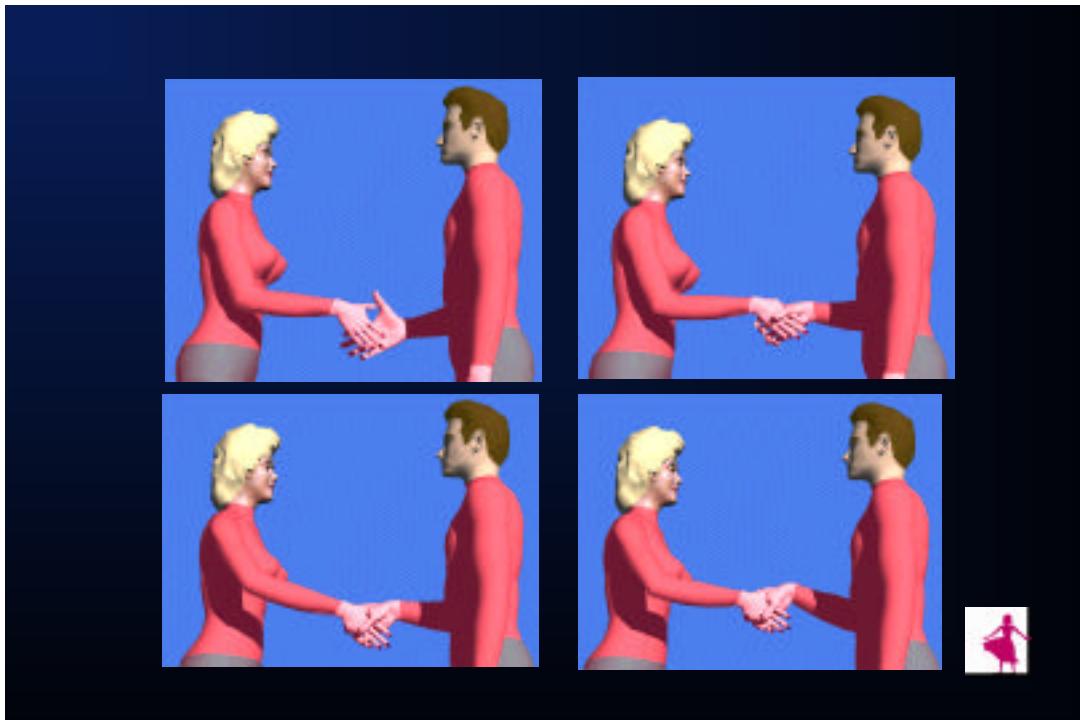
## Multi-sensors

- sphere multi-sensors with both touch and length sensor properties
- attached to articulated figure
- sensor activated for any collision with other objects or sensors.



## Hand deformations



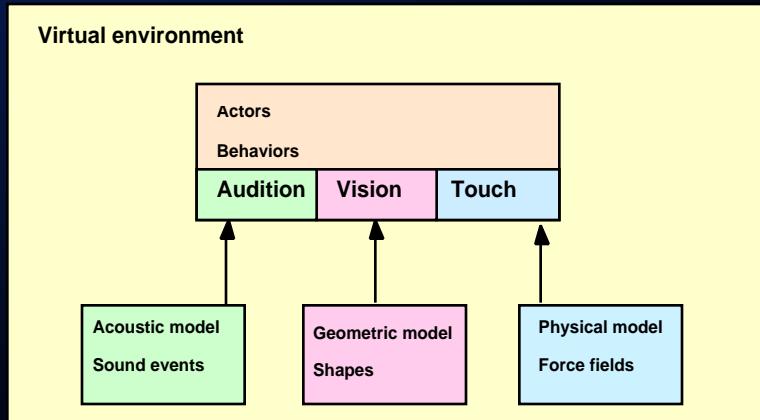


## Perception

- Humans remember discrete properties like the tree is on the right of the car.
- objective: give to actor behavior maximum of believability
- how ? by providing to actor world representation like representation that humans have.



## Perception through Virtual Sensors

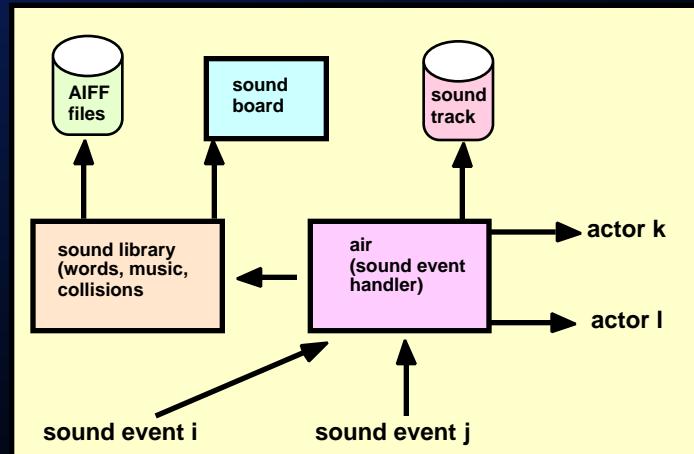


### Virtual audition

- model sound environment
- actors can directly access to positional and semantic sound source information of audible sound event.



## Virtual audition: Sound events



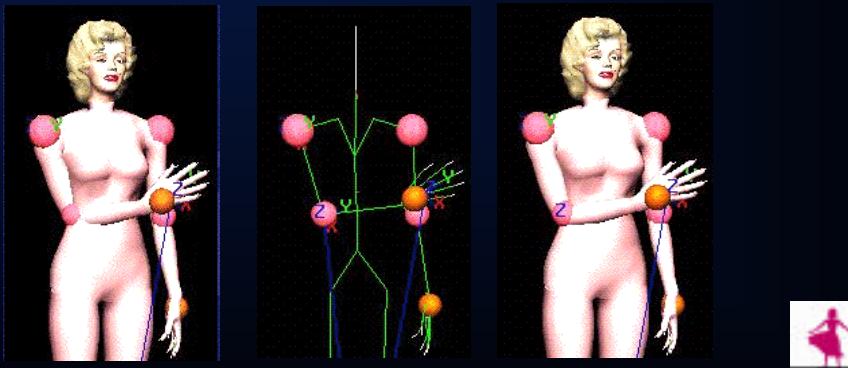
## Virtual tactile

- corresponds roughly to collision detection process
- information provided to actor
- could be also used for self collisions



## Self body collisions

- Sensor-body collision test
- Collision correction using Secondary Task of Inverse Kinematics



## Virtual vision

- Renault et al. (1990) first synthetic vision
- Initial objective: actor automatically moving in corridor avoiding obstacles.
- Actor uses vision for perception of world and input to behavioural model.



## Synthetic Vision simulation

- avoids problems of pattern recognition involved in robotic vision.
- Input: description of 3D environment, camera
- Output: view: 2D array of pixels. Each pixel contains distance between eye and point of object.



## Implementation

- based on hardware
- uses z-buffer and double frame buffer
- backbuffer: for each pixel, object identifier stored.
- front buffer: to display object projection animator may know what virtual actor sees.



## Case-study: Navigation

### Global navigation

- uses prelearned model of domain
- might not reflect recent changes in environment.

### Local navigation

- uses direct input information from environment to reach subgoals



## Case study: Sensor-based Tennis

- tennis playing: human activity severely based on vision of players.
- vision system to recognize flying ball, estimate trajectory, localize partner for game strategy planning.
- automata able to estimate collision ball / racquet and perform hit with given force and direction.



## **Autonomous referee**

- judges game by following ball with vision
- updates state of match when hears ball collision event according to vision and knowledge of tennis match
- communicates decisions and state of game by spoken words



## **A Model of Nonverbal Communication**

- concerned with postures and their indications on what people are feeling
- use relative positioning of both persons and their postures.



## Emotional state

- denotes general desire to communicate.
- represented with normalized value.
- low emotional state      weak desire to communicate.
- if actor nevertheless communicates, postures unfriendly.
- high emotional state      friendly postures.



- emotional state of actor affects way of walking.
- low emotional state gives sad look to walking and high happy look



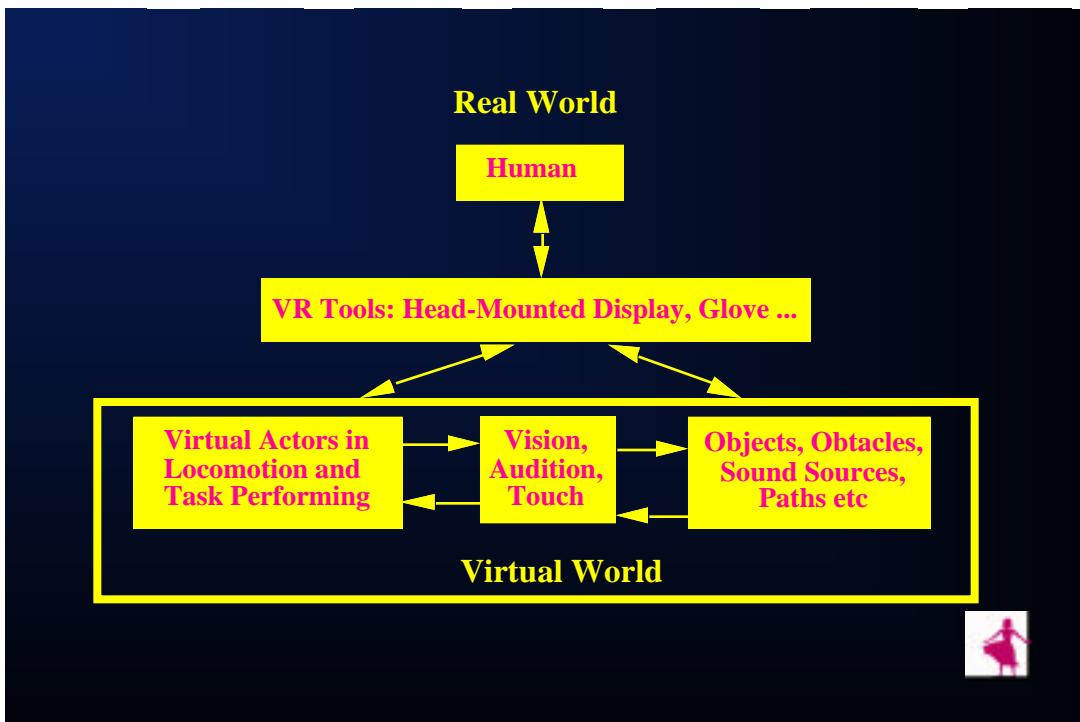
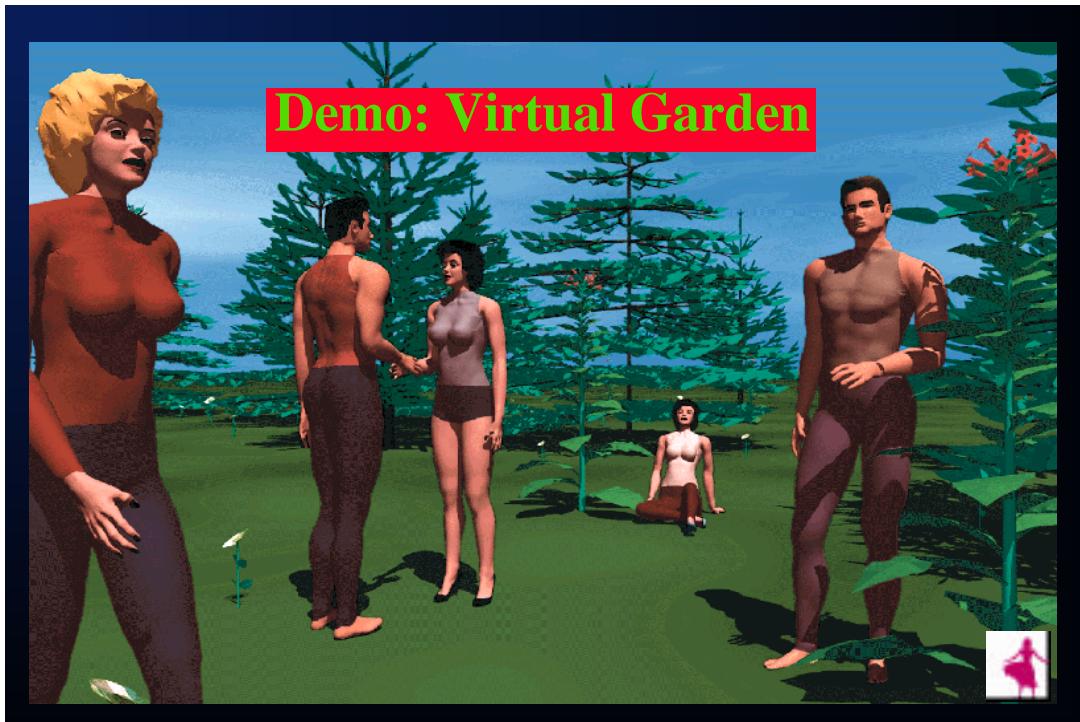
## Relationship

- normalized value
- each actor maintains description of relationship with each other actor
- low value: bad relationship
- high value: good relationship
- updated according to issue of communication.



- desire of actor to communicate with other actor evaluated with emotional state and relationship with it.
- actor with low emotional state may communicate if relationship good.
- actor with high emotional state but bad relationship may decide not to communicate.





## **Real Time Human Action Recognition for Behavioral Interaction with Autonomous Virtual Humans**

- new hierarchical model for human actions
- efficient action recognition algorithm
- does not need training phases



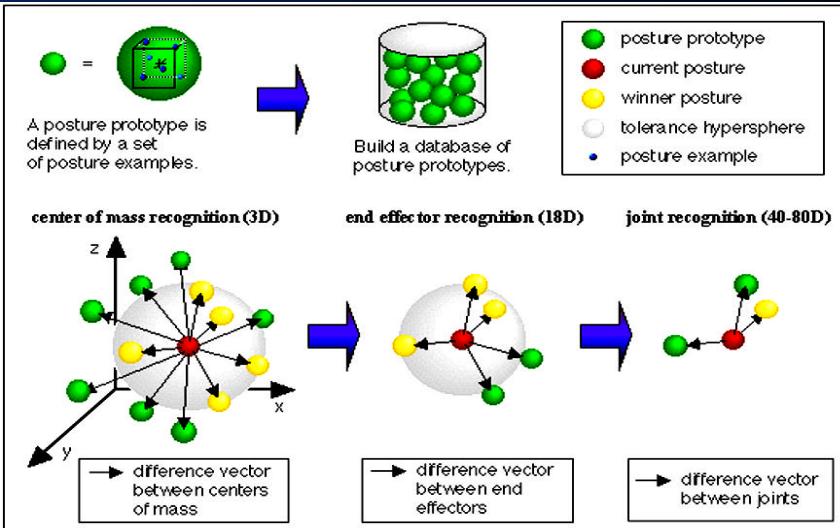
## **Characteristics**

- real-time perception: action recognition
- to simulate high-level bilateral communication between real people and autonomous humans
- actions modeled as combinations of primitives based on position of center of mass, body landmarks and postures of virtual skeleton

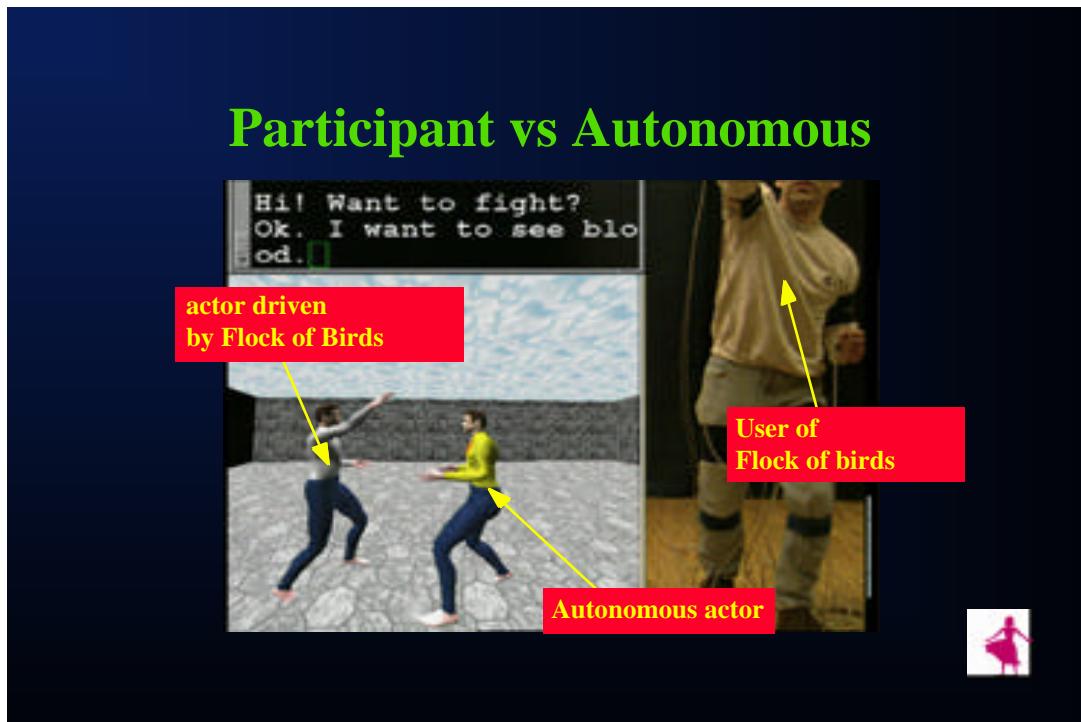


## System overview

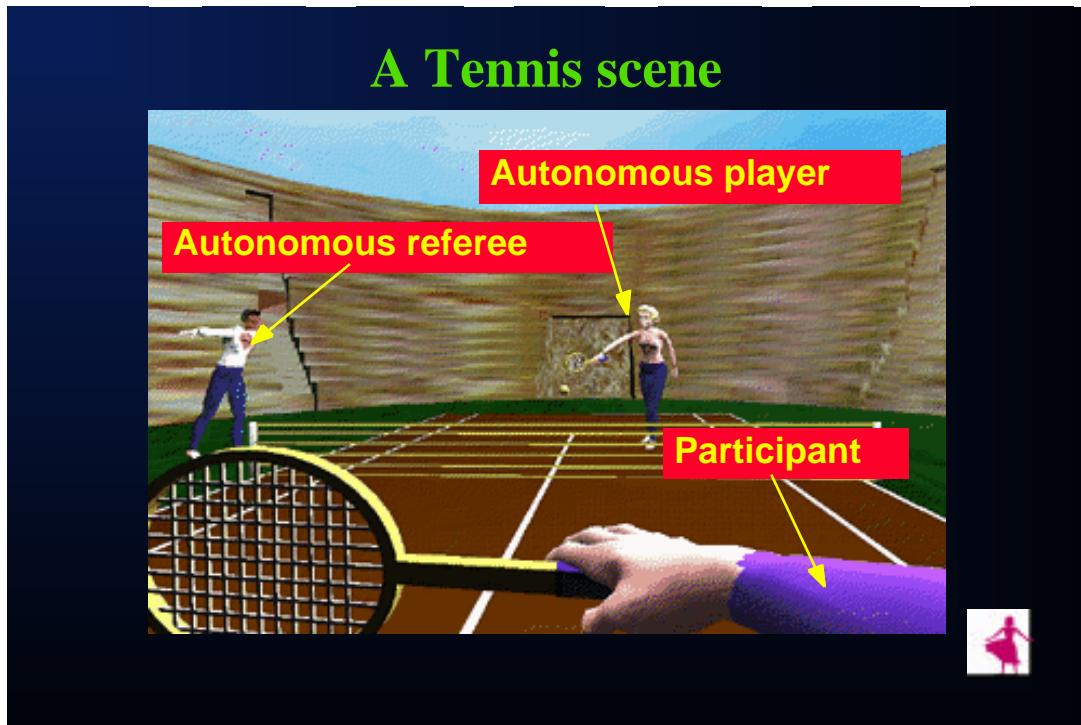
- gesture recognition
- posture recognition only if gesture recognition did not find a solution
- both use a recognition pyramid approach
- no neural net, but statistical approach



## Participant vs Autonomous



## A Tennis scene



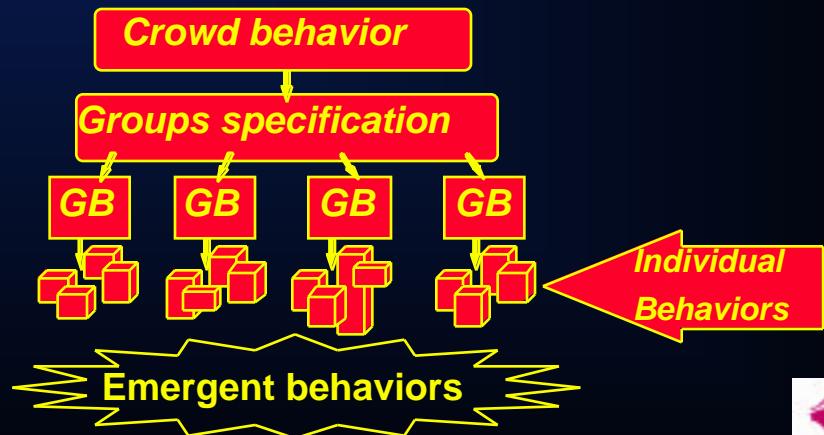
## Human Crowd Simulation

- Goal: Simulate in a realistic way the human crowd motion in specific environments.



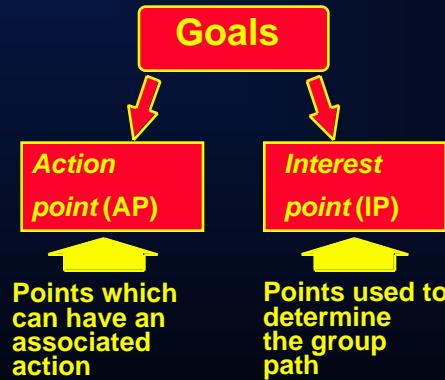
## Human Crowd Simulation

- Crowd behavior based on group behaviors

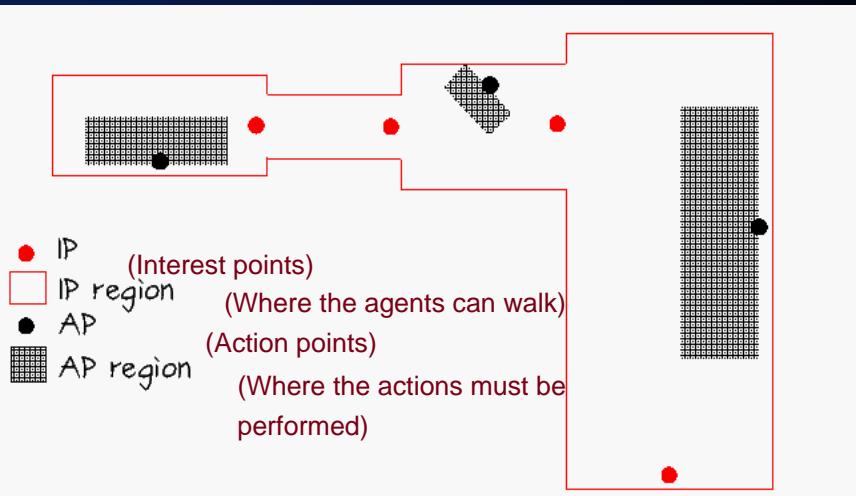


## Group Behavior: Seek goal

- Crowd motion based on goals



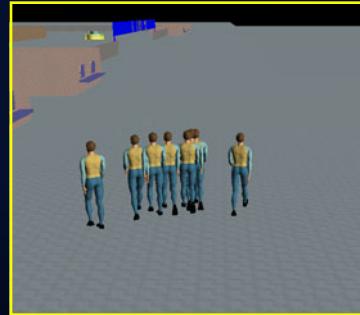
## Crowd Behavior: Seek goal



## Group Behaviors

- Seek goal
- Flocking motion

*(Ability to walk together)*



## Group Behaviors

- Seek goal
- Flocking motion
- Collision avoidance



## Group Behaviors

- Seek goal
- Flocking motion
- Collision avoidance
- Formation to reach a goal

*(in the associated region,  
in line, etc.)*



## Group Behaviors

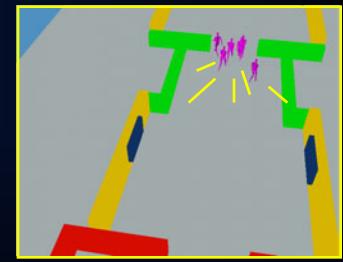
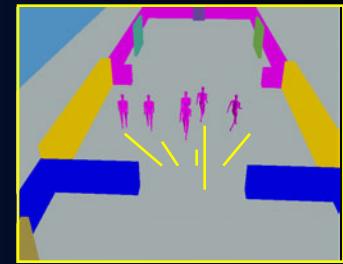
- Seek goal
- Flocking motion
- Collision avoidance
- Formation to reach a goal
- Following motion

*Follow the leader that can be  
an avatar or an agent from another group.*



## Group Behaviors

- Seek goal
- Flocking motion
- Collision avoidance
- Formation to reach a goal
- Following motion
- Dispersion/Agregation motion



## Virtual Actress in a Real Scene

- Real-time
- Mixing: Chroma-keying
- Shadows
- Collision detection (known environment)
- Camera coordination



# Augmented Reality

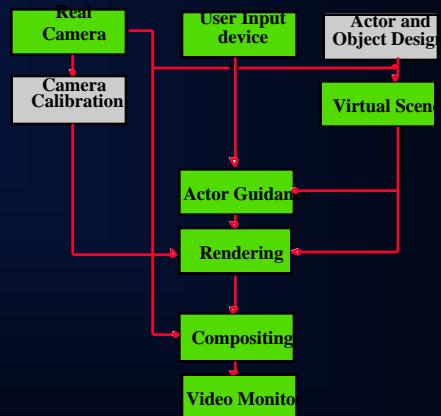
## Compositing

- ~ Chroma-keying
- ~ Correct Occlusion



# Augmented Reality

## Our AR system data flow diagram



Legend

Offline  
Operations

Real time  
Operations



## Augmented Reality

DRINK



## VLNET: Virtual Life NETwork

- Networked 3D Virtual Environments with Distant Partners
- 3D virtual models, sound, images, video
- 3 types of Virtual Humans: Participants, User-guided, and Autonomous
- Real Time Interaction



## Walking for User-guided Actors

- Participant updates eye position
- Incremental change in eye position
- walking cycle and time computed from velocity
- walking motor updates joints of virtual body



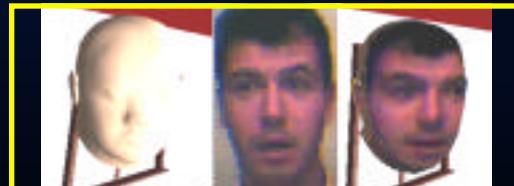
## Grasping for User-guided Actors

- Position and orientation of hand center provided by user
- arm motor used to compute joint angles
- normalized volume and inverse kinematics for arm positioning



## Facial Gestures

- Facial expression increases interaction
- texture mapping of real user's face
- Initially, background image stored
- At each frame, difference between background and current image is sent



## Video Texturing



## Applications



## Virtual Humans in VRML



# The Artificial Life of Virtual Humans

Daniel Thalmann  
Computer Graphics Lab  
Swiss Federal Institute of Technology (EPFL)  
CH 1015 Lausanne  
email: thalmann@lig.di.epfl.ch

## 1. Introduction

The purpose of this course part is to introduce the concepts of autonomous virtual humans. Our long-term objective at EPFL is the simulation of Virtual Life, which means Artificial Life in Virtual Reality<sup>1</sup>, as introduced in the book we have edited in 1994 and published by John Wiley. In the introductory chapter<sup>2</sup>, we explain that with Virtual Reality, new interfaces and 3D devices allow us a complete immersion into virtual worlds or at least a direct and real-time communication with them. At the same time, researchers have been able to create plants, trees, and animals and research in human animation has led to the creation of synthetic actors with complex motion. Moreover, a new field based on biology has tried to recreate the life with a bottom approach, the so-called *Artificial Life* approach. Now all these various approaches should be integrated in order to create truly Virtual Worlds with autonomous living beings, plants, animals and humans with their own behavior and real people should be able to enter into these worlds and meet their inhabitants. The ultimate objective is to build intelligent autonomous virtual humans able to take a decision based on their perception of the environment.

Perception is defined as the awareness of the elements of environment through physical sensation. In order to implement perception, virtual humans should be equipped with visual, tactile and auditory sensors. These sensors should be used as a basis for implementing everyday human behavior such as visually directed locomotion, handling objects, and responding to sounds and utterances. A simulation of the touching system should consist in detecting contacts between the virtual human and the environment.

At this time, nobody has developed a complete real-time human animation system based on the concept of perception-action. Although there is a deal of research in the area of autonomous agents particularly for robotics, the approach in Computer Animation is very different as the 3D Virtual Space is completely known from the computer. The essential is to decide which part of this space should be considered as known from the actor at a given time. Consequently, we don't need to use low-level mechanisms of recognition like artificial neural networks and we may investigate in much more complex situations. In particular, we may handle several sensors like vision, audition and touch at the same time and use them to simulate complex human behaviors like walking and grasping. We should also deal with a VR situation where the participant can dynamically modify the Virtual Space, which will modify the actors' behavior.

Although the term Artificial Life was not so popular in 1988, we were already involved in such a research and started a project of autonomous virtual humans based on synthetic vision. The concept of synthetic vision was first published<sup>3</sup> in 1990. Since this time, we have continued to developed the concept for navigation, walking, and playing tennis. We then added the concepts of synthetic audition and haptic system. More recently, we developed the Virtual Life Network (VLNET) allowing participants, guided and autonomous actors to share distributed Virtual Environments using Internet and ATM connections. Most of the results of this research will be presented in this course.

---

<sup>1</sup> Magnenat Thalmann N, Thalmann D, editors (1994) Artificial Life and Virtual Reality, John Wiley, Chichester

<sup>2</sup> Magnenat Thalmann N, Thalmann D Creating Artificial Life in Virtual Reality, in: Artificial Life and Virtual Reality, John Wiley, Chichester, 1994, pp.1-10

<sup>3</sup> O.Renault, N. Magnenat-Thalmann, D. Thalmann, A Vision-based Approach to Behavioural Animation, Journal of Visualization and Computer Animation, Vol.1, No1, 1990

## Contents

Theses notes on “The Artificial Life of Virtual Humans” are organized as follows. We first present a text to introduce the topics: “Introduction to the Artificial Life of Virtual Humans”. Then, two texts will present the problems of animating virtual humans:

- The Artificial Life of Synthetic Actors
- Realtime Deformable Actors
- The use of Space Discretization for Autonomous Virtual Humans

The main part of the course is dedicated to virtual sensors. An integrated text has been especially prepared for this SIGGRAPH course: **“Autonomous Virtual Actors based on Virtual Sensors”**. It is an integration of several papers:

- H.Noser, O.Renault, D.Thalmann, N.Magnenat Thalmann, Navigation for Digital Actors based on Synthetic Vision, Memory and Learning, Computers and Graphics, Pergamon Press, 1995  
 O.Renault, N.Magnenat-Thalmann, D.Thalmann, A Vision-based Approach to Behavioural Animation, Journal of Visualization and Computer Animation, Vol.1, No1, 1990  
 Z.Huang, R.Boulic, N.Magnenat Thalmann, D.Thalmann, A Multi-sensor Approach for Grasping and 3D Interaction, Proc. Computer Graphics International '95, Academic Press, pp.235-254  
 H.Noser, D.Thalmann, Synthetic Vision and Audition for Digital Actors, Proc. Eurographics '95, Maastricht, August 1995 pp.325-336.  
 D.Thalmann, Virtual Sensors: A Key Tool for the Artificial Life of Virtual Actors, Proc. Pacific Graphics '95, Seoul, Korea, August 1995, pp.22-40.

Then, we present four papers on the following topics:

- Playing Games through the Virtual Life Network
- A Model of Nonverbal Communication and Interpersonal Relationship between Virtual Actors
- Interacting with Virtual Humans through Body Actions
- A Model of Human Crowd Behavior: Group Inter-Relationship and Collision Detection Analysis

More details may be found in other related papers:

- A.Aubel, R. Boulic, D.Thalmann, Animated Impostors for Real-time Display of Numerous Virtual Humans, Proc. Virtual Worlds 98, Paris (to appear)  
 S.Bandi, D.Thalmann, Space Discretization for Efficient Human Navigation, Proc. Eurographics '98 (to appear)  
 S. Balcisoy, D. Thalmann, Hybrid Approaches to Interactions Between Real and Virtual Humans in Mixed Environments, Proc. Virtual Environments 98 (to appear)  
 L. Emering., R. Boulic, D.Thalmann, Virtual Reality Interactions with Live Participant's Action Recognition, Proc. Pacific Graphics 97, Seoul, IEEE Computer Society Press, 1997, pp.15-21.  
 L. Emering, R. Boulic, D. Thalmann, Interacting with Virtual Humans through Body Actions, IEEE Computer Graphics and Applications, 1998 , Vol.18, No1, pp8-11.  
 I. Pandzic, N. Magnenat Thalmann, T. Capin, D.Thalmann, Virtual Life Network: A Body-Centered Networked Virtual Environment, Presence, MIT, Vol. 6, No 6, 1997, pp. 676-686.  
 R.Boulic, R.Mas, D. Thalmann, Complex Character Positioning Based on a Compatible Flow Model of Multiple Supports, IEEE Transactions in Visualization and Computer Graphics , Vol.3, No3, 1997, pp.245-261  
 D. Thalmann, C. Babski, T. Capin, N. Magnenat Thalmann, I. Pandzic, Sharing VLNET Worlds on the WEB, Computer Networks and ISDN Systems Systems, Elsevier, Vol.29, 1997, pp.1601-1610.  
 R. Boulic, P. Becheiraz, L. Emering, and D. Thalmann, Integration of Motion Control Techniques for Virtual Human and Avatar Real-Time Animation, Proc. VRST '97, ACM Press, 1997, pp.111-118.  
 S.R. Musse, D. Thalmann, A Model of Human Crowd Behavior, Computer Animation and Simulation '97, Proc. Eurographics workshop, Budapest, Springer Verlag, Wien, 1997, pp.39-51.  
 I.Pandzic, T.Capin, E.Lee, N.Magnenat Thalmann, D.Thalmann, A Flexible Architecture for Virtual Humans in Networked Collaborative Virtual Environments, Proc. Eurographics '97, pp.177-188.  
 N. Magnenat Thalmann, D.Thalmann, Animating Virtual Actors in Real Environments, ACM Multimedia Systems, , Springer, Vol.5, No2, 1997, pp.113-125.

- S.Bandi, D.Thalmann, A configuration space approach for efficient animation of human figures, Proc. IEEE Nonrigid and Articulated Motion Workshop, Puerto Rico, IEEE CS Press (to appear)
- H. Noser, D. Thalmann, Sensor Based Synthetic Actors in a Tennis Game Simulation, Proc. Computer Graphics International '97, IEEE Computer Society Press, 1997, pp.189-198.
- S.Balcisoy, D.Thalmann, Interaction between Real and Virtual Humans in Augmented Reality, Proc. Computer Animation'97, IEEE CS Press, 1997, pp.31-38.
- R. Mas, R. Boulic, D.Thalmann, A Robust Approach for the Control of the Center of Mass with Inverse Kinetics, Computers and Graphics , Vol.20, No5, 1997, pp.693-702.
- T.Molet, Z.Huang, R. Boulic, D.Thalmann, An Animation Interface Designed for Motion Capture, Proc. Computer Animation'97, IEEE CS Press, 1997, pp.77-85.
- T.K.Capin, I.S.Pandzic, N.Thalmann, Virtual Human Representation and Communication in the VLNET Networked Virtual Environments, IEEE Computer Graphics and Applications, Vol.17, No2, 1997, pp.42-53.
- L. Emering, R. Boulic, S. Balcisoy, D. Thalmann, Real-Time Interactions with Virtual Agents Driven by Human Action Identification, Proc.Autonomous Agents '97, ACM Press.
- L. Emering, R. Boulic, S. Balcisoy, D. Thalmann, Multi-Level Modeling and Recognition of Human Actions Involving Full Body Motion, Proc.Autonomous Agents '97, ACM Press.
- R. Mas, R. Boulic, D. Thalmann, Extended Grasping Behavior for Autonomous Human Agents, Proc.Autonomous Agents '97, ACM Press.
- I.S.Pandzic, T. K. Capin, N. Magnenat Thalmann, D. Thalmann, Towards Natural Communication in Networked Collaborative Environments, FIVE '96, December 1996, pp.37-47.
- D. Thalmann, H. Noser, Z. Huang, Autonomous Virtual Actors based on Virtual Sensors, in: Creating Personalities (Ed. R.Trapp), Lecture Notes in Computer Science, Springer Verlag.
- T.K.Çapin, I.S.Pandzic, N.Magnenat Thalmann, D.Thalmann, A Dead-Reckoning Algorithm for Virtual Human Figures, Proc. VRAIS '97, pp.161-169.
- H.Noser, I.Pandzic, T.Capin, N.Magnenat-Thalmann, D.Thalmann, Playing Games through the Virtual Life Network, Proc.Alife'96, Nara, Japan.
- N.Magnenat Thalmann, D.Thalmann, Digital Actors for Interactive Television, Proc.IEEE, Special Issue on Digital Television, Part 2, July 1995, pp.1022-1031
- N.Magnenat Thalmann, D. Thalmann, The Artificial Life of Synthetic Actors, IEICE Transactions, Japan, invited paper, August 1993 Vol. J76-D-II, No.8, 1993, pp.1506-1514.
- P.Bécheiraz, D. Thalmann, The Use of Nonverbal Communication Elements and Dynamic Interpersonal Relationship for Virtual Actors, Proc. Computer Animation '96, IEEE Computer Society Press, 1996.
- R.Boulic, T.Capin, Z.Huang, L.Moccozet, T.Molet, P.Kalra, B.Lintermann, N.Magnenat-Thalmann, I.Pandzic, K.Saar, A.Schmitt, J.Shen, D.Thalmann, The HUMANOID Environment for Interactive Animation of Multiple Deformable Human Characters, Proc. Eurographics '95, 1995 pp.337-348.
- H.Noser, D. Thalmann, The Animation of Autonomous Actors Based on Production Rules, Proc. Computer Animation '96, IEEE Computer Society Press, 1996.
- N.Magnenat Thalmann, D.Thalmann, Creating Artificial Life in Virtual Reality in: Artificial Life and Virtual Reality, John Wiley, Chichester, 1994, pp.1-10
- H.Noser, D.Thalmann, Simulating the Life of Virtual Plants, Fishes and Butterflies in: Artificial Life and Virtual Reality, John Wiley, Chichester, 1994, pp.45-59
- N.Magnenat Thalmann, D.Thalmann, Complex Models for Animating Synthetic Actors, IEEE Computer Graphics and Applications, Vol.11, September 1991.
- S.Rezzonico, Z.Huang, R.Boulic, N.Magnenat Thalmann, D.Thalmann, Consistent Grasping Interactions with Virtual Actors Based on the Multi-sensor Hand Model, Virtual Environments '95, Springer, Wien, 1995, pp.107-118.
- D.Thalmann, Animating Autonomous Virtual Humans in Virtual Reality, Proc. IFIP Conference, Hamburg, September 1994, Vol.3, pp.177-184.
- R.Boulic, H.Noser, D.Thalmann, Automatic Derivation of Human Curved Walking Trajectories from Synthetic Vision, Proc. Computer Animation '94, IEEE Computer Society Press, 1994, pp.93-103.
- R.Boulic, H. Noser, D. Thalmann, Vision-based Human Free-Walking on Sparse Foothold Locations, Proc. Fourth Eurographics Workshop on Animation and Simulation, Barcelona, Sept. 1993, pp.173-191.
- D.Thalmann, Vision, Perception and Behaviour of Actors, Proc. BCS Conf. on Interacting with Images, British Computer Society, London, February 1993
- H.Noser, D.Thalmann, L-system-based Behavioral Animation, Proc. Pacific Graphics '93, Seoul, Korea, pp.133-146

# Introduction to the Artificial Life of Virtual Humans<sup>1</sup>

**Nadia Magnenat Thalmann**

*MIRALab, University of Geneva, Switzerland*

**Daniel Thalmann**

*Computer Graphics Lab, Swiss Federal Institute of Technology, Lausanne, Switzerland*

## **Virtual Worlds**

For about 20 years, computer-generated images have been created for films, generics and advertising. At the same time, scientific researchers, medical people, architects discovered the great potential of these images and used them to visualize the invisible or simulate the non-existing. It was the genesis of Virtual Worlds. However, these virtual worlds had two severe limitations:

- There was very little visual representation of living organisms or only very simple creatures in them
- Nobody could really enter into these worlds: the access to the virtual worlds was looking on 2D screens and 2D interaction.

Today, new interfaces and 3D devices allow us a complete immersion into these virtual worlds or at least a direct and real-time communication with them. This new way of immersion into the virtual worlds is called Virtual Reality. At the same time, researchers have been able to create plants, trees, and animals. Research in human animation has led to the creation of synthetic actors with complex motion. Moreover, a new field based on biology has tried to recreate the life with a bottom approach, the so-called Artificial Life approach. Now all these various approaches should be integrated in order to create truly Virtual Worlds with autonomous living beings, plants, animals and humans with their own behavior and real people should be able to enter into these worlds and meet their inhabitants.

## **Why Virtual Humans ?**

The fast development of multimedia communication systems will give a considerable impact to virtual worlds by allowing millions of people to enter into these worlds using TV networks. Among the applications of such virtual worlds with virtual humans, we can just mention:

- computer games involving people rather than cartoon-type characters
- computer-generated films which involve simulated people in simulated 3D worlds
- game simulations such as football games which show simulated people rather than cartoon-type characters.
- interactive drama titles in which the user can interact with simulated characters and hence be involved in a scenario rather than simply watching it.
- simulation based learning and training.
- virtual reality worlds which are populated by simulated people.

These applications will require:

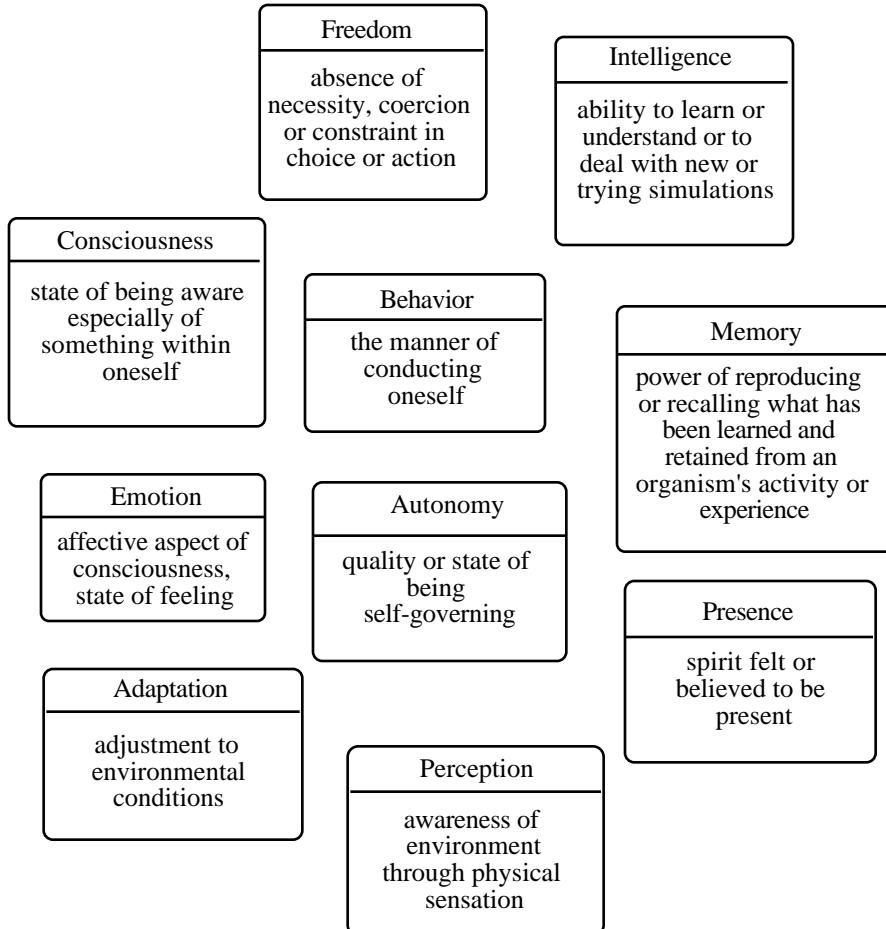
- realistic modeling of people's behavior, including interactions with each other and with the human user.
- realistic modeling of people's visual appearance, including clothes and hair

For the modeling of *behaviors*, the ultimate objective is to build *intelligent autonomous* virtual humans with *adaptation, perception* and *memory*. These virtual humans should be able to act *freely* and *emotionally*. They should be *conscious* and *unpredictable*, Finally, they should reinforce the concept of *presence*. But can we expect in the near future to represent in the computer the concepts of behavior,

---

<sup>1</sup> An expanded version of this text has appeared in: Magnenat Thalmann N and Thalmann D (eds) Artificial Life and Virtual Reality, John Wiley, 1994.

intelligence, autonomy, adaptation, perception, memory, freedom, emotion, consciousness, unpredictability, and presence? In this introductory part, we will try to define these terms and already identify research aspects in these concepts. In summary, virtual humans should have a certain number of qualities that are represented in Figure 1.



**Figure 1.** A few definitions.

## Conclusion

Artificial Life refers to all the techniques that try to recreate living organisms and creatures by computer, including the simulation of behavior processes which result from consciousness and emotions. Virtual Reality means the immersion of real humans in virtual worlds, worlds that are completely created by computer. This means interaction with objects from the virtual world, their manipulation, and the feeling that the human user is a real participant in the virtual world. In the future, most virtual worlds will become inhabited by virtual living creatures and users.

# The Artificial Life of Synthetic Actors<sup>1</sup>

**Nadia Magnenat Thalmann**

*MIRALab, University of Geneva, Switzerland*

**Daniel Thalmann**

*Computer Graphics Lab, Swiss Federal Institute of Technology, Lausanne, Switzerland*

This paper discusses the problem of simulating the artificial life of realistic synthetic actors. It first presents the development of the general concept of autonomous actors reacting to their environment and taking decisions based on perception systems, memory and reasoning. Then, we discuss facial animation techniques for synthetic actors. Finally, we show how to improve the appearance of synthetic actors by dressing them and designing hairstyles for them.

## 1. Introduction

The long-term objective of our research is the visualization of the simulation of the behavior of realistic human beings, called **synthetic actors**, in a given environment, interactively decided by the animator. The ultimate reason for developing realistic-looking synthetic actors is to be able to use them in virtually any scene that recreates the real world. However, a virtual scene -- beautiful though it may be -- is not complete without people.... Virtual people, that is. Scenes involving synthetic actors imply many complex problems we have been solving for several years <sup>1)</sup>. Behavioral techniques make possible the automating of high-level control of actors such as path planning. By changing the parameters ruling this automating, it is possible to give a different personality to each actor. This behavioral approach is a major step relatively to the conventional motion control techniques. Another complex objective is modeling human facial anatomy exactly, including movements to satisfy both structural and functional aspects of simulation. In order to improve the realism of synthetic actors, there are two important features to render: hair and clothes. Realistic hair has long been an unresolved problem because the great number of geometrical primitives involved and the potential diversity of the curvature of each strand of hair makes it a formidable task to manage. Dressing synthetic actors with complex clothes is also a challenge.

## 2. Animation System with Autonomous Actors

Most of the computer-generated films have been produced using traditional computer animation techniques like keyframe animation, spline interpolation, etc. Automatic motion control techniques (2) have been proposed, but they are strongly related to mechanics-based animation and do not take into account the behavior of characters. In fact, there are two ways of considering three-dimensional computer animation (3) and its evolution. The first approach corresponds to an extension of traditional animation methods. The animator uses the computer to assist him in the creation of keyframes and simple motions and deformations like stretching and squashing. The second approach corresponds to simulation methods based on laws of physics, physiology or even psychology. The purpose is not the same: traditional methods allow us to create three-dimensional characters with exaggerated movements while simulation methods are used to try to model a human behavior accurately. High level animation involving human beings and animals may be produced in this way.

Many authors have proposed methods to implement motion of articulated bodies. Some methods come from robotics like inverse kinematics and dynamics. Inverse kinematics permits direct specification of end point positions (e.g., a hand or foot). There are two major problems in inverse kinematics, finding any solution that will achieve the desired goal and, finding the most desirable solution. Inverse kinematics can be used as constraints with dynamic simulations (4). Dynamics has been used to automatically generate realistic motion (5,6) that successfully animates the motion of chains, waves,

---

<sup>1</sup> An expanded version of this text has been published in: *IEICE Transactions*, Japan, Vol. J76-D-II, No.8, 1993, pp.1506-1514.

spacecraft, and automobiles. The problem is that force/torque control is not intuitive. The course of a simulation is completely determined by the objects' initial positions and velocities, and by forces applied to the objects along the way. It just solves the initial value problems. How to return some level of control to the animator is one of the most difficult issues in dynamic simulation.

In task-level animation (7,8), the animator specifies what the synthetic actor has to do, for instance, "jump from here to there". For example, Witkin and Kass (9) describe a spacetime constraint system in which constraints and objectives are defined over Spacetime, referring to the set of all forces and positions of all of a creature's degrees of freedom from the beginning to the end of an animation sequence. Cohen (10) takes this concept further and uses spacetime window to control the animation interactively. But, task-level animation raises a problem: how to introduce individual differences into the generic activities which are generated automatically? In the task of walking, everybody walks more or less the same way, following more or less the same laws. It is the "more or less" which is difficult to model. Even the same person does not walk the same way everyday. If he is tired, or happy, or has just received some good news, the way of walking will appear somewhat different. As in traditional animation, an animator can create a lot of keyframes to simulate a tired character walking, but this is a very costly and time-consuming task. To individualize human walking, we have developed (11) a model built from experimental data based on a wide range of normalized velocities (see Fig.1).

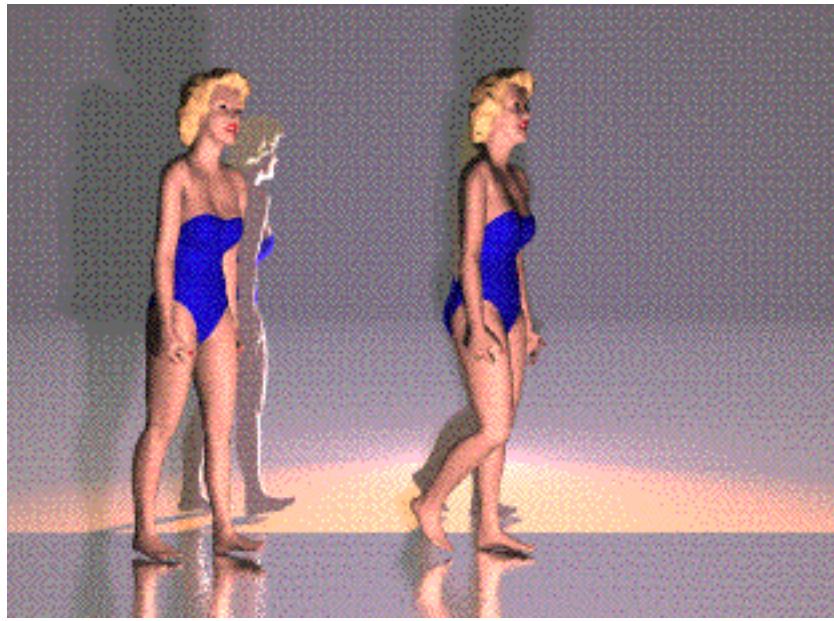


Fig.1. Biomechanics walking

The model is structured on two levels. At a first level, global spatial and temporal characteristics (normalized length and step duration) are generated. At the second level, a set of parameterized trajectories produce both the position of the body in space and the internal body configuration, in particular the pelvis and the legs. This is performed for a standard structure and an average configuration of the human body. The experimental context corresponding to the model is extended by allowing continuous variation of the global spatial and temporal parameters for altering the motion to try to achieve the effect desired by the animator. The model is based on a simple kinematics approach designed to preserve the intrinsic dynamic characteristics of the experimental model. But what is important is that this approach allows individualization of the walking action in an interactive real-time context in most cases.

Virtual humans are not only visual. They have a behavior, perception, memory and some reasoning. Behavior is often defined as the way animals and humans act, and is usually described in natural language terms which have social, psychological or physiological significance, but which are not

necessarily easily reducible to the movement of one or two muscles, joints or end effectors. In fact the behavior of any living creature may be simulated. Reynolds (12) introduced the term and the concept of behavioral animation in order to study in detail the problem of group trajectories: bird flocks, herds of land animals and fish schools.

A typical human behavioral animation system, is based on the three key components:

- the locomotor system
- the perceptual system
- the organism system

A locomotor system is concerned with how to animate physical motions of one or more actors in their environment. This is the control part of the system. A perceptual system is concerned with perceiving the environment. According to Gibson (13), we may consider five perceptual systems: basic orienting system, auditory system, haptic system, taste-smell system, and visual system. The organism system is concerned with rules, skills, motives, drives and memory. It may be regarded as the brain of the actor. Although many techniques have been developed for the control of articulated bodies; they are generally applied to much simpler systems than humans. For a general **locomotor system**, only a combination of various techniques may result in a realistic motion with a relative efficiency. Consequently, our locomotor system is based on several integrated methods and their blending: keyframe, inverse kinematics, direct/inverse dynamics (Fig.2) and biomechanics-based walking, we integrate them using a blending approach.



Fig.2. Dynamics-based motion

A **perceptual system** is concerned with perceiving the environment. The most important perceptual subsystem is the vision system. The originality of our approach in this wide research area is the use of a synthetic vision as a main information channel between the environment and the actor (14). We model the actor brain with a memory (essentially visual) and a limited reasoning system allowing the actor to decide his motion based on information. The movement decision procedure is the central coordinator to determine further movement. The controller is the thinking part of our system or the **organism system**. It makes decision and perform the high-level actions.

#### 4. Facial Animation

Computer modeling and animation of synthetic faces has attained a considerable attention recently. Because the human face plays the most important role for identification and communication, realistic construction and animation of the face is of immense interest in the research of human animation. The

ultimate goal of this research would be to model exactly the human facial anatomy and movements to satisfy both structural and functional aspects. However, this involves many problems to be solved concurrently. The human face is a very irregular structure, which varies from person to person. The problem is further compounded with its interior details such as muscles, bones and tissues, and the motion which involves complex interactions and deformations of different facial features. The properties of facial expressions have been studied for 25 years by Psychologist Ekman, who proposed a parameterization of muscles with their relationships to emotions: the Facial Action Coding System (FACS) (15). FACS describes the set of all possible basic actions performable on a human face. Various facial animation approaches have been proposed: parameterized models (16), muscle model for facial expressions (17,18), human performances (19, 20, 21).

For our facial deformations, we have extended (22) the concept of Free Form Deformations (FFD) introduced by Sederberg and Parry (23), a technique for deforming solid geometric models in a free-form manner. The free-form deformations correspond to the lower level of our multi-layer system SMILE (24) where, at each level, the degree of abstraction increases. Each successive layer defines entities from a more abstract point of view, starting with muscles deformations, and working up through phonemes, words, sentences, expressions, and emotions. The high level layers are the most abstract and specify "what to do", the low level layers describe "how to do". The top level requires direct input from the animator in the form of global abstract actions and their duration, corresponding to the intuitive and natural specifications for facial animation. A language HLSS for synchronizing speech, emotions and eye motions is developed to provide a way to naturally specify animation sequences. Figure 3 show an example of facial expression.



Fig.3. Facial animation

## 5. Hair

The difficulties of rendering hair result from the large number and detailed geometry of the individual hairs, the complex interaction of light and shadow among the hairs, and the small scale of the hair width in comparison with the rendered image. For the hair rendering in our system, an implementation module of the shadow buffer algorithm (25) has been added to a raytracing program. The process is step by step. First, the shadow of the scene is calculated for each light source, as well as for the light sources for the hair shadows. The hair shadows are calculated for the object surface and individually for each hair. Finally the hair style is blended into the scene, using all shadow buffers. The result is an image with a three-dimensional realistic hair style rendering where complex shadow interaction and highlight effects can be seen and appreciated. Fig.4 shows an animation with a synthetic actress with a hairstyle.



Fig.4. Synthetic actress with hairstyle

## 6. Clothes

Cloth animation in the context of human animation involves the modeling of garments on the human body and their animation. In our film "Rendez-vous à Montréal" (26) featuring Humphrey Bogart and Marilyn Monroe, clothes were simulated as a part of the body with no autonomous motion. To create autonomous clothes (27), we work as a tailor does, designing garments from individual two-dimensional panels seamed together. The resulting garments are worn by and attached to the synthetic actors. When the actors are moving or walking in a physical environment, cloth animation is performed with the internal elastic force (28) and the external forces of gravity, wind, and collision response. When a collision is detected, we pass through the second step where we act on the vertices to actually avoid the collision. For this collision response, we have proposed the use of the law of conservation of momentum for perfectly inelastic bodies. This means that kinetic energy is dissipated, avoiding the bouncing effect. We use a dynamic inverse procedure to simulate a perfectly inelastic collision. The constraints that join different panels together and attach them to other objects are very important in our case. Two kinds of dynamic constraints (29) are used during two different stages. When the deformable panels are separated, forces are applied to the elements in the panels to join them according to the seaming information. The same method is used to attach the elements of deformable objects to other rigid objects. Fig.5 shows an example of a dressed synthetic actress.



Fig.5. Dressed synthetic actress

## Conclusion

Modeling humans using computers is a very complex task (30). It will take years before we are able to represent synthetic actors who look and behave realistically. And if these actors are not to behave all in the same way, we will have to introduce interactive psychological description capabilities. New problems will arise: how to model the personality, the know-how, the common sense, the mind ? We need concrete and mathematical models of domains which as yet far from being formally described. This may be the challenge for the computer modeling of human in the coming century.

## References

1. Magnenat-Thalmann N, Thalmann D (1991) Complex Models for Animating Synthetic Actors, IEEE Computer Graphics and Applications, Vol.11, No5, pp.32-44.
2. Wilhelms J. Toward Automatic Motion Control, IEEE Computer Graphics and Applications, Apr., 1977
3. Magnenat Thalmann N, Thalmann D (1990) Computer Animation: Theory and Practice, Springer-Verlag, Tokyo.
4. Isaacs PM, Cohen MF (1987) Controlling Dynamic Simulation with Kinematic Constraints, Behavior Functions and Inverse Dynamics, Proc. SIGGRAPH'87, Computer Graphics, Vol.21, No4, pp.215-224.
5. Armstrong, B., and Green,M. The dynamics of articulated rigid bodies for purposes of animation. In Proceedings of Graphics Interface (May,1986)
6. Wilhelms,J. and Barsky,B. Using Dynamic Analysis to Animate Articulated Bodies Such As Humans and Robots, Proc. Graphics Interface, 1985.
7. Badler NI, Webber BL, Kalita J, Esakov J (1989) Animation from Instructions, in: Make Them Move, Morgan Kaufman, 1989, pp.51-93.
8. Zeltzer D (1987) Task-level Graphical Simulation: Abstraction, representation, and Control, in: Make Them Move, Morgan Kaufman, 1989, pp.3-33.
9. Witkin A, Kass M (1988) Spacetime Constraints, Proc. SIGGRAPH '88, pp.159-168.
10. Cohen MF Interactive Spacetime Control for Animation, Proc. Siggraph'92, pp.293-302.
11. Boulic R, Magnenat-Thalmann N, Thalmann D (1990) A Global Human Walking Model with real time Kinematic Personification, The Visual Computer, Vol.6, No6, pp.344-358.
12. Reynolds C (1987) Flocks, Herds, and Schools: A Distributed Behavioral Model, Proc.SIGGRAPH '87, pp.25-34
13. Gibson JJ, The Senses Considered as Perceptual Systems, Houghton Mifflin, 1966, Boston
14. Renault, O., Thalmann,N. M., Thalmann,D. A Vision-Based Approach to Behavioral Animation, Visualization and Computer Animation, Vol.1, No. 1, 1990, pp.18-21.
15. Ekman P, Friesen W (1978) Facial Action Coding System, Consulting Psychologists Press, Palo Alto.
16. Parke FI (1982) Parameterized Models for Facial Animation, IEEE Computer Graphics and Applications, Vol.2, No 9, pp.61-68.
17. Waters K (1987) A Muscle Model for Animating Three-Dimensional Facial Expression, Proc. SIGGRAPH '87, Computer Graphics, Vol.21, No4, pp.17-24.
18. Magnenat-Thalmann N, Primeau E, Thalmann D (1988), Abstract Muscle Action Procedures for Human Face Animation, The Visual Computer, Vol.3, No.5, pp. 290-297.
19. Terzopoulos D, Waters (1991) Techniques for Realistic Facial Modeling and Animation, Computer Animation '91, Springer, Tokyo, pp.59-74.
20. deGraf B (1989) in State of the Art in Facial Animation, SIGGRAPH '89 Course Notes No. 26, pp. 10-20.
21. Williams L (1990), Performance Driven Facial Animation, Proc SIGGRAPH '90, pp.235-242.
22. Kalra P, A. Mangili, N. Magnenat Thalmann, D. Thalmann, "Simulation of Facial Muscle Actions Based on Rational Free Form Deformations", Proc. Eurographics '92, Cambridge, pp. 59-69.
23. Sederberg TW, Parry SR (1986), Free Form Deformation of Solid Geometric Models, Proc. SIGGRAPH '86, pp. 151-160.
24. Kalra P, Mangili A, Magnenat-Thalmann N, Thalmann D (1991) SMILE : A Multilayered Facial Animation System, in: Kunii TL (ed) Modeling in Computer Graphics, Springer, Tokyo, pp. 189-198.
25. Williams L (1978) Casting Curved Shadows on Curved Surfaces, Proc. SIGGRAPH '78, Computer Graphics, Vol.12, No3, pp.270-274.
26. Magnenat-Thalmann N, Thalmann D (1987) The Direction of Synthetic Actors in the Film Rendez-vous à Montréal, IEEE Computer Graphics and Applications, Vol.7, No12, pp.9-19.
27. Carignan M, Yang Y, Magnenat Thalmann N, Thalmann D (1992) Dressing Animated Synthetic actors with Complex Clothes, Proc. SIGGRAPH'92, Computer Graphics, Vol. 26, No 2, Chicago, pp. 99-104.
28. Terzopoulos Demetri, Platt John, Barr Alan, Fleischer Kurt. Elastically Deformation Models, Proc. SIGGRAPH'87, Computer Graphics, 1987, Vol.21, No.4, pp.205-214.
29. Barzel R, Barr Alan H. A Modeling System Based on Dynamic Constraints. In Proc. SIGGRAPH '88, Computer Graphics, Vol. 22, No4, 1988, pp.179-188.

# Realtime Deformable Actors

Eric Chauvineau, Shen Jianhua, Daniel Thalmann  
EPFL, Computer Graphics Lab

Our realtime approach is based on contours deformation. First we compute these contours points in a given position of the body (we call it ‘initial position’). At this point, we don’t compute anymore B-spline patches and triangulate these surfaces: contours points are used directly for body triangles mesh. As we already explained, each contour belongs to one body part. For convenient reasons, we still consider these different body parts and use them to construct the final 3D surface.

Figure 1 shows the new triangles mesh made by contours computed in ‘initial position’. Hands, feet and head are not considered in our method : they are represented by undeformable triangles mesh.

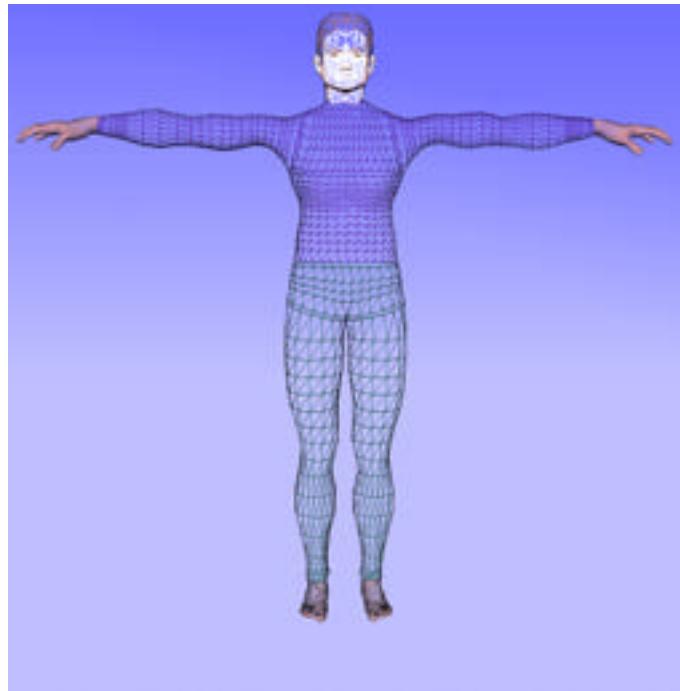


Figure 1 Body mesh in initial position

Then, during realtime animation, we deform directly contours still using joint angle driven configuration and boundary driven configuration methods. By this way, we avoid contours computation from implicit surfaces, which is, with B-spline patches triangulation, the most time-consuming operation.

## Implementation

In order to have an efficient implementation, we use the Performer library. This realtime graphics toolkit enables application to achieve maximum graphics performance from all Silicon Graphics workstations. It is easy with this library to create a 3D scene with many objects coming from different modelers. But, except morphing, no deformation capabilities have been proposed by Performer.

Consequently, we define our own structure compatible with Performer data arrangement (linear). By using index array to define triangle meshes, we avoid points duplication and make efficient memory management. It is also easy with Performer environment to apply texture on a 3D surface. By using our model configuration, we can apply different textures on each body part. Using hardware texture mapping, performance is the same than without texture, but enable texturing is a good way to increase surface realism. Figure 2 shows our implementation of deformable body using Performer library.



Figure 2 Body deformation based on Performer library a) without texturing, b) with texturing

### Level-of-detail management

In realtime applications, an efficient way to reduce computing time is to manage different levels-of-detail (LOD) for one object: considering the distance between the camera and our object, we display this object in the best resolution for this distance. As we know, the body envelope is one unique surface composed of eleven triangles meshes. Dimensions of each triangle mesh is given by the number of contours and the number of points per contour. By modify these numbers, it is possible to get different LODs for each body part and thus for all the body. Figure 3 shows three different LODs of the body surface which can be used in realtime applications using Performer toolkit.

### Applications

Realtime deformable actors are a key issue for many multimedia applications: virtual actors for digital television [1], games [2] and VR applications. In order to provide maximum immersion for using virtual actors, it is important to use the most realistic human representation possible with real time considerations. Therefore, it is preferred to use the deformed body surfaces as envelopes attached to the human skeleton, rather than simple representations such as basic geometric primitives. It is also necessary to simulate the articulations of the body, i.e. the joints to connect the body limbs with realistic constraints. In particular, we use them extensively in the VLNET system. The VLNET system [3] supports a networked shared virtual environment that allows multiple users to interact with each other and their surrounding in real time. The users are represented by 3D virtual human actors, with realistic appearances and articulations. The actors have similar appearance and behaviors with the real humans, to support the sense of presence of the users in the environment. In the environment, each participant is represented by a 3D virtual body that resembles the user. Each user sees the virtual environment through the eyes of her body. The user controls the movement of her virtual actor by various input devices, such as mouse, SpaceBall. Stereo display devices such as shutter glasses and head-mounted display can also be used for more realistic feeling in the environment. We also include additional virtual autonomous actors in the environment. The autonomous actors are connected to the system in the same way as human participants, and enhance the usability of the environment by

providing services such as replacing missing partners, providing services such as helping in navigation. As these virtual actors are not guided by the users, they should have sufficient behaviors to act autonomously to accomplish their tasks. This requires building behaviors for motion, as well as appropriate mechanisms for interaction.

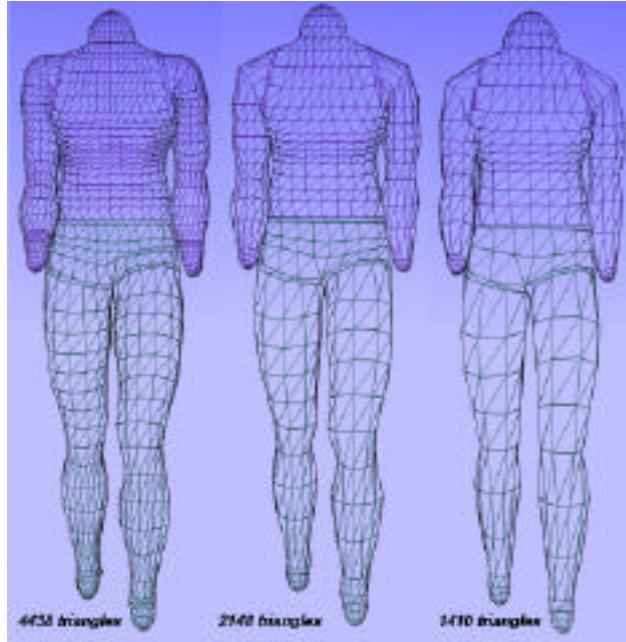


Figure 3 Different LODs for body surface

## References

- 1 Magnenat Thalmann, N., Thalmann, D. "Digital Actors for Interactive Television", *Proc. IEEE*, Special Issue on Digital Television, Part 2, July 1995, pp.1022-1031.
- 2 Noser, H., Çapin, T., Pandzic, I., Magnenat Thalmann, N., Thalmann, D. "Playing Games using the Virtual Life Network", *Proc. Alife '96*, Nara, Japan, 1996.
- 3 Pandzic, I., Çapin, T. Magnenat Thalmann, N., Thalmann, D. "VLNET: A Networked Multimedia 3D Environment with Virtual Humans", *Proc. Multi-Media Modeling MMM '95*, Singapore, 1995, pp.21-32.

# The use of Space Discretization for Autonomous Virtual Humans<sup>1</sup>

Srikanth Bandi and Daniel Thalmann  
Computer Graphics Lab  
Swiss Federal Institute of Technology  
e-mail: {srik, thalmann}@lig.di.epfl.ch

A motion planning algorithm for automatic path computation for an autonomous virtual human in an environment of obstacles is presented. The environment is discretized and an optimum path in terms of number of cells is computed using A\* search technique.

**Keywords:** motion planning, floodfill algorithm

## 1. INTRODUCTION

Motion of articulated bodies is important for simulating autonomous behavior of virtual humans. The problem of simulating motion in an articulated chain is well studied using dynamic as well as kinematics techniques. This simulates motion in immediate neighborhood of a virtual human. However, making human move long distances across complex 3D environments did not receive much attention. Endowing human model with such a capability greatly enhances autonomous navigation behavior of human models.

The algorithm presented here constructs a discrete representation of a 3D scene. The discretization is carried out using framebuffer hardware. Slices of the scene are displayed on a raster screen and pixels are read. The pixel values indicate occupancy status in a 3D cell grid. A search is made in the grid for an optimal path between two given cells. First, 2D path finding algorithm is presented and then extended to 3D.

## 2. Optimal Path in 2D grid

There are two steps in computing optimal path: (1) Generation of adjacent cells of all cells using a method known as floodfill. (2) Generation of path.

### 2.1 Generation of cells

The basic floodfill algorithm for robot motion planning is described in [2]. In Figure 5 a cell path from A to B is required in a planar grid. The grid contains a hole region as indicated by a dark patch with a protrusion. In the first step, cell A is entered into an initially empty list. This is the root or starting cell and has zero cost. Each time, first cell from the list is removed and all its four orthogonal neighbor cells are generated and added at the end of the list. However, the cells which were entered in the list in previous iterations are omitted. The cells occupied by hole regions are also ignored and initialized with a very large cost. The cells are generated as nodes in a quad-tree in breadth-first fashion. Cost of a cell is defined as its depth from the root cell. The cell generation continues until the goal cell B is reached.

In Figure 5 cell generation is shown as thin lines. It expands as a square wave front starting from A. Instead of terminating at B, all cells are generated for purpose of illustration.

In order to reduce number of cells generated, A\* technique is used. The cost function  $f(n)$  of a cell n is given by,

$$f(n) = g(n) + h(n), \quad g(n), h(n) \geq 0$$

---

<sup>1</sup> published as a Video paper in Proc. Autonomous Agents '98, Minneapolis.

$g(n)$  is depth of  $n$  in the quad-tree starting at the root A.  $h(n)$ , the heuristic cost, is computed as below:

$$h(n) = xb-xn + yb-yn$$

where  $(xn,yn)$  and  $(xb,yb)$  are integral cell coordinates of cells  $n$  and B respectively.  $h(n)$  is a lower bound on path length between  $n$  and B in a quad-tree expansion. In every iteration, the cell with the lowest cost is expanded next.

### 3.2 Path generation

Cell B is entered in an initially empty path list. At each step last cell in the list is selected and its orthogonal cell neighbor with the lowest cost is searched and appended to the path list. This is like tracing backwards across the wave front generated earlier. Eventually cell A which is the starting cell with zero cost is reached.

### 3.3 Path smoothening

In the cell path the cells where path changes direction are identified. These are node cells (C,D,E in Figure 5). The alternate nodes are connected from the first cell using Digital Differential Analyzer [3]. DDA enumerates cells along a straight line connecting two cells. If such newly enumerated cells fall on hole regions original path is retained. The smoothed path is shown in **Figure 4**.

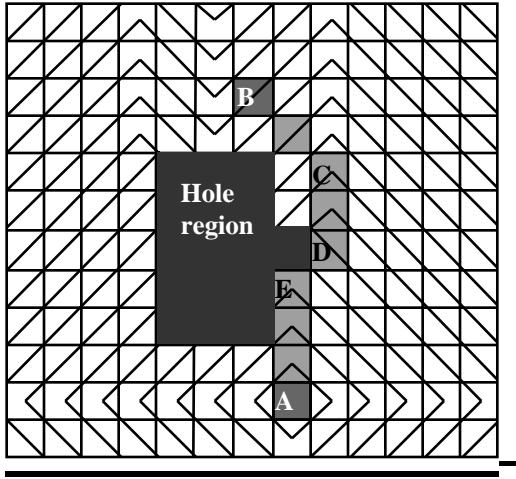


Figure 5. Floodfill from A to B

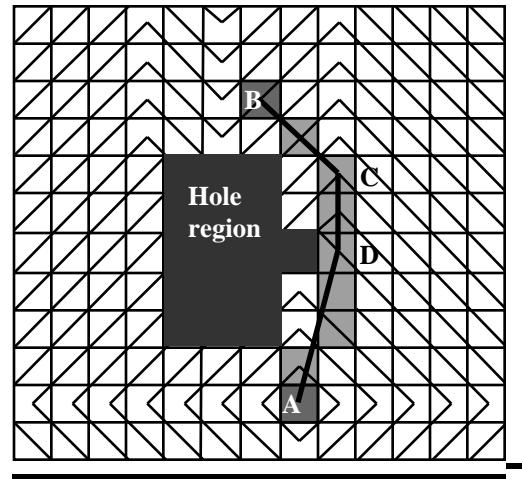


Figure 4. Path smoothening

## 4. Extension to human model in 3D

The motion planning algorithm should take human dimensions into account for 2D and 3D. This is expressed in terms of number of cells. A maximum limit on number of cells a human can cross or climb is set. Then 3D scenario is converted into standard 2D problem. For small obstacles such as steps in a staircase additional vertical dimension becomes immaterial so long as human can step on them. The obstacles the human can not overcome, such as walls, large gaps are projected as hole regions in 2D and avoided.

Once a path is generated, the human model is made to walk along the path. At each step, the legs should avoid obstacles in the immediate neighborhood. An efficient kinematics technique described in [1] is used.

## 5. Example:Entering Super Market

Super market scene (Figure 3) is created in Inventor file format. It is displayed using IRIS Performer on the screen and discretized. The discretization can be time consuming, but is performed only once. On SGI ONYX (200Mhz R4400), it takes around 2 minutes. The path generation is almost instantaneous.



**Figure 3. Super market scene**

## 6. Conclusion

The motion planning algorithm based on discretization is very efficient for interactive applications. However, discrete cells inaccurately represent objects which are curved or planes not orthogonal to coordinate axes. Motion planning in dynamically changing environment is also to be explored further.

## 7. Acknowledgments

The authors would like to thank Olivier Aune for creating scene files. The research was partly supported by the Swiss National Foundation for Scientific Research.

## 8. References

1. Bandi S., and Thalmann, D., A Configuration Approach for Efficient Animation of Human Figures, IEEE Non-Rigid and Articulated Motion Workshop, Puerto-Rico, June 15, 1997
2. Lengyel, J et al., Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware, Computer Graphics Proceedings, Annual Conference Series (August, 1990), ACM SIGGRAPH, 327-335
3. Rogers, D.F., Procedural elements for computer graphics, McGraw-Hill, pp.30 -39, 1985.

# Autonomous Virtual Actors based on Virtual Sensors

**Daniel Thalmann, Hansrudi Noser, Zhiyong Huang**  
 Computer Graphics Lab, Swiss Federal Institute of Technology  
 Lausanne, Switzerland

## Abstract

We present current research developments in the Virtual Life of autonomous synthetic actors. After a brief description of the perception action principles with a few simple examples, we emphasize the concept of virtual sensors for virtual humans. In particular, we describe in details our experiences in implementing virtual vision, tactile and audition. We then describe perception-based locomotion, a multisensor based method of automatic grasping and vision-based ball games.

## Introduction

As a virtual world is completely generated by computer, it expresses itself visually, with sounds and feelings. Virtual worlds deal with all the models describing physical laws of the real world as well as the physical, biological, and psychological laws of life. Virtual Life is linked to problems in artificial life but differs in the sense that these problems are specific to virtual worlds. It does not deal with physical or biological objects in real life but only with the simulation of biological virtual creatures. Virtual Life is at the intersection of Virtual Reality and Artificial Life (Magnenat Thalmann and Thalmann 1994), it is an interdisciplinary area strongly based on concepts of real-time computer animation, autonomous agents, and mobile robotics. Virtual Life cannot exist without the growing development of Computer Animation techniques and corresponds to the most advanced concepts and techniques of it.

In this paper, we first review the state-of-the-art in this emerging field of Virtual Life, then we emphasize the aspect of Virtual Life of Synthetic Actors, relating the topics to the previous chapters. The objective is to provide autonomous virtual humans with the skills necessary to perform stand-alone role in films, games (Bates et al. 1992) and interactive television (Magnenat Thalmann and Thalmann 1995). By autonomous we mean that the actor does not require the continual intervention of a user. Our autonomous actors should react to their environment and make decisions based on perception, memory and reasoning. With such an approach, we should be able to create simulations of situations such as virtual humans moving in a complex environment they may know and recognize, or playing ball games based on their visual and tactile perception.

## State-of-the-Art in Virtual Life

This kind of research is strongly related to the research efforts in behavioral animation as introduced by Reynolds (1987) to study the problem of group trajectories: flocks of birds, herds of land animals and fish schools. This kind of animation using a traditional approach (keyframe or procedural laws) is almost impossible. In the Reynolds approach, each bird of the flock decides its own trajectory without animator intervention. Reynolds introduces a distributed behavioral model to simulate flocks. The simulated flock is an elaboration of a particle system with the simulated birds being the particles. A flock is assumed to be the result of the interaction between the behaviors of individual birds. Working independently, the birds try both to stick together and avoid collisions with one another and with other objects in their environment. In a module of behavioral animation, positions, velocities and orientations of the actors are known from the system at any time. The animator may control several global parameters: e.g. weight of the obstacle avoidance component, weight of the convergence to the goal, weight of the centering of the group, maximum velocity, maximum acceleration, minimum distance between actors. The animator provides data about the leader trajectory and the behavior of other birds relatively to the leader. A computer-generated film has been produced using this distributed behavioral model: *Stanley and Stella*.

Haumann and Parent (1988) describe behavioral simulation as a means to obtain global motion by simulating simple rules of behavior between locally related actors. Letebridge and Ware (1989) propose a simple heuristically-based method for expressive stimulus-response animation. Wilhelms (1990) proposes a system based on a network of sensors and effectors. Ridsdale (1990) proposes a method that guides lower-level motor skills from a connectionist model of skill memory, implemented as collections of trained neural networks.

We should also mention the huge literature about autonomous agents (Maes 1991) which represents a background theory for behavioral animation. More recently, genetic algorithms were also proposed by Sims (1994) to automatically generate morphologies for artificial creatures and the neural systems for controlling their muscle forces. Tu and Terzopoulos (1994) described a world inhabited by artificial fishes

## **Virtual Sensors**

### **Perception through Virtual Sensors**

The problem of simulating the behavior of a synthetic actor in an environment may be divided into two parts: 1) provide to the actor a knowledge of his environment, and 2) to make react him to this environment.

The first problem consists of creating an information flow from the environment to the actor. This synthetic environment is made of 3D geometric shapes. One solution is to give the actor access to the exact position of each object in the complete environment database corresponding to the synthetic world. This solution could work for a very "small world", but it becomes impracticable when the number of objects increases. Moreover, this approach does not correspond to reality where people do not have knowledge about the complete environment.

Another approach has been proposed by Reynolds (1987): the synthetic actor has knowledge about the environment located in a sphere centered on him. Moreover, the accuracy of the knowledge about the objects of the environment decreases with the distance. This is of course a more realistic approach, but as mentioned by Reynolds, an animal or a human being has always around him areas where his sensitivity is more important. Consider, for example, the vision of birds (birds have been simulated by Reynolds), they have a view angle of 300° and a stereoscopic view of only 15°. The sphere model does not correspond to the sensitivity area of the vision. Reynolds goes one step further and states that if actors can see their environment, they will improve their trajectory planning.

This means that the vision is a realistic information flow. Unfortunately, what is realistic to do for a human being walking in a corridor seems unrealistic to do for a computer. However, using hardware developments like the **graphic engine** (Clark 1982), it is possible to give a geometric description of 3D objects together with the viewpoint and the interest point of a synthetic actor in order to get the vision on the screen. This vision may then be interpreted like the synthetic actor vision. This is our approach as described in this paper. More generally, in order to implement perception, virtual humans should be equipped with visual, tactile and auditory sensors. These sensors should be used as a basis for implementing everyday human behaviour such as visually directed locomotion, handling objects, and responding to sounds and utterances. For synthetic audition, in a first step, we model a sound environment where the synthetic actor can directly access to positional and semantic sound source information of a audible sound event. Simulating the haptic system corresponds roughly to a collision detection process. But, the most important perceptual subsystem is the vision system. A vision based approach for virtual humans is a very important perceptual subsystem and is for example essential for navigation in virtual worlds. It is an ideal approach for modeling a behavioral animation and offers a universal approach to pass the necessary information from the environment to the virtual human in the problems of path searching, obstacle avoidance, and internal knowledge representation with learning and forgetting. In the next sections, we describe our approach for the three types of virtual sensors: vision, audition and haptic.

## **Virtual vision**

Although the use of vision to give behavior to synthetic actors seems similar to the use of vision for intelligent mobile robots (Horswill 1993, Tsuji and Li 1993), it is quite different. This is because the vision of the synthetic actor is itself a synthetic vision. Using a synthetic vision allow us to skip all the problems of pattern recognition and distance detection, problems which still are the most difficult parts in robotics vision. However some interesting work has been done in the topic of intelligent mobile robots, especially for action-perception coordination problems. For example, Crowley (1987), working with surveillance robots states that "most low level perception and navigation tasks are algorithmic in nature; at the highest levels, decisions regarding which actions to perform are based on knowledge relevant to each situation". This remark gives us the hypothesis on which our vision-based model of behavioral animation is built.

We first introduced (Renault et al. 1990) the concept of synthetic vision as a main information channel between the environment and the virtual actor. Reynolds (1993, 1994) more recently described an evolved, vision-based behavioral model of coordinated group motion, he also showed how obstacle avoidance behavior can emerge from evolution under selection pressure from an appropriate measure using a simple computational model of visual perception and locomotion. The Genetic Programming is used to model evolution. Tu and Terzopoulos (1994, 1994b) also use a kind of synthetic vision for their artificial fishes.

In (Renault et al. 1990), each pixel of the vision input has the semantic information giving the object projected on this pixel, and numerical information giving the distance to this object. So, it is easy to know, for example, that there is a table just in front at 3 meters. With this information, we can directly deal with the problematic question: "what do I do with such information in a navigation system?" The synthetic actor perceives his environment from a small window of typically 30x30 pixels in which the environment is rendered from his point of view. As he can access z buffer values of

the pixels, the color of the pixels and his own position he can locate visible objects in his 3D environment. This information is sufficient for some local navigation.

We can model a certain type of virtual world representation where the actor maintains a low level fast synthetic vision system but where he can access some important information directly from the environment without having to extract it from the vision image. In vision based grasping for example, an actor can recognize in the image the object to grasp. From the environment he can get the exact position, type and size of the object which allows him to walk to the correct position where he can start the grasping procedure of the object based on geometrical data of the object representation in the world. This mix of vision based recognition and world representation access will make him fast enough to react in real time. The role of synthetic vision can even be reduced to a visibility test and the semantic information recognition in the image can be done by simple color coding and non shading rendering techniques. Thus, position and semantic information of an object can be obtained directly from the environment world after being filtered.

### **Virtual audition**

In real life, the behavior of persons or animals is very often influenced by sounds. For this reason, we developed a framework for modeling a 3D acoustic environment with sound sources and microphones. Now, our virtual actors are able to hear (Noser and Thalmann 1995). Any sound source (synthetic or real) should be converted to the AIFF format and processed by the sound renderer. The sound renderer takes into account the real time constraints. So it is capable to render each time increment for each microphone in "real time" by taking into account the final propagation speed of sound and the moving sound sources and microphones. So, the Doppler effect, for example, is audible. In sound event generation, we integrated in our L-system-based software (Noser et al. 1992, Noser and Thalmann 1993) a peak detector of a force field which allows to detect collision between physical objects. These collisions can be coupled to sound emission events. For example, tennis playing with sound effects (ball-floor and ball-racket collision) has been realized. The acoustic environment is composed of sound sources and a propagation medium. The sound sources can produce sound events composed of a position in the world, a type of sound, and a start and an end time of the sound. The propagation medium corresponds to the sound event handler which controls the sound events and transmits the sounds to the ears of the actors and/or to a user and/or a soundtrack file. We suppose an infinite sound propagation speed of the sound without weakening of the signal. The sound sources are all omnidirectional, and the environment is non reverberant.

### **Virtual tactile**

One of our aims is to build a behavioral model based on tactile sensory input received at the level of skin from the environment. This sensory information can be used in tasks as touching objects, pressing buttons or kicking objects. For example at basic level, human should sense physical objects if any part of the body touches them and gather sensory information. This sensory information is made use of in such tasks as reaching out for an object, navigation etc. For example if a human is standing, the feet are in constant contact with the supporting floor. But during walking motion each foot alternately experiences the loss of this contact. Traditionally these motions are simulated using dynamic and kinematic constraints on human joints. But there are cases where information from external environment is needed. For example when a human descends a stair case, the motion should change from walk to descent based on achieving contact with the steps of the stairway. Thus the environment imposes constraints on the human locomotion. We propose to encapsulate these constraints using tactile sensors to guide the human figure in various complex situations other than the normal walking.

As already mentioned, simulating the haptic system corresponds roughly to a collision detection process. In order to simulate sensorial tactile events, a module has been designed to define a set solid objects and a set of sensor points attached to an actor. The sensor points can move around in space and collide with the above mentioned solid objects. Collisions with other objects out of this set are not detected. The only objective of collision detection is to inform the actor that there is a contact detected with an object and which object it is. Standard collision detection tests rely on bounding boxes or bounding spheres for efficient simulation of object interactions. But when highly irregular objects are present, such tests are bound to be ineffective. We need much 'tighter' bounding space than a box or a sphere could provide. We make use (Bandi and Thalmann 1995) of a variant of a digital line drawing technique called DDA (Fujimoto et al. 1986) to digitize the surface of such objects to get very tight fitting bounds that can be used in preliminary collision detection.

### **Perception-based actions**

Synthetic vision, audition and tactile allow the actor to perceive the environment. Based on this information, his behavioral mechanism will determine the actions he will perform. Actions may be at several degrees of complexity. An actor may simply evolve in his environment or he may interact with this environment or even communicate with other actors. We will emphasize three types of actions: navigation and locomotion, grasping and ball games.

Actions are performed using the common architecture for motion control developed in the European projects HUMANOID (Boulic et al. 1995) and HUMANOID-2. HUMANOID has led to the development of a complete system for animating virtual actors for film production. HUMANOID-2 currently extends the project for realtime applications and behavioral aspects as described in this paper. The heart of the HUMANOID software is the motion control part which includes 5 generators: keyframing, inverse kinematics, dynamics walking and grasping and high-level tools to combine and blend them. An interactive application TRACK (Boulic et al. 1994) has been also developed to create films and sequences to be played in realtime playback for multimedia applications.

### **Perception-based navigation and locomotion**

When the actor evolves in his environment, a simple walking model is not sufficient, the actor has to adapt his trajectory based on the variations of terrain by bypassing, jumping or climbing the obstacles he meets. The bypassing of obstacles consists in changing the direction and velocity of the walking of the actor. Jumping and climbing correspond to more complex motion. These actions should generate parameterized motion depending on the height and the length of the obstacle for a jump and the height and location of the feet for climbing the obstacle. These characteristics are determined by the actor from his perception.

The actor can be directed by giving his linear speed and his angular speed or by giving a position to reach. In the first case, the actor makes no perception (virtual vision). He just walks at the given linear speed and turns at the given angular speed. In the second case, the actor makes use of virtual vision enabling him to avoid obstacles. We developed a special automata for walking in complex environments with local vision based path optimization. So an actor continues walking even if he detects a future collision in front of him. By dynamically figuring out a new path during walking he can avoid the collision without halting. We also proposed a system for the automatic derivation of a human curved walking trajectory (Boulic et al. 1994b) from the analysis provided by its synthetic vision module. A general methodology associates the two low-level modules of vision and walking with a planning module which establishes the middle term path from the knowledge of the visualized environment. The planning is made under the constraint of minimizing the distance, the speed variation and the curvature cost. Moreover, the planning may trigger the alternate walking motion whenever the decreasing in curvature cost is higher than the associated increasing in speed variation cost due to the corresponding halt and restart. The Analysis of walking trajectories on a discrete environment with sparse foothold locations has been also completed (Boulic et al. 1993b) regarding the vision-based recognition of footholds, the local path planning, the next step selection and the curved body trajectory. The walking model used is based on biomechanical studies of specific motion pattern (Boulic et al. 1990).

### **Global and local navigation**

The task of a navigation system is to plan a path to a specified goal and to execute this plan, modifying it as necessary to avoid unexpected obstacles (Crowley 1987). This task can be decomposed into global navigation and local navigation. The global navigation uses a prelearned model of the domain which may be a somewhat simplified description of the synthetic world and might not reflect recent changes in the environment. This prelearned model, or map, is used to perform a path planning algorithm.

The local navigation algorithm uses the direct input information from the environment to reach goals and sub-goals given by the global navigation and to avoid unexpected obstacles. The local navigation algorithm has no model of the environment, and doesn't know the position of the actor in the world. To make a comparison with a human being, close your eyes, try to see the corridor near your room, and how to follow it. No problem, you were using your "visual memory," which corresponds to the global navigation in our system. Now stand up and go to the corridor near your room, then close your eyes and try to cross the corridor... There the problems begin (you know that there is a skateboard in front of your boss's door but...). This is an empirical demonstration of the functionalities of the local navigation as we define it in our system.

The global navigation needs a model of the environment to perform path-planning. This model is constructed with the information coming from the sensory system. Most navigation systems developed in robotics for intelligent mobile robots are based on the accumulation of accurate geometrical descriptions of the environment. Kuipers et al. (1988) give a nearly exhaustive list of such methods using quantitative world modeling. In robotics, due to low mechanical accuracy and sensory errors, these methods have failed in large scale area. We don't have this problem in Computer Graphics because we have access to the world coordinates of the actor, and because the synthetic vision or other simulations of perception systems are more accurate. We develop a 3D geometric model, based on grid, implemented as an octree. Elfes (1990) proposed a 2D geometric model based on grid but using a Bayesian probabilistic approach to filter non accurate information coming from various sensor positions. Roth-Tabak (1989) proposed a 3D geometric model based on a grid but for a static world.

In the last few years, research in robot navigation has tended towards a more qualitative approach to world modeling, first to overcome the fragility of purely metrical methods, but especially, because humans do not make spatial reasoning on a continuous map, but rather on a discrete map (Sowa 1964). Kuipers et al. (1988) present a topological model as the basic element of the cognitive map. This model consists of a set of nodes and arcs, where nodes represent distinctively recognizable places in the environment, and arcs represent travel edges connecting them. Travel edges corresponding to arcs are defined by local navigation strategies which describe how a robot can follow the link connecting two distinct places. These local navigation strategies correspond to the Displacement Local Automata (DLA) implemented in the local navigation part of our system. These DLAs work as a black box which has the knowledge to create goals and sub-goals in a specific local environment. They can be thought of as low-level navigation reflexes which use vision, reflexes which are automatically performed by the adults.

Noser et al. (1995) use an octree as the internal representation of the environment seen by an actor because it offers several interesting features. With an octree we can easily construct enclosing objects by choosing the maximum depth level of the subdivision of space. Detailed objects like flowers and trees do not need to be represented in complete detail in the problem of path searching. It is sufficient to represent them by some enclosing cubes corresponding to the occupied voxels of the octree. The octree adapts itself to the complexity of the 3D environment, as it is a dynamic data structure making a recursive subdivision of space. Intersection tests are easy. To decide whether a voxel is occupied or not, we only have to go to the maximum depth (5-10) of the octree by some elementary addressing operations. The examination of the neighborhood of a voxel is immediate, too. Another interesting property of the octree is the fact that it represents a graph of a 3D environment. We may consider, for example, all the empty voxels as nodes of a graph, where the neighbors are connected by edges. We can apply all the algorithms of graph theory directly on the octree and it is not necessary to change the representation. Perhaps the most interesting property of the octree is the simple and fast transition from the 2D image to the 3D representation. All we have to do is take each pixel with its depth information (given by the z-buffer value) and calculate its 3D position in the octree space. Then, we insert it in the octree with a maximum recursion depth level. The corresponding voxel will be marked as occupied with possible additional information depending on the current application. The octree has to represent the visual memory of an actor in a 3D environment with static and dynamic objects. Objects in this environment can grow, shrink, move or disappear. In a static environment (growing objects are still allowed) an *insert* operation for the octree is sufficient to get an approximate representation of the world. If there are moving or disappearing objects like cars, other actors, or opening and closing doors, we also need a *delete* operation for the octree. The *insert* operation is simple enough. The *delete* operation however, is more complicated.

To illustrate the capabilities of the synthetic vision system, we have developed several examples. Figure 1 contains 5 different windows. The "Comment window" shows a title of the picture (yellow), the actual state of the actor (yellow) and the labels (red) of the "vision" and "projected memory" window. The vision window (50 x 50) pixels shows the scene from the point of view of the actor. From this window the actor takes all his information of his environment. In the "projected memory" window we can see the points of the actor's octree memory (occupied voxels) from his point of view. These points are used to compare the real scene with the visual memory and to decide whether an object (voxel) has to be removed from memory or not. The "Real scene" window shows the whole scene with the actor in it from any arbitrary position. The last window "Octree memory" visualizes the actual state of the actor's visual octree memory. Each occupied voxel of the memory octree is rendered by a corresponding yellow cube. The blue voxels (cubes) represent the path octree, which corresponds to the path the actor has found and will use. In moving mode, the first and the last voxel of the path octree represent the start and end points of the path. In explore mode, one end of the blue path corresponds to the start point and the other end to the actual position of the actor.

In our example, an actor was placed in the interior of a maze with an impasse, a circuit and some animated flowers. The actor's first goal was a point outside the maze. After some time the actor succeeded in finding his goal. When he had completely memorized the impasse and the circuit, he avoided them. After reaching his first goal, he had a nearly complete visual octree representation of his environment and he could find without problem his initial position by a simple reasoning process. In this example he was never obliged to use the "explore" mode. The heuristic guarantees that we can always find a path, if one exists. In the maze there was a wall which disappeared after some time. When the actor passed there a second time, he deleted it from his memory, too.

## Local Navigation System

The local navigation system is an extension of the approach described by Renault et al. [13] and can be decomposed into three modules. The *vision* module, conceptually the perception system, draws a perspective view of the world in the vision window, constructs the vision array and can perform some low level operation on the vision array. The *controller* module, corresponding to the decision system, contains the main loop for the navigation system, and decides on the creation of the goals and administers the DLAs. The *performer*, corresponding to the task execution system, contains all the DLAs.

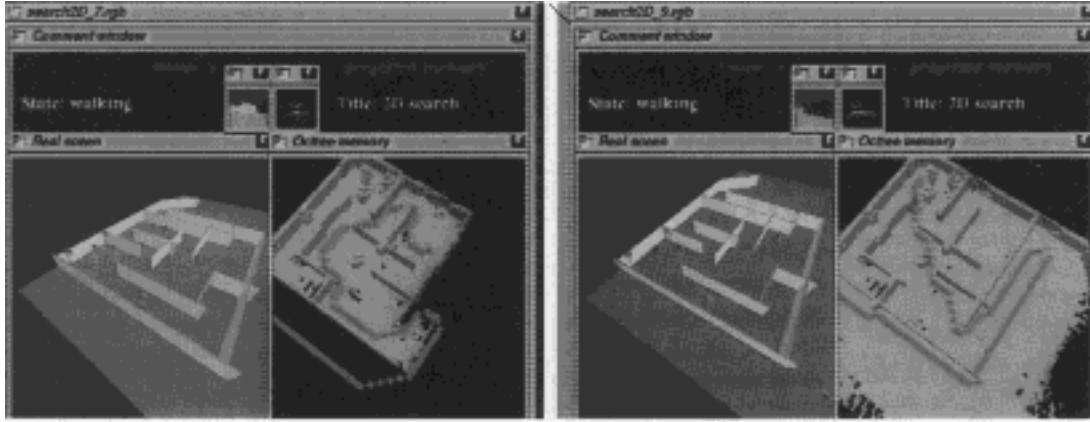


Figure 1. 2D search for an exit of a maze

- **The Vision module**

We use the hardware facilities of the Silicon Graphics workstation to create the synthetic vision, more precisely we use the flat shading and z-buffer drawing capabilities of the graphic engine. The vision module has a modified version of the drawing routine traveling the world; instead of giving the real color of the object to the graphic engine, this routine gives a code, call the *vision\_id*, which is unique for each object and actor in the world. This code allows the image recognition and interpretation. Once the drawing is done, the window buffer is copied into a 2D array. This array contains the *vision\_id* and the z-buffer depth for each pixel. This array is referred as the *view*.

- **The Controller module**

In local navigation there are two goals. These two goals are geometrical goals, and are defined in the local 2D coordinate system of the actor. The actor itself is the center of this coordinate system, one axis is defined by the direction "in front", the other axis is defined by the "side" direction. The global goal, or final goal, is the goal the actor must reach. The local goal, or temporary goal, is the goal the actor creates to avoid the obstacles encountered in the path towards the global goal. These goals are created by the Displacement Local Automata (DLA), or given by the animator or by the global navigation system. The main task of the controller is to create these goals created and to make the actor reach them. Goal creation and actor displacement are performed by the DLAs. The controller selects the appropriate DLA either by knowing some internal set-up of the actor, or by visual by analyzing the environment. For example, if the actor has a guide, the controller will choose the DLA *follow\_the\_guide*. Otherwise, from a 360 look-around, the controller will determine the visible objects and then determine the DLA corresponding to these objects. No real interpretation of the topology of the environment (as in Kuipers et al. 1989) has yet been implemented.

The actor has an internal clock administrated by the controller. This clock is used by the controller to refresh the global and local goal at regular intervals. The interval is given by the *attention\_rate*, a variable set-up for each actor that can be changed by the user or by the controller. This variable is an essential parameter of the system: with a too high attention rate the actor spends most of his time analyzing the environment and real-time motion is impossible; with a too low attention rate, the actor starts to act blindly, going through objects. A compromise must be found between these two extremes.

- **The Performer module**

This module contains the DLAs. There are three families of DLAs: the DLAs creating the global goal (*follow\_the\_corridor*, *follow\_the\_wall*, *follow\_the\_visual\_guide*), the DLAs creating the local goal (*avoid\_obstacle*, *closest\_to\_goal*), and the DLAs effectively moving the actor (*go\_to\_global\_goal*). The DLAs creating goals only use the vision as input. All these DLAs have access to a library of routines performing high level operations on the vision. "High level" means that these routines can change the orientation of the vision, i.e. to see if a particular object is visible from the current position of the actor.

For obstacle avoidance, as there is no internal representation of the world, the strategy is the classic one: try to go straight to the global goal and if there is an obstacle in the direction, move around until there is a possible straight path to the global goal. So, the DLA *avoid\_obstacle* only treats one obstacle at a time. Only the obstacles situated inside a delimited perimeter are taken into account. The size of this perimeter is set by a variable called *attention\_dist*, a duration (in seconds) which, multiplied by the speed (in m/s), gives the desired distance. A detailed algorithm of the use of vision to find avoidance goal is described by Renault et al.[12]

## 4.2 The complete system

The three basic modules of the local navigation are integrated in an interactive program allowing a user to perform simulation with digital actors in real time. This complete system has two purposes. The first is to test the development made on the basic modules vision, controller and performer. The user is just a viewer of the scene, only able to manipulate some variables like the speed of the actor or its attention rate. The second purpose of this system is to test the use of this vision-based navigation system for animation. In this case the user should have some possibilities to direct the actor at a high level. The idea is to give the animator a way to easily create the rough trajectory of the actor. This rough trajectory can correspond with direction orders such as "coming from the front door, going to the window and then leaving by the back door." We have implemented "guides" the animator can move while recording the motion. These guides are goals for the actor which try to reach them, avoiding obstacles. The animator can choose between a visual guide, a dummy object that the actor follows using his vision, or a "smell guide", a guide that can be hidden by the decor but whose location is known by the actor. The animator records the trajectory of the guide using the SpaceBall to control the movements, he then plays the guide and lets the actor follow it. At this time the animator has two possibilities to change the final animation. The first one is to keep the same guide trajectory and change parameters as attention\_rate, and the second one is to re-record the guide trajectory using the result from the previous try.

### Perception-based grasping

With the advents of virtual actors in computer animation, research in human grasping has become a key issue in this field. Magnenat Thalmann et al. (1988) proposed a semi-automatic way of grasping for a virtual actor interacting with the environment. A knowledge-based approach is suitable for simulating human grasping, and an expert system can be used for this purpose (Rijpkema and Girard 1991). Kunii et al. (1993) have also presented a model of hands and arms based on manifold. Our approach is based on three steps:

#### Inverse kinematics to find the final arm posture

Inverse kinematics (Philips et al. 1990) is a widely used technique in Robotics. This technique is suited to control or constraint strategic parts of the articulated figure which location depends on a set of parameters. The control scheme is based on a linearization of the location problem at the current state of the system.

Before using inverse kinematics, the first problem is to select an end effector on the open chain of the articulated figure. There are many choices for grasping that result in different treatments. We choose the hand center as the end effector for convenience. The hand center can be on palm surface or above it. It depends whether the final frames are inside or not. The Z axis is perpendicular to the palm of the hand, X is parallel to the direction of straight fingers, and Y is the cross product of X and Z. The frame should be aligned to the final object frame for the final posture. This final object frame depends on the geometry of the object, the hand size, and avoidance of collision if more than one hand are grasping one object.

#### Heuristic grasping decision.

Based on a grasp taxonomy, Mas and Thalmann (1994) proposed a completely automatic grasping system for synthetic actors. In particular, the system can decide to use a pinch when the object is too small to be grasped by more than two fingers or to use a two-handed grasp when the object is too large. Fig. 2 shows examples produced by the system.

From observation on real grasping and methods in robotics, there are many ways of grasping with one or two hands, with two to five fingers of each hand according to the type, geometry and weight of the object to be grasped. We summarize it in Table 1 only for solid primitives. For a more general object, the decision should be made according to its multiple bounding volumes. The bounding volumes can be a sphere, a cube, a cylinder etc. For example, for the surface model of a human body, the head will be bounded by a sphere, and the neck by a cylinder.

In all cases, pinch is applied if the primitive is too small to be grasped by more than two fingers. Two hands are applied if it is too large to be taken by one hand. Small and large are compared with hand size.

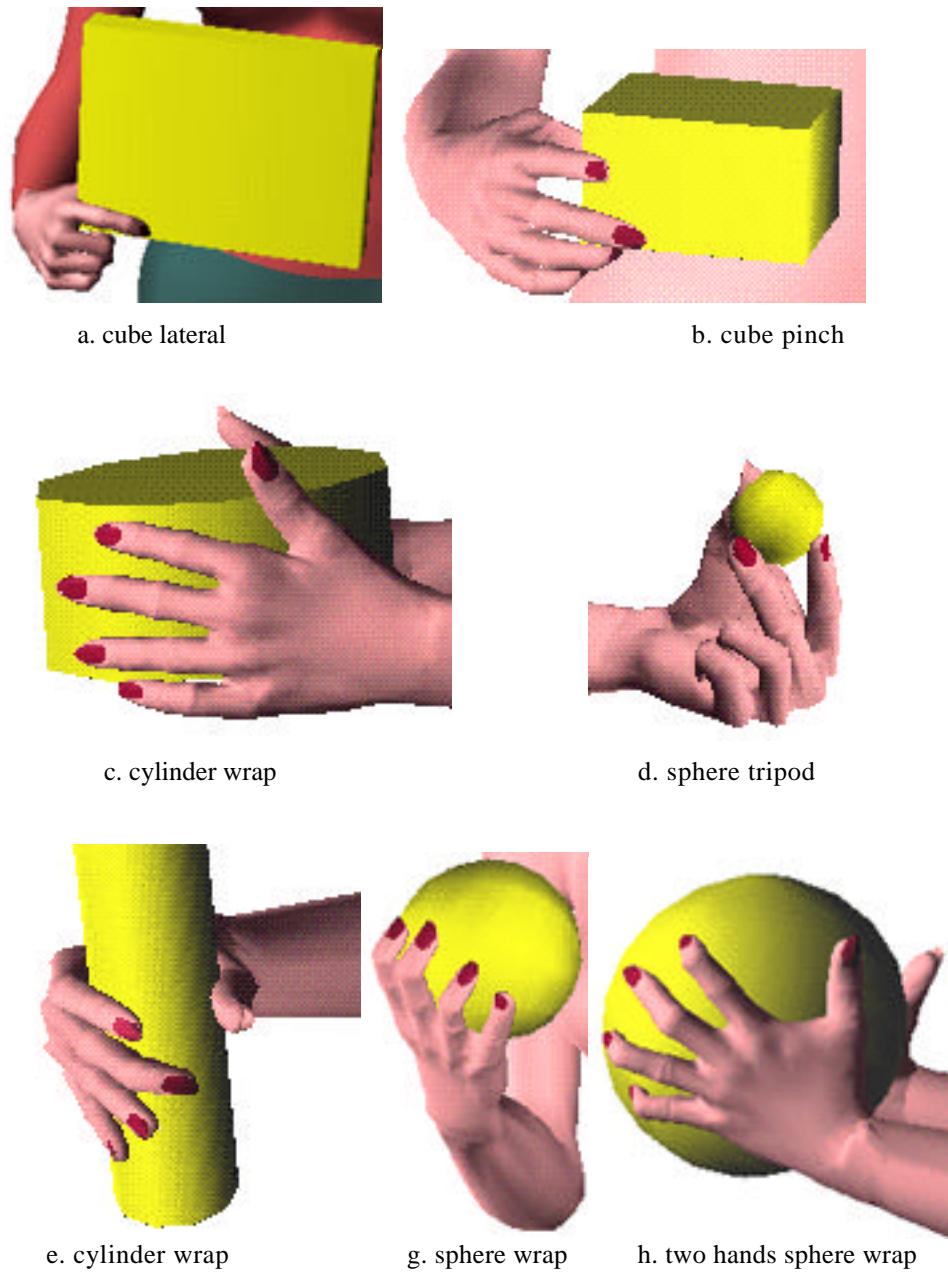


Figure 2. The different grasping ways for different objects

### Multi-sensor hand.

Our approach (Huang et al. 1995) is adapted from the use of proximity sensors in Robotics (Espiau and Boulic 1985), the sensor-actuator networks (van de Panne and Fiume 1993) and recent work on human grasping (Mas and Thalmann 1994). In our work, the sphere multi-sensors have both tactile and length sensor properties, and have been found very efficient for synthetic actor grasping problem. Multi-sensors are considered as a group of objects attached to the articulated figure. A sensor is activated for any collision with other objects or sensors. Here we select sphere sensors for their efficiency in collision detection (Figure 3). Each sphere sensor is fitted to its associated joint shape with different radii. This configuration is important in our method because when a sensor is activated in a finger, only the articulations above it stop moving, while others can still move. By doing this way, all the fingers are finally positioned naturally around the object, as shown in Fig. 3.

Type	Size	Grasping way
Cube	thin	lateral
	thick	pinch
Sphere	small diameter	tripod
	large diameter	wrap
Cylinder and Frustum	small diameter	pinch
	large diameter	wrap
	small height	disk

Table 1. The grasping decision

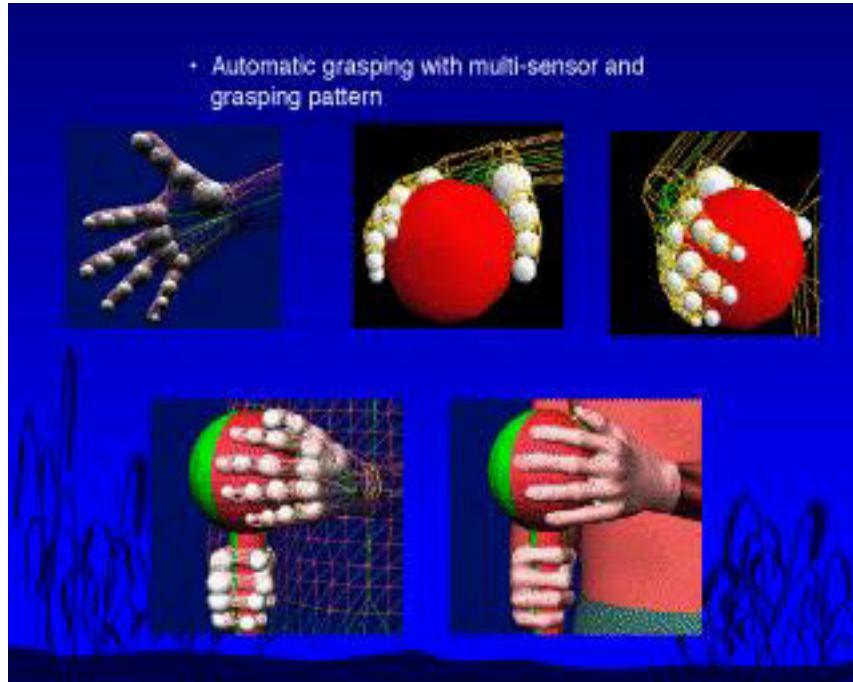


Figure 3. The hand with sphere sensors at each joint

All the grasping functions mentioned above have been implemented and integrated into the TRACK system (Boulic et al. 1994). The first example shows actors grasping different objects (Fig.4). In Figure 5, we extend the grasping method to interactions between two actors.

### Vision-based Tennis Playing

Tennis playing is a human activity which is severely based on the vision of the players. In our model, we use the vision system to recognize the flying ball, to estimate its trajectory and to localize the partner for game strategy planning. The geometric characteristics of the tennis court however, make part of the players knowledge. For the dynamics simulation of the ball, gravity, net, ground and the racquet we use the force field approach developed for the L-system animation system. The tracking of the ball by the vision system is controlled by a special automata.

The two actors play in a physically modeled environment given by differential equations and supported by the L-system. In this "force field animation system" the 3D world is modeled by force fields. Some objects have to carry repulsion forces, if there should not be any collision with them. Other objects can be attractive to others. Many objects are both attractive at long distances and repulsive at short distances. The shapes and sizes of these objects can vary, too. Space fields like gravity or wind force fields can greatly influence animation sequences or shapes of trees.

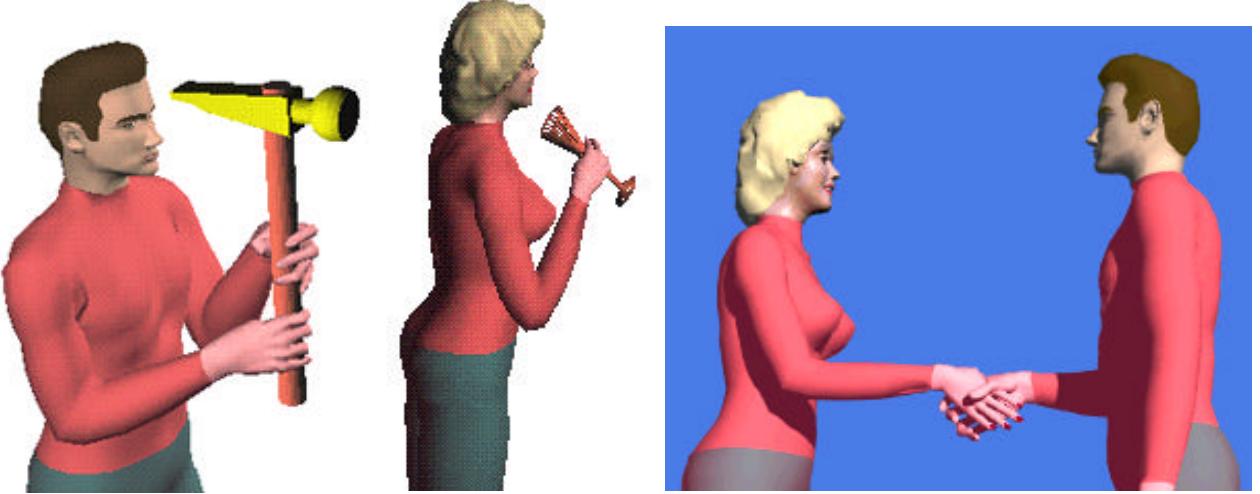


Figure 4. Examples of grasping different objects

Figure 5. Interaction between two actors

In the navigation problem each colored pixel is interpreted as an obstacle. No semantic information is necessary. In tennis playing however, the actor has to distinguish between the partner, the ball and the rest of the environment. The ball has to be recognized, its trajectory has to be estimated and it has to be followed by the vision system. At the beginning of a ball exchange, the actor has to verify that its partner is ready. During the game the actor needs also his partner's position for his play strategy. To recognize objects in the image we use color coding. The actor knows that a certain object is made of a specific material. When it scans the image it looks for the corresponding pixels and calculates its average position and its approximate size. Thus each actor can extract some limited semantic information from the image. Once the actor has recognized the ball, it follows it with his vision system and adjusts at each frame his field of view. To play tennis each partner has to estimate the future racket-ball collision position and time and to move as fast as possible to this point. At each frame (1/25 sec) the actor memorizes the ball position. So, every n-th frame the actor can derive the current velocity of the ball. From this current velocity and the current position of the ball it can calculate the future impact point and impact time.

All the above features are coordinated by a specialized "tennis play" automata. First an actor goes to his start position. There he waits until his partner is ready. Then he looks for the ball, which is thrown into the game. Once the vision system has found the ball, it always follows it by adjusting the field of view angle. If the ball is flying towards the actor, it starts estimating the impact point. Once the ball has passed the net, the actor localizes his partner with his vision system during one frame. This information is used for the game strategy. After playing the ball, the actor goes back to his start point and waits until the ball comes back to play it again. This simplified automata does not treat playing errors. In our current version the actor is a with synthetic vision equipped racket, moving along a spline with no speed limitations. The aim of the project was to show the feasibility of a vision based tennis playing. Error free simulations of tennis playing during several minutes have been performed. It is planned in a future extension to integrate synthetic actors with walk and hit motor and to add some error treatment.

A prototype of this automata is already able to track the ball, to estimate the collision time and collision point of ball and racquet and to perform successfully a hit with given force and a given resulting ball direction. In a first step, we have a prototype where only two racquets with synthetic vision can play against each other (Fig.6), in order to develop, test and improve game strategy and the physical modeling. The integration of the corresponding locomotor system of a sophisticated actor ha been developed recently as seen in Fig.7.

## Conclusion

In this paper, we have presented a new approach to implement autonomous virtual actors in virtual worlds based on perception and virtual sensors. We believe this is an ideal approach for modeling a behavioral animation and offers a universal approach to pass the necessary information from the environment to an actor in the problems of path searching, obstacle avoidance, game playing, and internal knowledge representation with learning and forgetting characteristics. We also think that this new way of defining animation is a convenient and universal high level approach to simulate the behavior of intelligent human actors in dynamics and complex environments including virtual environments.

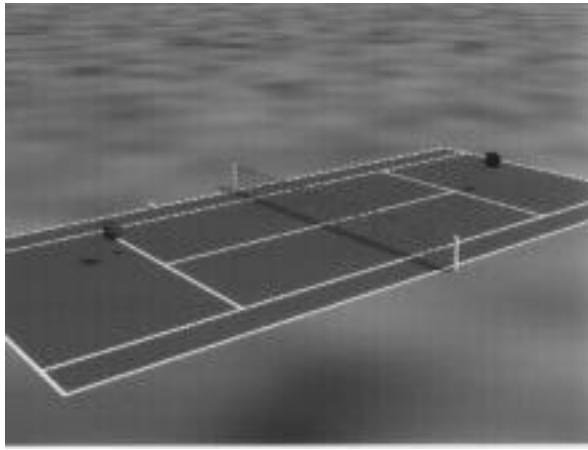


Figure 6. Vision-based tennis playing

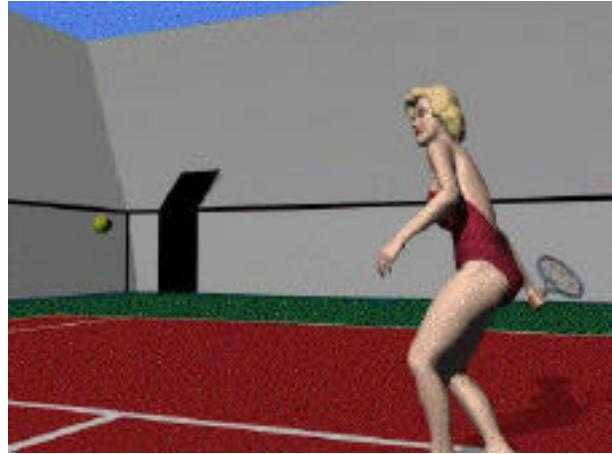


Figure 7. Marilyn playing tennis

### Acknowledgments

The authors are grateful to the people who contributed to this work, in particular Srikan Bandi, Pascal Bécheiraz, Ronan Boulic, and Serge Rezzonico. The research was supported by the Swiss National Science Research Foundation, the Federal Office for Education and Science, and is part of the Esprit Project HUMANOID and HUMANOID-2.

### References

- Astheimer P., Dai Gobel M. Kruse R., Müller S., Zachmann G. (1994) "Realism in Virtual Reality", in: (Magnenat Thalmann and Thalmann, eds) *Artificial Life and Virtual Reality*, John Wiley, Chichester, pp.189-208.
- Bandi S., Thalmann D. (1995) "An Adaptive Spatial Subdivision of the Object Space for Fast Collision Detection of Animated Rigid Bodies", Proc. Eurographics '95
- Bates J, Loyall AB, Reilly WS (1992) "An architecture for Action, Emotion, and Social Behavior", Proc. Fourth Europeans Workshop on Modeling Autonomous Agents in a multi Agents World, S. Martino al Cimino, Italy.
- Boulic R., Capin T., Kalra P., Lintermann B., Moccozet L., Molet T., Huang Z., Magnenat-Thalmann N., Saar K., Schmitt A., Shen J. and Thalmann D. (1995) "A system for the Parallel Integrated Motion of Multiple Deformable Human Characters with Collision Detection", *EUROGRAPHICS' 95*, Maastricht.
- Boulic R., Huang Z., Magnenat-Thalmann N., Thalmann D. (1994) "Goal-Oriented Design and Correction of Articulated Figure Motion with the TRACK System", *Comput. & Graphics*, Vol. 18, No. 4, pp. 443-452.
- Boulic R., Noser H., Thalmann D. (1993) "Vision-Based Human Free-Walking on Sparse Foothold Locations", Fourth Eurographics Workshop on Animation and Simulation, Barcelona Spain, Eurographics, pp.173-191
- Boulic R., Noser H., Thalmann D. (1994b) "Automatic Derivation of Curved Human Walking Trajectories from Synthetic Vision", *Computer Animation '94*, Geneva, IEEE Computer Society Press, pp.93-103.
- Boulic R., Thalmann D, Magnenat-Thalmann N. (1990) "A global human walking model with real time kinematic personification" *The Visual Computer*, 6(6).
- Braitenberg V. (1984) "Vehicles, Experiments in Synthetic Psychology". The MIT Press.
- Clark J.H. (1982) "The Geometric Engine: A VLSI Geometry System for Graphics", Proc. SIGGRAPH '82, Computer Graphics, Vol. 10, No3, pp.127-133.
- Crowley J.L. (1987) "Navigation for an Intelligent Mobile Robot", *IEEE journal of Robotics and Automation*, Vol. RA-1, No. 1, pp 31-41.
- Elfes A. (1990) "Occupancy Grid: A Stochastic Spatial Representation for Active Robot Perception", Proc. Sixth Conference on Uncertainty in AI.
- Espiau B., Boulic R. (1985) "Collision avoidance for redundant robots with proximity sensors", Proc. of Third International Symposium of Robotics Research, Gouvieux, October.
- Fujimoto A., Tanaka T., Iwata K.(1986) "ARTS:Accelerated Ray-Tracing System", *IEEE CG&A*, vol.6, No4, pp.16-26.
- Haumann D.R., Parent R.E. (1988) "The Behavioral Test-bed: Obtaining Complex Behavior from Simple Rules", *The Visual Computer*, Vol.4, No 6, pp.332-347.
- Horswill I. (1993) "A Simple, Cheap, and Robust Visual Navigation System", in: *From Animals to Animats 2*, Proc. 2nd Intern. Conf. on Simulation of Adaptive Behavior, MIT Press, pp.129-136.
- Huang Z., Boulic R., Magnenat Thalmann N., Thalmann D. (1995) "A Multi-sensor Approach for Grasping and 3D Interaction", Proc. CGI '95

- Hügli H., Facchinetti C., Tièche F., Müller J.P., Rodriguez M., Gat Y. (1994) "Architecture Of An Autonomous System: Application to Mobile Robot Navigation". In Proceedings of Symposium on Artificial Intelligence and Robotics, pp. 97-110.
- Kuipers B., Byun Y.T. (1988) "A Robust Qualitative Approach to a Spatial Learning Mobile Robot", SPIE Sensor Fusion: Spatial Reasoning and Scene Interpretation, Vol. 1003.
- Kunii T.L., Tsuchida Y., Matsuda H., Shirahama M., Miura S. (1993) "A model of hands and arms based on manifold mappings", Proceedings of CGI'93, pp.381-398.
- Lethbridge T.C. and Ware C. (1989) "A Simple Heuristically-based Method for Expressive Stimulus-response Animation", Computers and Graphics, Vol.13, No3, pp.297-303
- Maes P. (ed.) (1991) "Designing Autonomous Agents", Bradford MIT Press.
- Maes P. (1991b) "Bottom-Up Mechanism for Behavior Selection in an Artificial Creature", Proc. First International Conference on Simulation of Adaptive Behavior, 1991.
- Magnenat-Thalmann N., Laperrière R., Thalmann D. (1988) "Joint-dependent local deformations for hand animation and object grasping", Proceedings of Graphics Interface '88, pp.26-33.
- Magnenat Thalmann N., Thalmann D. (1991) "Still Walking", video, 1 min.
- Magnenat Thalmann N., Thalmann D. (1994) "Creating Artificial Life in Virtual Reality" in: (Magnenat Thalmann and Thalmann, eds) Artificial Life and Virtual Reality, John Wiley, Chichester, 1994, pp.1-10
- Magnenat Thalmann N., Thalmann D. (1995) "Digital Actors for Interactive Television", Proc. IEEE, July.
- Mas S.R., Thalmann D. (1994) "A Hand Control and Automatic Grasping System for Synthetic Actors", Proc. Eurographic'94, pp.167-178.
- Noser H., Thalmann D. (1993) "L-System-Based Behavioral Animation", Proc. Pacific Graphics '93, pp.133-146.
- Noser H., Thalmann D., Turner R. (1992) "Animation based on the Interaction of L-systems with Vector Force Fields", Proc. Computer Graphics International, in: Kunii TL (ed): Visual Computing, Springer, Tokyo, pp.747-761.
- Noser H., Renault O., Thalmann D., Magnenat Thalmann N. (1995) "Navigation for Digital Actors based on Synthetic Vision, Memory and Learning", Computers and Graphics, Pergamon Press, Vol.19, No1, pp.7-19.
- Noser H., Thalmann D. (1995) "Synthetic Vision and Audition for Digital Actors", Proc. Eurographics '95.
- Renault O., Magnenat Thalmann N., Thalmann D. (1990) "A Vision-based Approach to Behavioural Animation", The Journal of Visualization and Computer Animation, Vol 1, No 1, pp 18-21.
- Reynolds C. (1987) "Flocks, Herds, and Schools: A Distributed Behavioral Model", Proc. SIGGRAPH '87, Computer Graphics, Vol.21, No4, pp.25-34
- Reynolds C.W. (1993) "An Evolved, Vision-Based Behavioral Model of Coordinated Group Motion", in: Meyer J.A. et al. (eds) From Animals to Animats, Proc. 2nd International Conf. on Simulation of Adaptive Behavior, MIT Press, pp.384-392.
- Reynolds C.W. (1994) "An Evolved, Vision-Based Model of Obstacle Avoidance Behavior", in: C.G. Langton (ed.), Artificial Life III, SFI Studies in the Sciences of Complexity, Proc. Vol. XVII, Addison-Wesley.
- Ridsdale G. (1990) "Connectionist Modelling of Skill Dynamics", Journal of Visualization and Computer Animation, Vol.1, No2, 1990, pp.66-72.
- Rijpkema H., Girard M. (1991) "Computer animation of knowledge-based human grasping", Proceedings of Siggraph'91, pp.339-348.
- Roth-Tabak Y. (1989) "Building an Environment Model Using Depth Information", Computer, pp 85-90.
- Sims K. (1994) "Evolving Virtual Creatures", Proc. SIGGRAPH '94, pp. 15-22.
- Slater M., Usoh (1994) "Body centred Interaction in Immersive Virtual environments", in: (Magnenat Thalmann and Thalmann, eds) Artificial Life and Virtual Reality, John Wiley, Chichester, 1994, pp.1-10
- Sowa JF (1964) Conceptual Structures, Addison-Wesley.
- Tsuji S., Li S. (1993) "Memorizing and Representing Route Scenes", in: Meyer J.A. et al. (eds) From Animals to Animats, Proc. 2nd International Conf. on Simulation of Adaptive Behavior, MIT Press, pp.225-232.
- Tu X., Terzopoulos D. (1994) "Artificial Fishes: Physics, Locomotion, Perception, Behavior", Proc. SIGGRAPH '94, pp.42-48.
- Tu X., Terzopoulos D. (1994b) "Perceptual Modeling for the Behavioral Animation of Fishes", Proc. Pacific Graphics '94, World Scientific Publishers, Singapore, pp.165-178
- Tyrell T. (1993) "The Use of Hierarchies for Action Selection", in: From Animals to Animats 2, Proceedings of the Second International Conference on Simulation of Adaptive Behavior, pp. 138-147.
- van de Panne M., and Fiume E. (1993) Sensor-Actuator Network, Proc. SIGGRAPH'93, pp.335-342.
- Wilhelms J. (1990) "A "Notion" for Interactive Behavioral Animation Control", IEEE Computer Graphics and Applications , Vol. 10, No 3 , pp.14-22A

# Playing Games through the Virtual Life Network<sup>‡</sup>

**Hansrudi Noser<sup>1</sup>, Igor Sunday Pandzic<sup>2</sup>, Tolga K. Capin<sup>1</sup>,  
Nadia Magnenat Thalmann<sup>2</sup>, Daniel Thalmann<sup>1</sup>**

<sup>1</sup> Computer Graphics Laboratory  
Swiss Federal Institute of Technology (EPFL)  
CH 1015 Lausanne, Switzerland

<sup>2</sup> MIRALAB - CUI  
University of Geneva  
24 rue du Général-Dufour  
CH 1211 Geneva 4, Switzerland

Simulating autonomous virtual actors living in virtual worlds with human-virtual interaction and immersion is a new challenge. The sense of "presence" in the virtual environment is an important requirement for collaborative activities involving multiple remote users working with social interactions. Using autonomous virtual actors within the shared environment is a supporting tool for presence. This combination of Artificial Life with Virtual Reality cannot exist without the growing development of Computer Animation techniques and corresponds to its most advanced concepts and techniques. In this paper, we present a shared virtual life network with autonomous virtual humans that provides a natural interface for collaborative working and games. We explain the concept of virtual sensors for virtual humans and show an application in the area of tennis playing.

**Keywords:** Artificial Life, Virtual Reality, Telecooperative Work, Computer Animation, Networked Multimedia, Virtual Actors.

## 1. Introduction

One of the most challenging Virtual Reality applications are real-time simulations and interactions with autonomous actors especially in the area of games and cooperative work. In this paper, we present VLNET which is a shared virtual life network with virtual humans that provides a natural interface for collaborative working and games. Virtual actors play a key role in VLNET. Three kinds of virtual actors may coexist in a VLNET scene: participant, guided and autonomous actors. A participant actor is a virtual copy of the real user or participant; his movement is exactly the same as the real user. A guided actor is an actor completely controlled in real-time by the user. An autonomous actor is an actor who may act without intervention of the user. As a typical example, we will consider tennis playing as shown in Figure 1.



Figure 1. Tennis

---

<sup>‡</sup> An expanded version of this text has been published in Proc. Alife '96, Nara, Japan, 1996.

Using VLNET, we will show how it is possible to have players who may be participants, guided or autonomous. For example, a participant could play tennis or chess against another participant or an autonomous actor. A 3D puzzle may be solved by two autonomous actors or a participant with the help of a guided actor. VLNET brings together four essential technologies: networked Computer Supported Cooperative Work (CSCW), Virtual Reality, Artificial Life, and Computer Animation.

In the next Section, we will present an overview of the system, then we will discuss the three types of actors, emphasizing the autonomous actors. Section 5 explains the simulation of virtual sensors for these autonomous actors. Section 6 presents tennis playing and navigation based on these sensors. Section 7 is dedicated to the problem of making autonomous virtual actors aware of participants. Finally, some aspects of implementation are described.

## 2. The System Architecture

A typical environment for game playing through VLNET is shown in Figure 2. For the real time tennis game simulation with synthetic actors we use a behavioral L-system interpreter. This process shares with the participant client through VLNET clients and the VLNET server the environment elements important for the simulation, as the tennis court, the tennis ball, an autonomous referee, the autonomous player and the participant. The representation of the participant in the L-system interpreter is reduced to a simple racket whose position is communicated to the other clients through the network at each frame. The racket position of the autonomous actor is communicated at each frame to the user client where it is mapped to an articulated, guided actor. This guided actor is animated through inverse kinematics according to the racket position. The referee is also represented by a guided articulated actor in the user client getting its position at each frame from the L-system animation process. The ball movement is modeled according to physical laws in the animation system and communicated to all clients through the network.

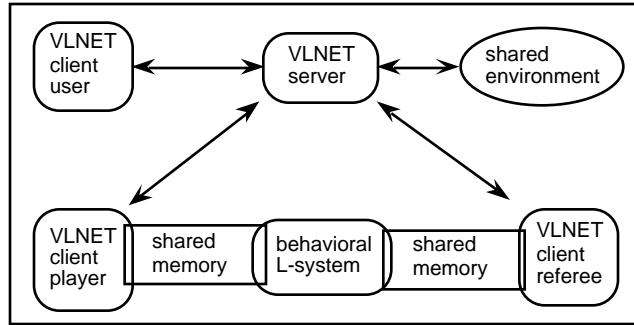


Figure 2. The system architecture

### 2.1 The VLNET System

Providing a behavioral realism is a significant requirement for systems that are based on human collaboration, such as Computer Supported Cooperative Work (CSCW) systems. Networked CSCW systems [1, 2, 3, 4] also require that the shared environment should: provide a comfortable interface for gestural communication, support awareness of other users in the environment, provide mechanisms for different modes of interaction (synchronous vs. asynchronous, allowing to work in different times in the same environment), supply mechanisms for customized tools for data visualization, protection and sharing.

The VLNET [5, 6] (Virtual Life NETwork) system supports a networked shared virtual environment that allows multiple users to interact with each other and their surrounding in real time. The users are represented by 3D virtual human actors, which serve as agents to interact with the environment and other agents. The agents have similar appearance and behaviors with the real humans, to support the sense of presence of the users in the environment. In addition to user-guided agents, the environment can also be extended to include fully autonomous human agents used as a friendly user interface to

different services such as navigation. Virtual humans can also be used in order to represent the currently unavailable partners, allowing asynchronous cooperation between distant partners.

The environment incorporates different media; namely sound, 3D models, facial interaction among the users, images represented by textures mapped on 3D objects, and real-time movies. Instead of having different windows or applications for each medium, the environment integrates all tasks in a single 3D surrounding, therefore it provides a natural interface similar to the actual world. The objects in the environment are classified into two groups: fixed (e.g. walls) or free (e.g. a chair). Only the free objects can be picked, moved and edited. This allows faster computations in database traversal for picking. In addition to the virtual actors representing users, the types of objects can be: simple polygonal objects, image texture-mapped polygons, etc. Once a user picks an object, he or she can edit the object. Each type of object has a user-customized program corresponding to the type of object, and this program is spawned if the user picks and requests to edit the object.

## 2.2 The Behavioral L-system

The environment of the autonomous actors is modeled and animated with L-systems which are timed production systems designed to model the development and behavior of static objects, plant like objects and autonomous creatures. They are based on timed, parameterized, stochastic, conditional environmentally sensitive and context dependent production systems, force fields, synthetic vision and audition. More details about our L-system-based animation may be found in [7, 8, 9, 10]. Original L-systems [11] were created as a mathematical theory of plant development with a geometrical interpretation based on turtle geometry. Our behavioral L-system [12] is based on the general theory about L-Grammars described in this work.

An L-system is given by an axiom being a string of parametric and timed symbols, and production rules specifying how to replace corresponding symbols in the axiom during the time evolution. The L-system interpreter associates to its symbols basic geometric primitives, turtle control symbols or special control operations necessary for an animation. We define the non generic environment as a tennis court directly in the axiom of the production system. The generic parts as growing plants are defined by production rules having only their germ in the axiom. Actors are also represented by a special symbol.

## 3. Participant, user-guided and autonomous real-time creatures

As virtual actors play a key role in VLNET, we will first try to clarify the concept of virtual actor. We define a virtual (or synthetic) actor as a human-like entity with an abstract representation in the computer. A real-time virtual actor is a virtual actor able to act at the same speed as a real person. Virtual Reality, Interactive Television, and Games require real-time virtual actors. In VLNET, three types of real-time virtual actors may coexist in the same shared environment.

- **Participant actors**

A participant actor is a virtual copy of the real user or participant. His movement is exactly the same as the real user. This can be best achieved by using a large number of sensors to track every degree of freedom in the real body, however this is generally not possible due to limitations in number and technology of the sensing devices. Therefore, the tracked information is connected with behavioral human animation knowledge and different motion generators in order to "interpolate" the joints of the body which are not tracked.

- **Guided actors**

A guided actor is an actor completely controlled in real-time by the user. In VLNET, the best example of actor guidance is guided navigation. The participant uses the input devices to update the transformation of the eye position of the virtual actor. This local control is used by computing the incremental change in the eye position, and estimating the rotation and velocity of the center of body. The walking motor uses the instantaneous velocity of motion, to compute the walking cycle length and time, by which it computes the necessary joint angles. The arm motion for picking an object as a chess pawn (see Figure 3) is a similar problem to walking: given 6 degrees of freedom

(position and orientation) of the sensed right hand with respect to body coordinate system, the arm motor should compute the joint angles within the right arm.

- **Autonomous actors**

An autonomous actor is an actor who may act without intervention of the user. In the next sections, the role of the autonomous real-time virtual actor is explained in more details.



Figure 3. Chess

#### 4. Autonomous Actors

An autonomous system is a system that is able to give to itself its proper laws, its conduct, opposite to heteronomous systems which are driven by the outside. Guided actors are typically driven by the outside. Including autonomous actors that interact with participants increases the real-time interaction with the environment, therefore we believe that it contributes to the sense of presence in the environment. The autonomous actors are connected to the VLNET [13] system in the same way as human participants, and also enhance the usability of the environment by providing services such as replacing missing partners, providing services such as helping in navigation. As these virtual actors are not guided by the users, they should have sufficient behaviors to act autonomously to accomplish their tasks. This requires building behaviors for motion, as well as appropriate mechanisms for interaction.

Simulating autonomous actors should be based on biology and is directly a part of Artificial Life as well as simulation of plants, trees, and animals. This kind of research is also strongly related to the research efforts in autonomous agents [14] and behavioral animation as introduced by Reynolds [15] to study the problem of group trajectories: flocks of birds, herds of land animals and fish schools. Haumann and Parent [16] describe behavioral simulation as a means to obtain global motion by simulating simple rules of behavior between locally related actors. Wilhelms [17] proposes a system based on a network of sensors and effectors. Ridsdale [18] proposes a method that guides lower-level motor skills from a connectionist model of skill memory, implemented as collections of trained neural networks. More recently, genetic algorithms were also proposed by Sims [19] to automatically generate morphologies for artificial creatures and the neural systems for controlling their muscle forces. Tu and Terzopoulos [20] described a world inhabited by artificial fishes.

Our autonomous virtual actors [21] are able to have a behavior, which means they must have a manner of conducting themselves. Behavior is not only reacting to the environment but should also include the flow of information by which the environment acts on the living creature as well as the way the creature codes and uses this information. Behavior of autonomous actors is based on their perception of the environment as described in details in the previous text on “Autonomous Virtual Actors based on Virtual Sensors”.

Our players and the referee have a synthetic vision. Also, tennis playing with sound effects (ball-floor and ball-racket collision) and simple verbal player - referee communication has been realized. The synthetic actor can directly access to positional and semantic sound source information of an audible sound event. This allows him to localize and recognize sound sources in a reliable way and to react immediately.

A particular type of tactile sensor has been implemented in the L-system animation system where parts of the environment are modeled by a force field based particle system which, for example, animates the tennis ball dynamics of the tennis game. To simulate the tactile sensor we defined a function used also in conditions of production rules evaluating the global force field at the position given by the parameters  $x$ ,  $y$ ,  $z$  of the query symbol ? [22]. The query symbol ? makes part of environmentally sensitive L-systems and sets its parameters  $x$ ,  $y$  and  $z$  during the interpretation step of the formal symbol string to the actual position of the turtle. Thus, at the following time step, this position data can be used through the parameters in the conditions of the production rules. The force field function returns the amount of the global force field at the actual turtle position and by comparing this value with a threshold value, a kind of collision detection with force field modeled environments can be simulated.

## 6. Perception-based behaviors

Synthetic vision, audition and tactile allow the actor to perceive the environment. Based on this information, his behavioral mechanism will determine the actions he will perform. Actions may be at several degrees of complexity. An actor may simply evolve in his environment or he may interact with this environment or even communicate with other actors. As we emphasize the aspect of game, sensor-based tennis playing, game judging and navigation are the most important behaviors. A behavior can be composed of other behaviors and basic actions. A basic action is in general a motor procedure allowing an actor to move. A high level behavior uses often sensorial input, special knowledge and basic actions or other high level behaviors. To model behaviors we use an automata approach. Each actor has an internal state which can change each time step according to the currently active automata and its sensorial input. To control the high level behavior of an actor we use a stack of automata for each actor. At the beginning of the animation the user furnishes a sequence of behaviors (a script) and pushes them on the actor's stack. When the current behavior ends the animation system pops the next behavior from the stack and executes it. This process is repeated until the actor's behavior stack is empty. Some of the behaviors use this stack too, in order to reach subgoals by pushing itself with the current state on the stack and switching to the new behavior allowing them to reach the subgoal. When this new behavior has finished the automata pops the old interrupted behavior and continues. This stack based behavior control helps an actor to get more autonomous and to create his own subgoals while executing the original script.

### 6.1 Tennis playing based on vision, audition and tactile sensors

We modeled a synthetic sensor based tennis match simulation for autonomous players and an autonomous referee, implemented in the L-system based animation system. The different behaviors of the actors are modeled by automata controlled by an universal stack based control system. As the behaviors are severely based on synthetic sensors being the main channels of information capture from the virtual environment we obtain a natural behavior which is mostly independent of the internal environment representation. By using this sensor based concept the distinction between a digital actor and an interactive user merged into the virtual world becomes small and they can easily be exchanged as demonstrated with the interactive game facility.

The autonomous referee judges the game by following the ball with his vision system. He updates the state of the match when he "hears" a ball collision event (ball - ground, ball - net, ball - racket) according to what he sees and his knowledge of a tennis match, and he communicates his decisions and the state of the game by "spoken words" (sound events).

A synthetic player can also hear sound events and he obeys the decisions of the referee. The player's game automata uses synthetic vision to localize the ball's and his opponent's position and he

adaptively estimates the future ball -racket impact point and position. He uses his partner's position to fix his game strategy and to plan his stroke and his path to the future impact point.

## 6.2 Vision-based navigation

In cooperative work as a 3D puzzle (Figure 4), actors have to walk through the virtual space and it requires a navigation system. More generally, the task of a navigation system is to plan a path to a specified goal and to execute this plan, modifying it as necessary to avoid unexpected obstacles [23].

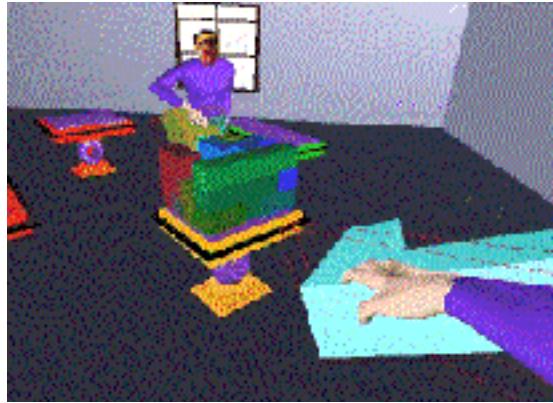


Figure 4. 3D puzzle

## 7. Making virtual actors aware of real ones

The participants are of course easily aware of the actions of the virtual humans through VR tools like Head-mounted display, but one major problem to solve is to make the virtual actors conscious of the behavior of the participants. Virtual actors should sense the participants through their virtual sensors (Figure 5).

We have seen that virtual vision is a powerful tool in modeling virtual autonomous actors. Such actors can have different degrees of autonomy and different sensing channels to the environment

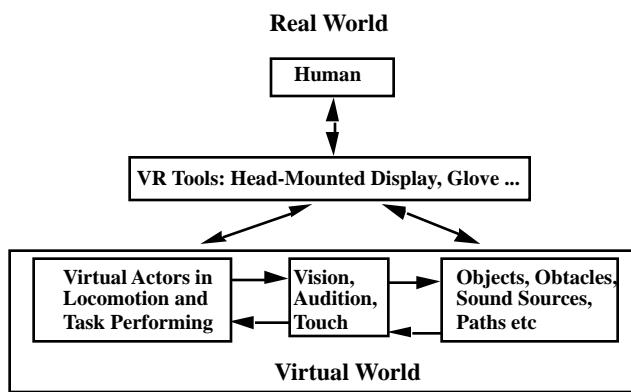


Figure 5. Real and virtual sensors

Let us now consider the case of a participant playing tennis with an autonomous virtual human. The participant can participate in VR by the head-mounted display and the earphones. He cannot get any internal VR information. His only source of knowledge from the VR is communicated by the vision, the sound, and some tactile sensory information. His behavior is strongly influenced by this sensory input and his proper intelligence. In order to process the vision of the autonomous virtual actor in a similar way than the vision of the participant, we need to have a different model. In this case, the only information obtained by the autonomous virtual actor will be through the virtual vision looking

at the participant actor. Such an autonomous virtual actor would be independent of each VR representation (as a human too) and he could in the same manner communicate with human participants and other autonomous virtual actors.

For virtual audition, we encounter the same problem as in virtual vision. The real time constraints in VR demand fast reaction to sound signals and fast recognition of the semantic it carries.

Concerning the tactile sensor, we may consider the following example: the participant places an object into the Virtual Space using a CyberGlove and the autonomous virtual actor will try to grasp it and put it on a virtual table for example. The actor interacts with the environment by grasping the object and moving it. At the beginning of interactive grasping, only the hand center sensor is active. The six palm values from the CyberGlove are used to move it towards the object. Inverse kinematics update the arm postures from hand center movement. After the sensor is activated, the hand is close enough to the object final frame. The hand center sensor is deactivated and multi-sensors on hand are now used, to detect sensor object collision.

## 8. Implementation

We exploit a distributed model of communication, therefore each user is responsible for updating its local set of data for the rendering and animation of the objects. There is always one user that determines the environment. The other users are "invited" and do not need to specify any parameters, all the environment data is initially loaded over the network to the local machine when the user is connected to the shared environment. There exists one server responsible of transmitting the actions to the participants. The communication is asynchronous. The information about the users' actions are transmitted to the server as the actions occur. The actions can be changing the position or orientation of actors, as well as grasping or releasing an object. The actions are sent to the other users by the server in terms of new orientations of the updated objects in space, as well as other possible changes such as modification to the objects. The architecture requires the broadcasting of the data from the server to all the users in the system. To overcome the problem of bottleneck when there are a lot of users, we take advantage of the geometric coherence of interactions among the actors in the 3D environment.

The network overhead can have a significant effect, especially with increasing number of users. Therefore, it is important to provide low-latency high-throughput connections. Therefore we are experimenting our system over the ATM pilot network, provided to the Swiss Federal Institute of Technology and University of Geneva, by Swiss Telecom. The ATM technology, based on packet switching using fixed-length 53-byte cells, allows to utilize traffic rates for videoconferencing, video-on-demand, broadcast video. The quality of service is achieved on demand, and guarantees a constant performance. The network has full logical connectivity at the virtual path level and initially supports PDH 34 Mbit/s and SDH 155 Mbit/s links. The pilot provides point to point links in the first phase. Multipoint links are expected to be added in the future, allowing more efficient multiple-user virtual environments. An experience was held during Telecom '95 between Singapore and Geneva using ATM.

The simulation part should also be performed efficiently using appropriate mechanism. We make use of the HUMANOID system for modeling and real-time animation of virtual actors [23]. The HUMANOID environment supports all the facilities real-time manipulation of virtual actors on a standard graphics workstation.

For realistic modeling of human shapes, we make use of deformed body surfaces attached to the human skeleton, rather than simple geometric primitives representing body links with simple skeleton. The original model is based on metaballs and splines [24] and allows parametric representation of different human bodies. The human skeleton that we use is based on anatomical structure of real skeleton without compromising real-time control; and consists of 74 degrees of freedom without the hands, with additional 30 degrees of freedom for each hand.

We include the facial interaction by texture mapping the image containing the user's face on the virtual actor's head. To obtain this, the subset of the image that contains the user's face is selected from the captured image and is sent to other users. To capture this subset of image, we apply the following method: initially the background image is stored without the user. Then, during the session, video stream images are analyzed, and the difference between the background image and the current image is used to determine the bounding box of the face in the image. This part of the image is sent to the other users after optional compression and the receiving side recognizes automatically the type of incoming images.

L-systems are given by ASCII text files which can be created and edited by any text editor. Our L-system animation system is capable to read several L-system text files and to interpret them in parallel during an animation. One L-system text file typically models for example the growth and topology of a plant or a tree or contains the rules of some actor behavior.

For fast display, we use the IRIS Performer environment [25]. Performer provides an easy-to-use environment to develop real-time graphics applications. It can extract the maximum performance of the graphics subsystem, and can use parallel processors in the workstation for graphics operations. Therefore, it is an appropriate platform to increase the performance of display part of the system.

## 9. Future Work

Further improvements are to include physical models for interaction, natural language interface with the virtual actors, and sound rendering. We also intend to incorporate more complex behaviors and communication between the three different types of virtual actors.

## Acknowledgments

The research was partly supported by ESPRIT project HUMANOID-2, Swiss National Foundation for Scientific Research, l'Office Fédéral de l'Education et de la Science, and the Department of Economy of the State of Geneva.

## References

1. Macedonia M.R., Zyda M.J., Pratt D.R., Barham P.T., Zestwitz, "NPSNET: A Network Software Architecture for Large-Scale Virtual Environments", *Presence*, Vol. 3, No. 4, 1994.
2. Gisi M. A., Sacchi C., "Co-CAD: A Collaborative Mechanical CAD System", *Presence*, Vol. 3, No. 4, 1994.
3. Stansfield S., "A Distributed Virtual Reality Simulation System for Simulational Training", *Presence: Teleoperators and Virtual Environments*, Vol. 3, No. 4, 1994.
4. Fahlen L.E., Stahl O., "Distributed Virtual Realities as Vehicles for Collaboration", *Proc. Imagina '94*, 1994.
5. Pandzic I., Çapin T., Magnenat Thalmann N., Thalmann D., "VLNET: A Networked Multimedia 3D Environment with Virtual Humans", *Proc. Multi-Media Modeling MMM '95*, World Scientific, Singapore
6. Thalmann D., Capin T., Magnenat-Thalmann N., Pandzic I.S., " Participant, User-guided, and Autonomous Actors in the Virtual Life Network VLNET ", *Proc. ICAT/VRST '95*, Chiba, Japan, November 1995, pp.3-11.
7. Noser H., Thalmann D., Turner R., Animation based on the Interaction of L-systems with Vector Force Fields, *Proc. Computer Graphics International '92*, pp. 747-761
8. Noser H., Thalmann D., L-System-Based Behavioral Animation, *Proc. Pacific Graphics 93*, Aug. 1993, World Scientific Publishing Co Pte Ltd, pp. 133-146
9. H. Noser, D. Thalmann, Simulating Life of Virtual Plants, Fishes and Butterflies, in: Magnenat Thalmann N. and D. Thalmann, *Artificial Life and Virtual Reality*, John Wiley, 1994.
10. Noser H., Thalmann D., Synthetic Vision and Audition for Digital Actors, *Computer Graphics Forum*, Vol. 14. Number 3, *Proc. Eurographics '95*, pp. 325 -336, 1995
11. P. Prusinkiewicz, A. Lindenmaier, *The Algorithmic Beauty of Plants* (1990), Springer Verlag
12. Noser H., Thalmann D., The Animation of Autonomous Actors Based on Production Rules, *Proc. Computer Animation '96*, IEEE Computer Society Press, 1996
13. Thalmann D., "Automatic Control and Behavior of Virtual Actors", *Interacting with Virtual Environments*, MacDonald L., Vince J. (Ed), 1994.
14. Maes P. (ed.) "Designing Autonomous Agents", Bradford MIT Press, 1991.

15. Reynolds C. "Flocks, Herds, and Schools: A Distributed Behavioral Model", Proc.SIGGRAPH '87, Computer Graphics, Vol.21, No4, 1987, pp.25-34
16. Haumann D.R., Parent R.E. "The Behavioral Test-bed: Obtaining Complex Behavior from Simple Rules", The Visual Computer, Vol.4, No 6, 1988, pp.332-347.
17. Wilhelms J. "A "Notion" for Interactive Behavioral Animation Control", IEEE Computer Graphics and Applications , Vol. 10, No 3 , 1990, pp.14-22
18. Ridsdale G. "Connectionist Modelling of Skill Dynamics", Journal of Visualization and Computer Animation, Vol.1, No2, 1990, pp.66-72.
19. Sims K. "Evolving Virtual Creatures", Proc. SIGGRAPH '94, 1994, pp. 15-22.
20. Tu X., Terzopoulos D. "Artificial Fishes: Physics, Locomotion, Perception, Behavior", Proc. SIGGRAPH '94, , pp.42-48.
21. Magnenat Thalmann N., Thalmann D., "Digital Actors for Interactive Television", Proc. IEEE, August 1995.
22. P. Prusinkiewicz, M. James, R. Mech, "Synthetic Topiary" , Proc. SIGGRAPH 94, pp. 351-358.
23. Boulic R., Capin T., Huang Z., Kalra P., Lintermann B., Magnenat-Thalmann N., Moccozet L., Molet T., Pandzic I., Saar K., Schmitt A., Shen J., Thalmann D., "The Humanoid Environment for Interactive Animation of Multiple Deformable Human Characters", Computer Graphics Forum, Vol.14. No 3, Proc.Eurographics '95, 1995.
24. Shen J., Thalmann D, "Interactive Shape Design Using Metaballs and Splines", Proc. Eurographics Workshop on Implicit Surfaces, Grenoble, France, 1995
25. Rohlf J., Helman J., "IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics", Proc. SIGGRAPH'94, 1994.

# A Model of Nonverbal Communication and Interpersonal Relationship between Virtual Actors<sup>1</sup>

P. Bécheiraz and D. Thalmann

Computer Graphics Lab, Swiss Federal Institute of Technology  
Lausanne, Switzerland

## Abstract

This paper presents a model of nonverbal communication and interpersonal relationship between virtual actors. Nonverbal communication improves their believability. They react not only to the presence of the other actors but also to their postures. Furthermore, their interpersonal relationships are affected by the issue of social interactions. To avoid homogenous group behaviors, each actor is set with a different character profile. We present an application of this model to create actors involved in social interactions in a virtual public garden. The animation of virtual actors is based on the library AGENTlib which is dedicated to the management of agent entities able to coordinate perception and action.

## 1 Introduction

A public garden is a place where social interactions occur. People come there for a walk and to meet each other. Therefore, it is an ideal place to study social interactions and changes in social relationships. The way social interactions take place affects relationships. The friendliness or interest of a person for another one evolves according to the attitude adopted by this other person during a communication. Its attitude might have been for instance very disappointing or surprisingly nice. As people are usually not talking loudly, for other people not involved in the conversation, it is hard to hear a word people are saying and only gestures and postures can be watched. Nevertheless, by watching this body language, it is possible to infer the type of relationship between two or three persons and how the conversation is taking place. For instance, how well they know each other or how interested they are in the conversation. Thus, the use of a model of nonverbal communication for virtual actors can increase their believability although they do not use an audible spoken language. They adopt and respond to postures. We use this model to create a virtual public garden where synthetic actors come for a walk and try to engage in communication with other actors.

Virtual actors reacting only to the presence of the others lack of believability and we believe that this can be improved by the use of elements of nonverbal communication. As for real people, virtual actors should have a specific behavior and the issue of a communication should affect them. Therefore, virtual actors are provided with a character and a description of their interpersonal relationship with other actors. Their character influences how they evaluate a posture and an emotional state describes their happiness. Actors interact by adopting postures in response to the postures of the others. The response posture corresponds to an attitude resulting from the attitude of the other person. But, in the same context, the reaction of an actor will be different from that of another because its character influences its style of nonverbal communication and its perception of postures and the postures it adopts. The interpersonal relationship between two actors determines if they begin a communication and the issue of this communication affects their relationship.

The remainder of the paper is organized as follows. In the next section, we review some papers related to behavioral animation and social behaviors. In section 3, we give a definition of nonverbal communication and we describe the elements of nonverbal communication we focus on. In section 4, we present the behavioral model we implemented to control the social behavior of actors using nonverbal communication. We explain how they evaluate and select postures according to their character and how their relationships evolve. In section 5, we describe how the public garden and the

<sup>1</sup> An expanded version of this text has appeared in: P.Bécheiraz, D. Thalmann, The Use of Nonverbal Communication Elements and Dynamic Interpersonal Relationship for Virtual Actors, Proc. Computer Animation '96, IEEE Computer Society Press, 1996.

actors are modeled with the libraries SCENElib and BODYlib [4] dedicated to the creation of scenes of 3D objects and human actors and how actors are animated with the library AGENTlib. The purpose of this library is the management of agents able to coordinate perception and action and the scheduling of execution of concurrent or sequential actions. Finally, in section 6, we illustrate this model of nonverbal communication with an example of a communication involving three actors.

## 2 Background

The use of behavioral animation to generate computer animation is a well explored domain. Reynolds [18] described the first use of a behavioral model to produce a flocking behavior. Other papers focused mainly on specific motion such as locomotion, grasping, and lifting [3, 6, 8, 13, 17]. Tu and Terzopoulos [19] created autonomous fishes living in a physically modeled virtual marine world. Hodgins et al. [10] described dynamic athletic behaviors. Unuma et al. [20] modeled human figure locomotion with emotions. Other papers presented systems to interact with virtual creatures. Maes et al. [11] designed a full-body interaction system with a virtual dog. Perlin [16] described virtual actors appearing to be emotionally responsive. Random noise functions were used together with rhythmic motions to generate lifelike behaviors. Research in autonomous robots have also focused on social behaviors. Mataric [12] described complex group behavior produced from simple local interactions. Dautenhahn [7] studied interactions between individualized robots able to recognize each other in order to establish relationships. Our model of nonverbal communication attempts to go further. The believability of virtual actors is improved by their capability to interpret and use a nonverbal language.

## 3 Nonverbal communication

### 3.1 Definition and examples

A nonverbal communication is concerned with postures and their indications on what people are feeling. Postures are the means to communicate and are defined by a specific position of the arms and legs and angles of the body. Usually, people don't use consciously nonverbal communication, but they instinctively understand it to a considerable extent and respond to it without any explicit reasoning. As stated in [2], as much as 65 percent of the substance of a face to face communication is conveyed by nonverbal elements and many psychologists suggest that a posture is the result of an underlying attitude and is an important means of conveying interpersonal attitudes, an attitude being a mental state inducing a particular posture. The effects of nonverbal communication, though unconscious, can't therefore be ignored.

Now, let us examine some typical examples of nonverbal behaviors between two and three persons. The first example describes two persons talking to each other. Two persons engaged in an intimate conversation often stand directly facing each other. The orientation of their bodies and feet shows how much they are interested in the conversation. The body and feet of the first are oriented towards the other, showing that he or she seems to be interested in him or her. Meanwhile, one foot of the second begins to shift, pointing that the interest of the second person is waning or the second one finds the topic under away, probably meaning discussion not very interesting or exciting and would like to escape. The second example is concerned with two persons communicating with each other and rejecting a third one. Two persons are standing in an open triangle formation commonly used for friendly conversation. It leaves a potential opening for a third person to join them. A third person comes near them, holding his hands clasped in front of him in a defensive manner. He attempts to join the group, but the first two respond by rejecting him out. One raises his left arm, forming a barrier. The other deliberately avoids making eye contact with the newcomer. They have formed a closed formation, suppressing any possibility for a new person to join them. The newcomer feels rejected and leaves unhappily. The third example involves a man and a woman. The man tries to court the woman. But his postures are aggressive. He is raising one arm to the woman, cutting off one possibility of escape. The other hand is on his hip, emphasizing the sexual threat. The woman reacts with a defensive body language. She turns her head to avoid eye contact and takes one step backward.

### 3.2 Elements of nonverbal communication

The elements we use for a nonverbal communication are the relative positioning of both persons and their postures. Furthermore, the emotional state of an actor affects the way it walks [5, 20]. A low emotional state will give a sad look to its walking and a high one a happy look. Figure 1 shows the two extremes of walking.



Figure 1. Walking affected by emotional state

#### 3.2.1 Relative positioning

The relative positioning consists of proximity and orientation. There is a range within which variation of these two dimensions is possible, and this variation denotes modifications in interpersonal relationship.

The proximity dimension is the distance between two persons. Hall [9] suggested that there are four proximity zones. The first zone, within 50 centimeters in front of the body, is for intimate relationship, reserved for partners, children, or close family members. The second zone is for close relationship like friends and most conversations will take place within this personal zone between 50 centimeters and 1.2 meters away. More formal and impersonal interactions occur within a social zone between 1.2 and 2.75 meters away and a public zone within 2.75 meters away and above. The distance between individuals is linked to their relationship. For instance, if one person likes another, his or her relationship with him or her is good, resulting in a close proximity. But, if the other person comes too close, the proximity becomes too small, so that the first person backs away.

The orientation dimension is the angle at which one person faces another, usually the angle between a line joining both persons and the plane of the shoulder of the first person. This orientation refers to the body, not the head or eyes. An angle of  $0^\circ$  means that both persons are face to face and an angle of  $90^\circ$  means that the orientation of the first person is perpendicular to the orientation of the second person. People tend to turn more towards those they find interesting or attractive than towards those they don't. The orientation of a person towards another is a means to evaluate its interest. The person engaging the communication will turn towards the other which, depending on his or her interest or desire to communicate, will turn more or less towards the first.

As actors are involved in close communications, we use the first three zones. The first two zones are used by actors with strong or good relationship and the third zone for actors with average or bad relationship. Actors change their proximity by taking steps forward or backward and their orientation by turning from one or more steps to the right or to the left.

#### 3.2.2 Postures

As stated by Mehrabian [14], the main dimension of postures is an open-closed dimension. The open-closed dimension provides or denies access to body spaces by opening up to other people or hiding behind self-imposed barriers. A barrier can be constructed by folding the arms across the chest or by clasping the hands in front of oneself. These barriers symbolically keep other people out and

hold self feelings in. The openness consists of components like a direct orientation, a forward lean, an openness of arms and legs. Openness indicates a positive attitude, especially in male-female encounters. Machotka and Spiegel [15] described more precisely standing male and female postures which vary by the positions of the arms. They described the effects of the position of the arms of a female on permitting or denying access to body spaces. Nine postures were defined to represent all the combinations of the three arm positions assumed to be important for a female person: the left arm extended, resting at the side and parallel to the body or covering the breast, and the right arm extended, resting at the side and behind the body or covering the genital area. Similarly, they defined postures for a male person. Nine postures were defined: the arms crossed on the chest, planted on the hips, covering the genital area, clasped at the back, hanging at the side, clasped at the back slightly on the left, one arm extended, both arms slightly or fully extended. These postures were evaluated over bipolar normalized scales. Each pole of a dimension was qualified with an attribute. A rating of 0 means that the attribute on the left of the scale is fully appropriate to describe the posture, and a rating of 1 means that the attribute on the right is. The female and male postures were evaluated with many dimensions. Machotka and Spiegel presented drawings of the postures to a group of male and female persons and asked them to rate the postures for all the dimensions. For our model of nonverbal communication, we kept four dimensions that we thought are meaningful. These dimensions are: rejecting / receiving, other-concerned / self-concerned, cold / warm, pulled away / pulled towards.

## 4 Behavioral model

### 4.1 Actor

We define an actor as an entity capable of perceiving and performing actions and provided with a behavioral model. The behavioral model is responsible for the coordination between perception and action and the action selection. Each actor is provided with the same behavioral model. But, in order to avoid homogenous behaviors, each actor distinguishes itself from the other actors by its attitude, its character, its emotional state, its relationships with the other actors and its desire to communicate.

As actors are involved in social relationship, we define the attitude of an actor as a mental state corresponding to its friendliness for another actor and leading it to adopt a particular posture. A hostile attitude leads to unfriendly postures and a friendly attitude leads to friendly postures. An attitude is a normalized value. A value of 0 denotes a hostile attitude and a value of 1 denotes a friendly attitude.

The character of an actor affects how it selects a posture from its attitude and how it evaluates the underlying attitude of the posture of another actor. This character is defined by weights given to the four dimensions rating the postures. A weight is a normalized value and denotes the importance given by the actor to a particular dimension. The dimensions with high weights have a greater effect on the selection or the evaluation of a posture than those with low weights. When an actor evaluates the attitude of another actor, the evaluated attitude may be different from the actual attitude of the other actor because both actors have a character set with other weights.

The emotional state denotes the general desire of an actor to communicate. The emotional state is represented with a normalized value. A low emotional state denotes a weak desire to communicate. If an actor nevertheless communicates, its postures are unfriendly. On the other hand, a high emotional state induces friendly postures. The emotional state changes dynamically from the issue of a communication. If an actor involved in a communication gets friendly postures from another actor all along, its emotional state increases. Similarly, if the actor gets unfriendly postures, its emotional state decreases. Each actor maintains a description of its relationship with each other actor. A relationship is represented with a normalized value. A low value denotes that the actor has a bad relationship with the other actor and a high value denotes a good relationship. As for the emotional state, the value of a relationship is updated according to the development and the issue of a communication. The desire of an actor to communicate with another actor is evaluated with its emotional state and its relationship with it. An actor with a low emotional state may nevertheless communicate if its relationship is good.

On the other hand, an actor with a high emotional state but a bad relationship may decide not to communicate.

In the next sections, we use the following notations for the parameters of an actor. The attitude of an actor A is denoted  $\text{att}_A$ , its posture, proximity and orientation when facing another actor are denoted  $\text{post}_A$ ,  $\text{prox}_A$  and  $\text{or}_A$  respectively, its character is denoted  $\text{char}_A$ , its emotional state is denoted  $\text{em}_A$ , its interpersonal relationship with an actor B is denoted  $\text{ir}_{AB}$  and its desire to communicate is denoted  $\text{des}_A$ .

## 4.2 Behavior selection

The typical behavior of a person going to a public garden can be described as follows. A person mainly goes to a public garden for a walk. But presumably, he or she may also have social contact with others. He or she might have made an appointment with friends or meet them there by chance. He or she might approach an unknown but attracting person. Then, he or she probably engages in a communication with another person. Typical social behaviors happen. When seeing a friend, a person might go to him or her and have a chat. After a moment, one might perhaps feel bored and wish to go away. A third person might also try to join in the conversation and will be either accepted or rejected by the first two. If their relationship with the new one is bad or their conversation is very intimate, he or she will probably be rejected. The new person might also be accepted and becomes the centre of attention. One of the first two persons might therefore feel rejected and leaves.

We implemented a behavioral model of this social behavior. The actions performed by an actor are selected by a general behavior composed of a social behavior and a collision avoidance behavior. Only one of these behaviors is active at any time. The collision avoidance behavior takes precedence over the social behavior. When a risk of collision occurs, this behavior is selected while the social behavior is temporarily suspended and its status is stored in order to continue when the avoidance behavior ends [19]. The social behavior is composed of two simpler behaviors. A walking and a communication behavior. The walking behavior is used either for wandering or to go to another actor in order to communicate with it. When the other actor is reached, the communication behavior is selected and the first actor starts communicating with the other.

When the active behavior of an actor A is the walking behavior, actor A is initially wandering. After a while, actor A sees another actor B and evaluates its desire  $\text{des}_A = f(\text{em}_A, \text{ir}_{AB})$  to communicate with actor B. If this desire is strong enough, actor A goes to actor B. But, during its walk, it might see another actor C and evaluates its desire to communicate with actor C. As actor A has a greater desire to communicate with actor C than with actor B. Actor A stops its walk to actor B and goes to actor C. When actor C is reached, the communication starts.

Then the communication behavior is the active behavior. As suggested in [2], in a friendly or close conversation each person imitates the attitude of the other. Therefore, actor A mainly tries to improve the attitude of actor B. First, actor A evaluates an initial attitude  $\text{att}_A = f(\text{des}_A)$ , an initial proximity  $\text{prox}_A = f(\text{des}_A)$  and an initial orientation  $\text{or}_A = f(\text{des}_A)$ . The attitude  $\text{att}_A$  is used to select an initial posture  $\text{post}_A = f(\text{att}_A)$ . After this initialization phase, a communication loop starts. Actor A perceives the posture of actor B and matches it to a posture of the predefined postures repertoire. The underlying attitude  $\text{att}_B$  of the posture of actor B is evaluated by a function  $\text{att}_B = f(\text{char}_B, \text{post}_B)$ . Then, actor A reevaluates its emotional state with a function  $\text{em}_A = f(\text{att}_A, \text{att}_B)$ . Depending on its current attitude  $\text{att}_A$  and the evaluated attitude  $\text{att}_B$ , the emotional state  $\text{em}_A$  can increase or decrease. Then, actor A reevaluates its desire  $\text{des}_A$ . If a third actor C is also communicating with actor A and B, actor A evaluates its desire for actor B and actor C and continues the communication with the actor for which it has the highest desire. If its desire is less than a minimal desire or if the evaluated attitude  $\text{att}_B$  of actor B is greater than a maximal desired attitude or actor B stopped the communication with actor A the communication concludes. Otherwise, actor A reevaluates a new attitude  $\text{att}_A$  and its

related posture  $pos_A$ , a new proximity  $prox_A$  and a new orientation  $or_A$ . Figure 2 shows the communication behavior.

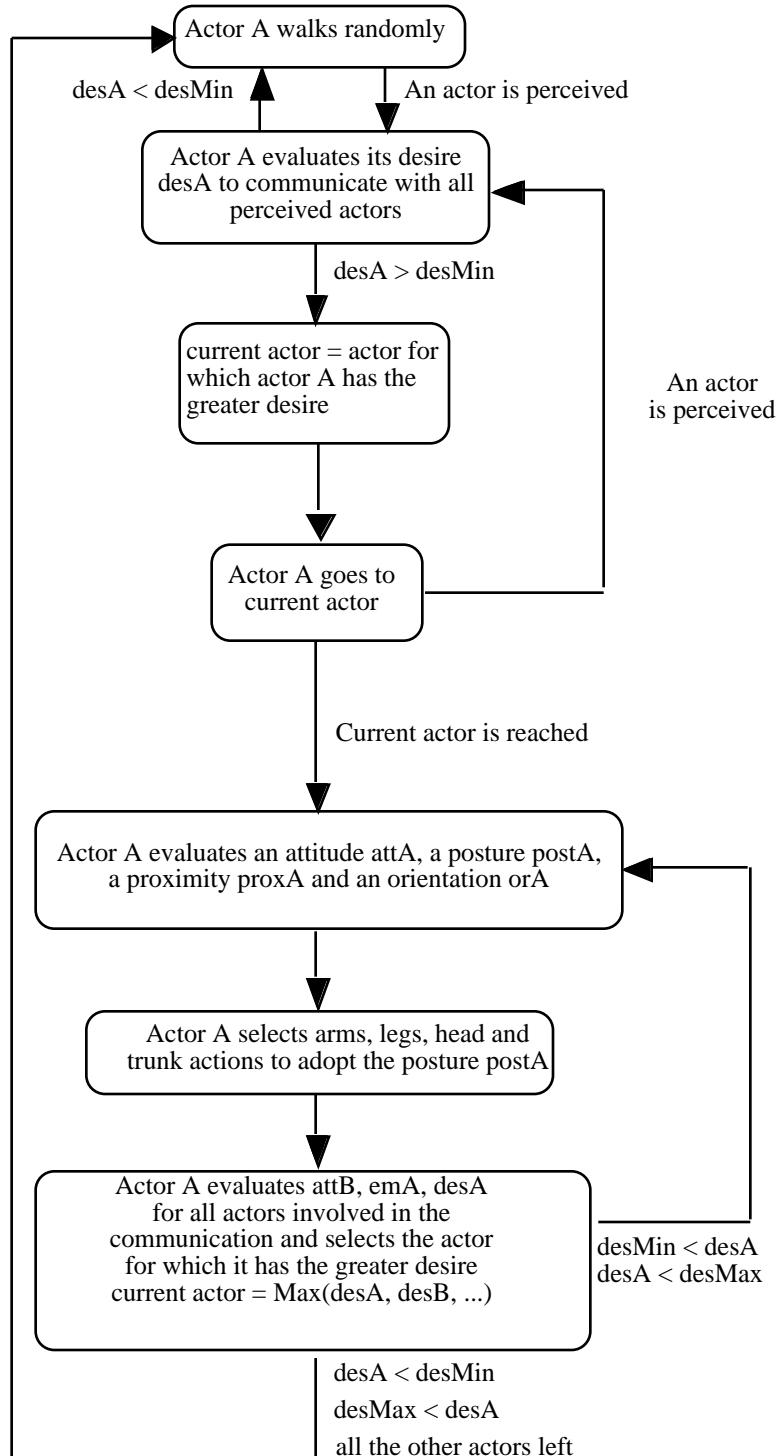


Figure 2. Communication behavior

### 4.3 Evaluation functions

An actor A decides to communicate with an actor B if its desire is strong enough. The desire is evaluated with the emotional state of actor A and its relationship with actor B. The desire reflects the current emotional state of actor A and is either decreased when the relationship with actor B is bad or increased when the relationship is good. The evaluation of the desire is given by:

$$\text{des}_{AB} = \begin{cases} \text{em}_A + \text{em}_A * (\text{ir}_{AB} - 0.5) & \text{ir}_{AB} < 0.5 \\ \text{em}_A + (1 - \text{em}_A) * (\text{ir}_{AB} - 0.5) & \text{ir}_{AB} > 0.5 \end{cases} \quad (1)$$

The attitude of actor A is simply set equal to its desire to communicate with actor B. The evaluation of the attitude is given by:

$$\text{att}_A = \text{des}_{AB} \quad (2)$$

The posture postA which actor A selects is the posture with the closest rating to its attitude attA as evaluated by its character. In the formula (3), wd is the weight given to the dimension d, ratdp is the rating of the dimension d given to the posture p.

$$\text{post}_A = \underset{d}{\text{Min}} \frac{\text{Min}_p |\text{att}_A - \text{rat}_{dp}|}{w_d} \quad (3)$$

The proximity and the orientation of actor A are affected by its desire desA to communicate with actor B. When this desire is high, actor A draws near and turns more towards actor B. Its proximity and its orientation are increased. When this desire is low, actor A moves off and turns away from actor B. Its proximity and its orientation are decreased. They are calculated by formulas (7) and (8). The range proxmax and proxmin and the range ormax and ormin depends of the desire desA and are set by formulas (4), (5) and (6).

$$0.75 < \text{des}_A < 1 :$$

$$\begin{aligned} \text{prox}_{\text{max}} &= 0 & \text{prox}_{\text{min}} &= 0.5 \\ \text{or}_{\text{max}} &= 0 & \text{or}_{\text{min}} &= /8 \end{aligned} \quad (4)$$

$$0.5 < \text{des}_A < 0.75 :$$

$$\begin{aligned} \text{prox}_{\text{max}} &= 0.5 & \text{prox}_{\text{min}} &= 1.2 \\ \text{or}_{\text{max}} &= /8 & \text{or}_{\text{min}} &= /4 \end{aligned} \quad (5)$$

$$0 < \text{des}_A < 0.5 :$$

$$\begin{aligned} \text{prox}_{\text{max}} &= 1.2 & \text{prox}_{\text{min}} &= 2.75 \\ \text{or}_{\text{max}} &= /4 & \text{or}_{\text{min}} &= /2 \end{aligned} \quad (6)$$

$$\text{prox}_A = \frac{\text{prox}_{\text{max}} - \text{prox}_{\text{min}}}{\text{des}_{\text{max}} - \text{des}_{\text{min}}} * \text{des}_A + \frac{\text{prox}_{\text{min}} * \text{des}_{\text{max}} - \text{prox}_{\text{max}} * \text{des}_{\text{min}}}{\text{des}_{\text{max}} - \text{des}_{\text{min}}} \quad (7)$$

$$\text{or}_A = \frac{\text{or}_{\text{max}} - \text{or}_{\text{min}}}{\text{des}_{\text{max}} - \text{des}_{\text{min}}} * \text{des}_A + \frac{\text{or}_{\text{min}} * \text{des}_{\text{max}} - \text{or}_{\text{max}} * \text{des}_{\text{min}}}{\text{des}_{\text{max}} - \text{des}_{\text{min}}} \quad (8)$$

The evaluation by an actor A of the attitude attAB underlying a posture of another actor B is inspired by the model of Fishbein [1]. The attitude is evaluated by formula (9) with the ratings ratdp given by each dimension d rating the posture p and the weights wd given by the character of the actor to these dimensions.

$$\text{att}_{AB} = \frac{\sum_d \text{rat}_{dp} * w_d}{\sum_d w_d} \quad (9)$$

During the communication, the emotional state of actor A is evaluated by formula (10) with its attitude attA and the attitude attB of actor B. When the attitude of actor B is greater than the attitude of actor A, the emotional state is increased. In the other case, the emotional state is decreased.

$$em_A = \begin{cases} em_A + em_A * (att_B - att_A) & att_A > att_B \\ em_A + (1 - em_A) * (att_B - att_A) & att_A < att_B \end{cases} \quad (10)$$

At the end of the communication the interpersonal relationship is increased if the emotional state increased during the communication or decreased if the emotional state decreased during the communication. The update of the interpersonal relationship is given by:

$$ir_{AB} = \begin{cases} ir_{AB} + ir_{AB} * (em_{final} - em_{initial}) & em_{initial} > em_{final} \\ ir_{AB} + (1 - ir_{AB}) * (em_{final} - em_{initial}) & em_{initial} < em_{final} \end{cases} \quad (11)$$

## 5 Implementation with the AGENTlib architecture

The public garden and the actors are modeled with the libraries SCENElib and BODYlib dedicated to the creation of scenes of 3D objects and the creation of human actors. Actors are animated with the library AGENTlib dedicated to the coordination of perception and action and the combination of motion generators. The libraries SCENElib and BODYlib are part of the HUMANOID environment [4]. This environment supports several facilities including the management of multiple humanoid entities and the skin deformation of a human body. The library AGENTlib defines agent entities responsible of the integration of a fixed set of perception senses and a dynamic set of actions. For this implementation we use a simple model of perception. During a nonverbal communication, an actor needs just to know the position, the orientation and the posture of another actor. Therefore, a perception is performed with a direct access to the local database of another actor. An action encapsulates a motion generator. An agent entity maintains actions in two lists. A list of inactive actions and a list of active actions. Actions can be performed concurrently or sequentially according to activation rules. Concurrent actions are performed with priority rules. The contribution of their motion generator can be mixed with weights given to each action or the contribution of an action can take precedence over the contribution of another action. The transitions between sequential actions can be smooth or full. When the transition is smooth, the contribution of the terminating action decreases while the contribution of the starting action increases. In case of a full transition, the new action starts only when the previous action is terminated.

We defined actions for a walking motion and arms and legs postures. The walking action encapsulates a walking motion generator [5]. The walking is directed with a linear and an angular speed and its pattern can be personified. We use this facility to reflect the emotional state of the actor. When its emotional state is low, the actor walks sadly. When its emotional state is high, the actor walks happily. The arms and legs actions are performed with keyframe motions. We captured keyframe motions for legs actions to reflect the changes of proximity and orientation and keyframe motions to take any of the postures defined in [15].

## 6 Example

We illustrate the use of this model of nonverbal communication with an interaction between three actors we call Rod, Marilyn and James. First, we set the character, the emotional state and the relationships of each actor.

The three actors are wandering. After a while, Rod perceives Marilyn. Both actors evaluate their desire to communicate. As their desire is strong enough, they walk to each other. When Rod reaches Marilyn, the communication starts. Both actors evaluates their attitude, posture, proximity and orientation. Then, they move to adopt the selected posture and adjust their proximity and orientation. Rod extends both arms slightly and steps forward. Marilyn covers her breast with the left arm, covers her genital area with the right arm and steps backward and turn. Figure 5 shows Rod and Marilyn. Each actor evaluates the underlying attitude of the posture of the other actor and updates its emotional state and desire.

James joins. As his desire to communicate with Marilyn is greater than his desire to communicate with Rod, James communicates with Marilyn. The three actors evaluate their attitude, posture, proximity and orientation. They move to adopt the selected posture and adjust their proximity and orientation. Rod lets his arms hang at the side and steps forward. Marilyn covers her breast with the left arm, puts the right arm behind her body and steps backward and turns, James crosses his arms on the chest and steps forward. Figure 6 shows Rod, Marilyn and James.

All actors evaluate their attitude, emotional state and desire for the two other actors and select the actor with the greatest desire to continue the communication.



Figure 5. Rod and Marilyn

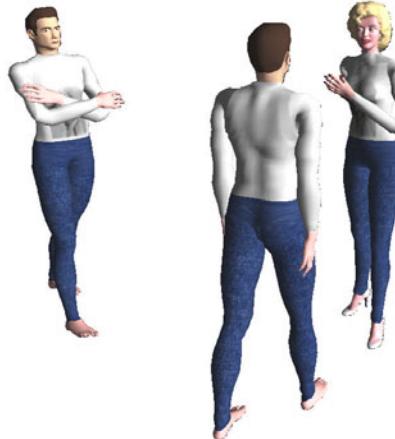


Figure 6. Rod, Marilyn and James

As the desire of Marilyn to communicate with James is greater than her desire to communicate with Rod and the desire of James to communicate with Marilyn is greater than his desire to communicate with Rod, Marilyn and James now communicate together. Rod is rejected from the communication, he updates his relationship with Marilyn and leaves. Marilyn and James move. Marilyn lets her left arm rest at the side, keeps the right arm behind her body and turns towards James, James keeps his arms crossed on the chest and steps forward. Figure 7 shows Marilyn and James.



Figure 7. Marilyn and James

## 7 Conclusion

We have described a model of nonverbal communication and interpersonal relationship between virtual actors. From psychological studies on nonverbal communication, we proposed a behavioral

model intended to increase the believability of virtual actors involved in social interactions by making them react not only to the presence of other actors but also to their postures. Furthermore, their interpersonal relationships are not static but evolve as they are affected by the issue of interactions. We demonstrated the application of this model to create actors interacting in a public garden.

Current research includes the evaluation of the relevance of this model for a greater number of actors. In a future work, we will extend this model to interactions between real persons and virtual actors. The real person will be represented by a virtual actor. A motion capture software will map the real person posture onto the virtual actor.

## 8 Acknowledgments

The authors would like to thanks Tom Molet who captured the keyframe motions, Ronan Boulic and Luc Emeling for their work on the AGENTlib library and Eric Chauvineau for his model of skin deformation. This research was supported by the Swiss National Foundation for Scientific Research, and the Federal Office for Education Science (Project ESPRIT Humanoid-2).

## References

- [1] I. Ajzen, M. Fishbein. Understanding attitudes and predicting social behavior, Prentice-Hall, 1980.
- [2] M. Argyle. Bodily Communication. Methuen and Co Ltd., 1975.
- [3] N. I. Badler, C. Phillips, and B. L. Webber. Simulating Humans: Computer Graphics, Animation, and Control. 1993, Oxford University Press, New York.
- [4] R. Boulic, T. Capin, Z. Huang, P. Kalra, B. Lintermann, N. Magnenat-Thalmann, L. Moccozet, T. Molet, I. Pandzic, K. Saar, A. Schmitt, J. Shen and D. Thalmann. The HUMANOID Environment for Interactive Animation of Multiple Deformable Human Characters. Proceedings of EUROGRAPHICS 95, p. 337-348 (Maastricht, The Netherlands, August 28-September 1, 1995).
- [5] R. Boulic, N. Magnenat-Thalmann and D. Thalmann. A Global Human Walking Model with Real Time Kinematic Personification. *The Visual Computer*, Vol. 6, 1990.
- [6] Bruderlin, Armin and Thomas W. Calvert. Dynamic Animation of Human Walking. Proceedings of SIGGRAPH 89, p. 233-242 (Boston, MA, July 31-August 4, 1989).
- [7] K. Dautenhahn. Trying to Imitate - a Step towards Releasing Robots from Social Isolation. Proceedings of From Perception to Action Conference, p. 290-301 (Lausanne, Switzerland, September 7-9, 1994).
- [8] Girard, Michael and A. A. Maciejewski. Computational Modeling for the Computer Animation of Legged Figures. Proceedings of SIGGRAPH 85, p. 263-270 (San Francisco, CA, July 22-26, 1985).
- [9] E. T. Hall. The Silent Language. Garden City, Doubleday, 1959.
- [10] J. K. Hodgins, W. L. Wooten, D. C. Brogan, J. F. O'Brien. Animating Human Athletics. Proceedings of SIGGRAPH 95, p. 71-78 (Los Angeles, CA, August 6-11, 1995).
- [11] P. Maes, T. Darell, B. Blumberg, A. Pentland. The ALIVE System: Full-Body Interaction with Autonomous Agents. Proceedings of Computer Animation 95, p. 11-18 (Geneva, Switzerland, April 19-21, 1995).
- [12] M. J. Mataric. From Local Interactions to Collective Intelligence. From the book: The biology and Technology of Intelligent Autonomous Agents. Edited by Luc Steels. NATO ASI Series F, p. 275-295, Vol. 144, 1995.
- [13] McKenna, Michael and David Zeltzer. Dynamic Simulation of Autonomous Legged Locomotion. Proceedings of SIGGRAPH 90, p. 29-38 (Dallas, TX, August 6-10, 1990).
- [14] A. Mehrabian. Nonverbal communication. Aldine-Atherton, 1972.
- [15] J. Spiegel, P. Machotka. Messages of the body. The Free Press, Macmillan, 1974.
- [16] K. Perlin. Interacting with virtual actors. Visual Proceedings of SIGGRAPH 95, p. 92-93 (Los Angeles, CA, August 6-11, 1995).
- [17] M. H. Raibert and Jessica K. Hodgins. Animation of Dynamic Legged Locomotion. Proceedings of SIGGRAPH 91, p. 349-358 (Las Vegas, NV, July 28-August 2, 1991).
- [18] C. W. Reynolds. Flocks, Herds, and Schools: A Distributed Behavioral Model. Proceedings of SIGGRAPH 87, p. 25-34 (Anaheim, CA, July 27-31, 1987).
- [19] X. Tu and D. Terzopoulos. Artificial Fishes: Physics, Locomotion, Perception, Behavior. Proceedings of SIGGRAPH 94, p. 43-50 (Orlando, FL, July 24-29, 1994).
- [20] M. Unuma, K. Anjyo, R. Takeuchi. Fourier Principles for Emotion-based Human Figure Animation. Proceedings of SIGGRAPH 95, p. 91-96 (Los Angeles, CA, August 6-11, 1995).

# Interacting with Virtual Humans through Body Actions<sup>1</sup>

Luc Emeling, Ronan Boulic, Daniel Thalmann

LIG - Computer Graphics Lab., Swiss Federal Institute of Technology, Lausanne, CH-1015 Switzerland  
 {emeling,boulic,thalmann}@lig.di.epfl.ch

Interactive Virtual environments suffer, since their inception, from being unable to interpret gestures made by their users. Quite a few tentatives have been investigated with success but most of them are limited to a specific set of body parts like hands, arms or facial expressions. However when projecting a real participant into a virtual world to interact with the synthetic inhabitants, it would be more convenient and intuitive to use body-oriented actions. To achieve this goal we developed a hierarchical model of human actions based on fine-grained primitives. An associated recognition algorithm allows on-the-fly identification of simultaneous actions.

## VR interfaces

To date, basically two techniques exist to capture the human body posture in real-time. One uses video cameras which deliver either conventional or infrared pictures. This technique has been successfully used in the ALIVE system (cf. [1]) to capture the user's image. The image is used for both the projection of the participant into the synthetic environment and the extraction of cartesian information of various body parts. If this system benefits from being wireless, it suffers from visibility constraints relative to the camera and a strong performance dependence on the vision module for information extraction. The second technique is based on sensors which are attached to the user. Most common are sensors measuring the intensity of a magnetic field generated at a reference point. The measurements are transformed into position and orientation coordinates and sent to the computer. This raw data is matched to the rotation joints of a virtual skeleton by the means of an anatomical converter (cf. [2]). This is the approach we use currently for our interactive VR testbeds.

Figure 1 shows a snapshot of a life participant with ten sensors used to reconstruct the avatar in the virtual scene. The participant performs fight gestures which are recognized by the virtual opponent [3]. The latter responds by playing back a pre-recorded keyframe sequence. The most disturbing factors of this system are the setup time to fix all the sensors and the wires hanging around during the animation. However wireless systems are already available which solve these problems. For the interactive part of the VR testbed we developed a model of body actions as base of the recognition system.

## A Model of Body Actions

By analyzing human actions we have detected three important characteristics which inform us about the specification granularity needed for the action model. First, an action does not necessarily involve the whole body but may be performed with a set of body parts only. Second, multiple actions can be performed in parallel if they use non-intersecting sets of body parts. Finally a human action can already be identified by observing strategic body locations rather than skeleton joint movements. Based on these observations, a top-down refinement paradigm appears to be appropriate for the action model. The specification grain varies from coarse at the top level to very specialized at the lowest level. The number of levels in the hierarchy is related to the feature information used. At the lowest level, we use the skeleton degrees of freedom (DOF) which are the most precise feature information available (30-100 for a typical human model). At higher levels, we take advantage of strategic body locations like the center of mass and end effectors, i.e. hands, feet, the head and the spine root. Table 1 shows the feature levels used in the model hierarchy.

---

<sup>1</sup> An expanded version of this text has appeared in L. Emeling, R. Boulic, D. Thalmann, Interacting with Virtual Humans through Body Actions, IEEE Computer Graphics and Applications, 1998 , Vol.18, No1, pp8-11.



Figure 1. Interactive environment with the video image of the participant using ten motion capture sensors (right of a & b), his avatar (middle of a & b) and the virtual opponent (left of a & b)

		Level	Level description
Action	Gesture	1	Center of Mass velocity
		2	End Effector velocities
	(final) Posture	3	Center of Mass position
		4	End Effector positions
		5	Joint values

Table 1 The five characteristic levels of an action

Human activity is composed of a continuous flow of actions. This continuity makes it difficult to define precise initial and in-between postures for an action. As a consequence we consider only actions defined by a gesture, a posture or a gesture followed by a posture.

The action model of relies on cartesian data and joint angle values. For a given action this data is not unique but depends on the anatomical differences of the live performers. A solution is, to normalize all cartesian action data by the body height of the performer. This is reasonable as statistical studies have measured a significant correlation between the body height and the major body segment lengths. Also it's important to choose adequate reference coordinate frames. We use three coordinate systems (cf. Figure 2): the Global, the Body and the Floor Coordinate System (in short GCS, BCS, FCS). The BCS is attached to the spine base of the body and its up axis is aligned with the main spine direction. The FCS is located at the vertical projection of the spine root onto the floor level and reuses the GCS Up axis.

Each level of the action model defines action primitives. At the gesture level (Table 1, levels 1 & 2) an action primitive is the detection of the motion of the center of mass (CoM) or an End Effector (EE) along a specific direction. They are of the form (CoM, velocity direction) or (EE<sub>i</sub>, velocity direction), where *i* denotes one of the end effectors. If the average motion is above a normalized threshold, the velocity direction is assigned to one of the following values: upward, downward, forward, backward, leftward, rightward. For the head end effector it's preferable to use rotation directions of a 'look-at' vector rather than velocity directions, in order to specify messages like 'yes' or 'no'. Additionally a not\_moving primitive detects a still CoM or EE. Thus the gesture of an action is described by an explicit boolean expression of gesture primitives, e.g.:

'Body downward motion' = (CoM, downward)

'Walking motion' = ((spine\_root, forward) AND (left foot, forward))

OR ((spine\_root, forward) AND (right foot, forward))

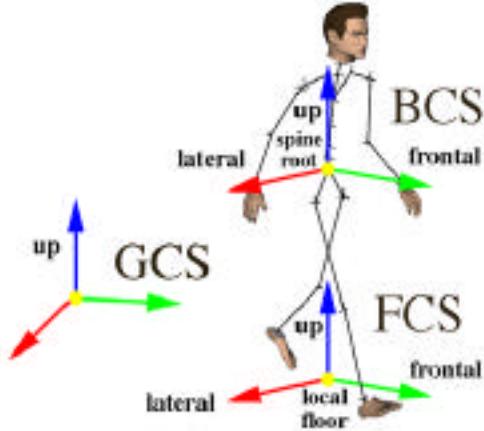


Figure 2. The human skeleton model and the three normalized coordinate systems

At the posture level (Table 1, levels 3, 4 & 5) an action primitive is the cartesian position of the CoM or the EE's or the joint values of the body posture. As it is not convenient to specify position or joint information explicitly, we use a 'specify-by-example' paradigm: we build a database of posture prototypes and extract the posture primitives automatically.

### Action Recognition

During the recognition phase we keep a trace of the actions which are potential candidates for the recognition result. Initially this Candidate Action Set (CAS) is a copy of the complete action database. Then the candidate selection is performed sequentially on the five levels of the action model, starting with level 1 (Figure 3). Here we take advantage of the hierarchical nature of the action model: at the higher levels the CAS is large, but the data to be analyzed is small and therefore the matching costs are low. At the lower levels a higher matching cost is acceptable because the CAS has considerably shrunken.

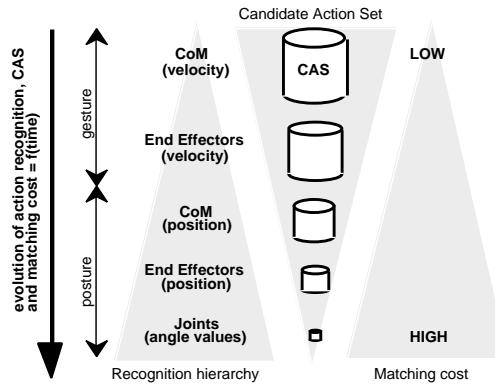


Figure 3. Action Recognition structure

- Gesture Levels Matching. Here we compute the current gesture primitives of the avatar and evaluate the actions' gesture definition. All action candidates whose boolean expression of gesture primitives results in a *False* value are removed from the CAS. Action candidates without a gesture definition remain in the CAS.
- Posture Levels Matching: For all the actions in the CAS, the algorithm computes the squared distance between their stored final CoM position and the current CoM position. The selection retains all the candidate actions for which the distance is smaller than a selectivity radius  $R$  given by:

$$R = \text{Min} + (1-S) * (\text{Max} - \text{Min})$$

*Min* and *Max* are the smallest and largest squared distances. *S* is a normalized selectivity parameter within [0,1]. The same algorithm is applied to the levels four and five with possibly a different selectivity factor. The only difference resides in the dimension of the cartesian vector: 3D for the CoM, 18D for the EEs (concatenation of 6 end effector 3D positions) and 74D for the joints (74 degrees of freedom of the body model). Note that this algorithm always selects at least one posture among the posture candidates. In practice it means that the posture database should always contain some elementary postures: if in a VR session the participant is standing still most of the time then the database should contain a 'stand still' posture even if this posture is not used as recognition result in the application. Always selecting a winner posture is a desirable property in interactive environments because participants are more tolerant to an action mis-interpretation than ignorance. Nevertheless it is possible to add a test which discards winner postures if the deviation between the current and the database posture is beyond some threshold.

If at any level the CAS happens to get empty, the algorithm reports 'unknown action' as output. Simultaneous actions can be detected as long as they act on complementary body parts. For example a 'right hand phoning' action can be defined by a posture involving the right arm and the neck (levels four and five). Another action is 'walking' defined by a (CoM, forward) primitive and the 'walking motion' expression. So, whenever the performer walks while phoning, both actions are recognized due to non-intersecting sets of body parts used for both actions. Figure 4 shows an interactive office environment with an avatar and an autonomous character: the employer. The employer's decision automata is completely coordinated by the recognition feedback of the performer's actions. For example the employer insists energetically on the fact of a non-smoking area if the performer wants to lit his cigarette. If the performer throws away his cigarette the employer invites him to take the contract files. The motions of the employer character consist of pre-recorded keyframe sequences, inverse kinematics and a procedural walking and grasping motors.

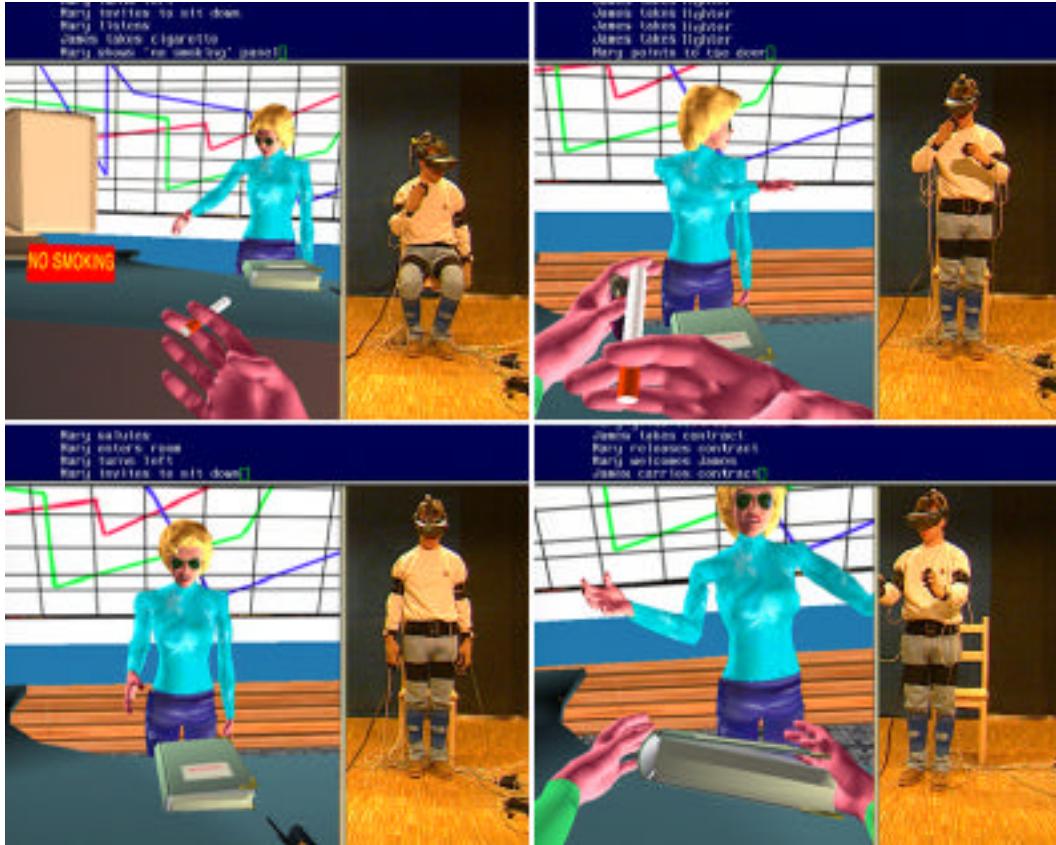


Figure 4. A virtual office environment with the participant (right), his avatar (virtual camera view) and a digital character (cyan). The animation automata is driven by the recognition events of the performers actions.

## Performance

The hierarchical approach overcomes the limitation resulting from the simplicity of the matching tests while allowing real-time performance. A risk of false detection exists whenever two actions have a similar specification. In this case the recognition can oscillate between these actions. We added a simple low-pass filter for the posture recognition part: a posture is declared recognized only after it has been identified successfully for a given number of iterations. Similarly a recognized posture loses its status of recognition result only after it couldn't be identified for several iterations.

For a database of 35 actions we measured an average recognition time of 1.2 ms (R4400 CPU, 200 MHz) with a standard deviation lower than 0.1ms. The frame rate achieved for the present benchmark applications Figure 1, Figure 4) is currently limited by the combined cost of the sensor data processing and the final display; values are around 12 Hz on an SGI Onyx.

The robustness is evaluated on four performers whose total heights range from 1.65m to 1.85m (cf. Figure 5 right). The tested action set consists in four gesture-only actions, twenty-seven posture-only actions and sixteen full-level actions. The test has the following structure: stand still, perform one action, then stand still again. Examples of tested actions are walking (forwards, backwards), sidewalking (left, right), kneeing, different kinds of pointing, sitting down, standing up, nodding ('yes','no').

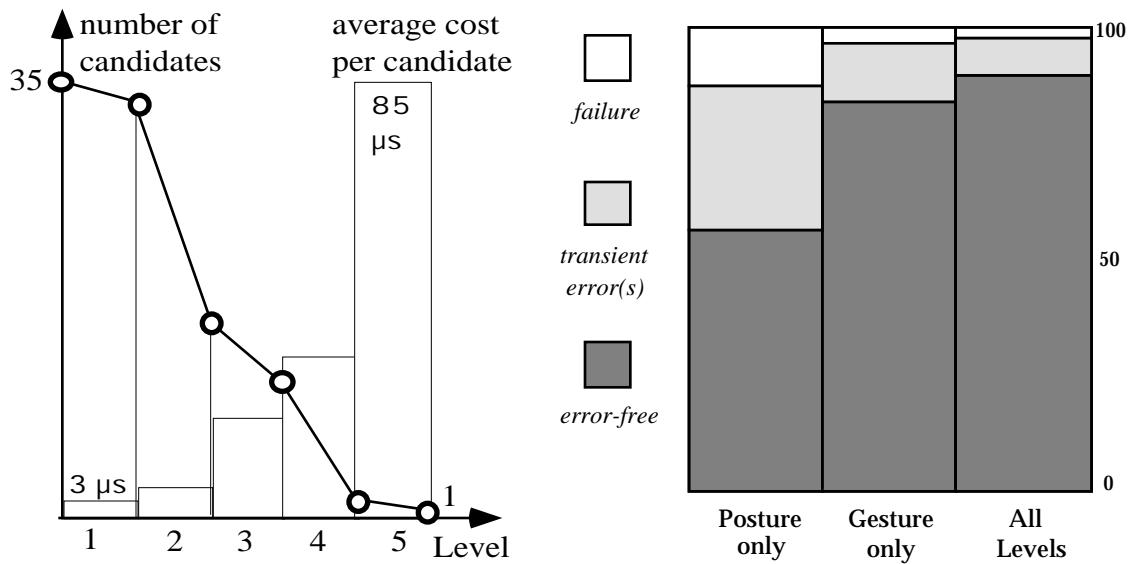


Figure 5. Cost analysis and average action recognition rates (%)

## Discussion

Basically there are three issues worth to be explored for our action model: the integration of a finger model, the concept of constraints and the inclusion of a temporal dimension. By analogy to the body action model, it is interesting to decompose hands into three information levels: the center of the hand, finger tips and finger joints. These levels can be fully integrated into the gesture and posture levels defined in Table 1. The participants' finger movements are captured with datagloves. The second issue is the introduction of the constraint concept. The idea has risen while experimenting with our VR testbed and is best explained with an illustration: gestures and postures allow to detect whether a performer is grasping an object, for example a box. However they can't detect if the performer's hands are still holding the object because the hand/arm configuration does not necessarily stay in a permanent posture. The recognizable invariants of such a situation are the constant distance and orientation between the hands. We define a distance constraint on the hand end effectors which is satisfied as long as the distance equals a constant distance, i.e. the size of the box. Theoretically one can define such constraints between any end effectors. However the high number of possible end effector combinations excludes the computation of constraints for all of them. Therefore only frequently used

combinations can be considered. Finally one can amplify the temporal dimension present at the gesture levels of our action model. Currently a gesture is expressed by single boolean expression of gesture primitives. Only gestures with a single goal can be defined. To specify structured gestures they have to be defined by a sequence of boolean expressions of gesture primitives. This means that the recognition system has to keep a trace of the activation of action primitives, i.e. an activation history. The structure grain of such structured gestures determines the update frequency of the activation history. If the grain is very fine, the gesture primitives reflect instantaneous motion rather than average motions. At the limit, the mechanism of gesture primitives equals a trajectory matching algorithm. As a consequence we can have different classes of action specifications, ranging from average motion considerations to very detailed motions.

Basic concerns of the recognition system are accuracy and evaluation speed. Our interactive VR testbeds have shown that the recognition system satisfies the severe time constraints of real-time applications while assuring a reasonable action identification rate. This has been achieved by compensating the relatively simple matching algorithms with a hierarchical model where action candidates are discarded in the earliest possible stages. With increasing computer power, the matching algorithms can be enhanced to raise accuracy. Other parameters influencing the action recognition system are related to the underlying skeleton model of the avatar and the motion capture system. As the recognition system extracts all the posture data from the avatar, the fidelity of the performer's motion imitation is important. However a realistic virtual skeleton model has a lot more degrees of freedom (more than 70) than could be measured with the current sensor technology (20-40). Moreover the sensors cannot measure precisely the performers' skeleton configuration because they are attached at the skin level. This lack of quality and quantity of raw data is compensated by a powerful anatomical converter but its output cannot be more than a high quality approximation. Finally a drawback is the long setup time for the sensors and the head-mounted display as well as their wires hindering the interaction. However wireless magnetic motion capture systems are already available, dethroning optical systems.

We presented a new action model along with a real-time recognition system and illustrated it with a virtual environment. As technology improves, virtual reality interfaces based on body actions will get more and more important. In the domain of games, an evident application is the control of the hero by body actions. The robotics domain is another area where such an interface presents an attractive issue, especially because telepresence is a hot topic nowadays.

## Acknowledgments

The authors thank S. Rezzonico for the sensor interface, T. Molet for his converter, Selim Balcisoy for previous work, J. Shen and E. Chauvineau for the body deformations, Olivier Aune for the scene design and Mireille Clavien for pre-recorded motions. The research was partly supported by the Swiss National Foundation for Scientific Research.

## References

- [1] Pattie Maes, Trevor Darrell, Bruce Blumberg and Alex Pentland, The ALIVE system: Full-body interaction with Autonomous Agents, Proceedings of the Computer Animation'95 Conference, Geneva, Switzerland, IEEE-Press, April 1995
- [2] Molet T., Boulle R., Thalmann D., A Real Time Anatomical Converter For Human Motion Capture, Eurographics workshop on Computer Animation and Simulation'96, R. Boulle & G. Hegron (Eds.), pp 79-94, ISBN 3-211-828-850, Springer-Verlag Wien, <http://ligwww.epfl.ch/~molet/EGCAS96.html>
- [3] Luc Emeling, Ronan Boulle, Selim Balcisoy, Daniel Thalmann, Real-Time Interactions with Virtual Agents Driven by Human Action Identification, First ACM Conf. on Autonomous Agents'97, Marina Del Rey, 1997, [http://ligwww.epfl.ch/~emeling/AA97\\_video\\_poster.html](http://ligwww.epfl.ch/~emeling/AA97_video_poster.html)

# A Model of Human Crowd Behavior: Group Inter-Relationship and Collision Detection Analysis<sup>1</sup>

**S. R. Musse and D. Thalmann**

{soriaia,thalmann}@lig.di.epfl.ch

*Computer Graphics Lab. Swiss Federal Institute of Technology  
EPFL, DI-LIG, CH 1015 Lausanne, Switzerland*

## Abstract

This paper presents a model of crowd behavior to simulate the motion of a generic population in a specific environment. The individual parameters are created by a distributed random behavioral model which is determined by few parameters. This paper explores an approach based on the relationship between the autonomous virtual humans of a crowd and the emergent behavior originated from it. We have used some concepts from sociology to represent some specific behaviors and represent the visual output. We applied our model in two applications: a graphic called sociogram that visualizes our population during the simulation, and a simple visit to a museum. In addition, we discuss some aspects about human crowd collision.

## 1. Introduction

There are very few studies on crowd modeling. We may mention the following related papers: C. Reynolds 1 developed a model for simulating a school of fish and a flock of birds using a particle systems method 2; D. Terzopoulos 3 developed a model for behavioral animation of fish groups based on the repertoire of behaviors which are dependent on their perception of the environment; S. Velastin 4 worked on the characterization of crowd behavior in confined areas such as railway-stations and shopping malls using image processing for the measure of the crowd motion; T. Calvert 5 developed the blackboard architecture that allows the animator to work cooperatively with a family of knowledge based tools; E. Bouvier 6 presented a crowd simulation in immersive space management and a new approach of particle systems as a generic model for simulations of dynamic systems 7.

In this paper we present a new approach of crowd behavior considering the relationship between groups of individuals and the emergent behavior originated from it (i.e. the global effect generated by local rules). We treat the individuals as autonomous virtual humans that react in presence of other individuals and change their own parameters accordingly. In addition we describe a multiresolution collision method specific for the crowd modeling.

This paper is structured as follows. In section 2, we present some sociological concepts of crowd modeling which were used in our application. In section 3, we present information concerning the model : individual and group parameters, distributed group behavior and implemented sociological effects. In section 4 we present the scenarios where we have applied our model. In section 5, we describe our collision avoidance methods and we present some results and analysis of this problem in the context of crowd simulation. In section 6 we present the applied methodology. Finally, section 7 draws some conclusions about the model.

---

<sup>1</sup> An expanded version of this text has appeared in S.R. Musse, D. Thalmann, A Model of Human Crowd Behavior, Computer Animation and Simulation '97, Proc. Eurographics workshop, Budapest, Springer Verlag, Wien, 1997, pp.39-51.

## 2. Some Sociological Aspects

An accepted definition of crowd is that of a large group of individuals in the same physical environment, sharing a common goal (e.g. people going to a rock show or a football match). The individuals in a crowd may act in a different way than when they are alone or in a small group 12.

Although sociologists are often interested in crowd effects arising from social conflicts or social problems 11,13, the normal behavior of a crowd can also be studied when no changes are expected.

There are, however, some other group effects relevant to our work which are worth mentioning. Polarization occurs within a crowd when two or more groups adopt divergent attitudes, opinions or behavior and they may argue or fight even if they do not know each other. In some situations the crowd or a group within it may seek an adversary 10. The sharing effect is the result of influences by the acts of others at the individual level. Adding is the name given to the same effect when applied to the group. Domination happens when one or more leaders in a crowd influence the others.

Our goal is to simulate the behavior of a collection of groups of autonomous virtual humans in a crowd. Each group has its general behavior specified by the user, but the individual behaviors are created by a random process through the group behavior. This means that there is a trend shared by all individuals in the same group because they have a pre specified general behavior.

## 3. Our Human Crowd Model

This model presents a simple method for describing the crowd behavior through the group inter-relationships. Virtual actors only react in the presence of others, e.g., they meet another virtual human, evaluate their own emotional parameters with those of the other one and, if they are similar, they may walk together. As it is recognized that the model is limited in describing human relationships, only a set of criteria were defined based on the available literature.

The group parameters are specified by defining the goals (specific positions which each group must reach), number of autonomous virtual humans in the group and the level of dominance from each group. This is followed by the creation of virtual humans based on the groups' behavior information. The individual parameters are: a list of goals and individual interests for these goals (originated from the group goals), an emotional status (randomic number), the level of relationship with the other groups (based on the emotional status of the agents from a same group) and the level of dominance (which follows the group trend). As could be seen, some virtual human's parameters are totally random while others parameters follow the group trend.

All the individual parameters can be changed depending on their relations with the others. Therefore, we have defined:

- i) A a virtual human from the group GA and
- ii) B a virtual human from the group GB.

The agents A and B have the following parameters when they are created in the initial population :

- Agent.List\_of\_goals = Group.List\_of\_goals
- Agent.List\_of\_interests= Group.List\_of\_interests
- Agent.Emotional\_status= random([0;1])
- Agent.Relation\_with\_each\_other\_group is defined by the mean emotional status value of all virtual humans from this group
- Agent.Domination\_value = Group.Domination\_value

Based on these parameters, we created some situations in which the individual parameters can be changed during the simulation:

i) The virtual human A can be changed from group GA to group GB if,

i.1) *Relation (A,GA) < Relation (A,GB): if the relationship between A and GA is smaller than between A and GB.*

i.2) *Relation (A,GB) > 0.90: if A has a high value in relation to the group GB.*

In this case, some parameters of virtual human A change according to :

- A.List\_of\_goals = GB.List\_of\_goals

If A.Domination\_value > all\_agents.Domination\_value from GB, if the domination value of A is greater than all autonomous virtual humans from group GB, then A will be the new leader of group GB and the other virtual humans from this group will have a new tendency to follow it. This represents the changes of group parameters consequent to individual changes.

- A.List\_of\_interests = random(each\_goal)GB, meaning that the list of interests for each goal of A is generated by a random process.

Else

- A.List\_of\_interests = (List\_of\_interests) from\_most\_leader\_ag from GB, that means A is influenced by the leader of group GB and
- A.Domination\_value is decreased.

ii) The emotional status can be changed in the encounter of two autonomous virtual humans in a random process. Only when both have a high value for the domination parameter, one virtual human is chosen in a randomic way to reduce its emotional status. In this case a polarization between two leaders will follow. In any other case, both virtual humans must assume the highest emotional status between them.

iii) The relation of A with group GB is defined by the mean state emotional value of all GB virtual humans which have greater emotion status value than A. This value must be normalized (between 0. and 1.).

$$R(A,GB) = \text{Normalized\_value}_{\substack{i=0 \\ \text{N\_AGENTS} \\ \text{FROM\_GB}}} (\text{state\_emotional})_{\text{ag}_i} > (\text{state\_emotional})_{\text{ag}_A}$$

The sociological effects modeled in the presented rules are:

- i) *grouping of individuals depending on their inter-relationships and the domination effect;*
- ii) *polarization and the sharing effects as the influence of the emotional status and domination parameters; and finally,*
- iii) *adding in the relationship between autonomous virtual humans and groups.*

The group behavior is formed by two behaviors: seek goal, that is the ability of each group to follow the direction of motion specified in its goals, e.g. in the case of a visit to a museum, the agents walk in the sense of its goals (the work-of-arts); and the flocking (ability to walk together), has been considered as a consequence of the group movement based on the specific goals (list\_of\_goals) during a specific time (list\_of\_interests). For example, if the user chooses to have flocking in the crowd, the agents from the same group change of goal just when all the agents from this group have arrived at a same point. Thus, if one group g must go to goal number 2 (from the list of goals), all the agents from the group g must arrive before at goal number 1 (also from the list of goals).

The individual behavior is composed of three simpler behaviors. A walk, a collision avoidance (section 5) and a relationship behavior which occurs when the agents meet each other.

## 4. The Implemented Scenarios

We have applied our model in two situations. The first is a sociogram which means a sociological graphic representing one population, its relationships, and the different levels of domination. Figure 1 shows an example of a sociogram.

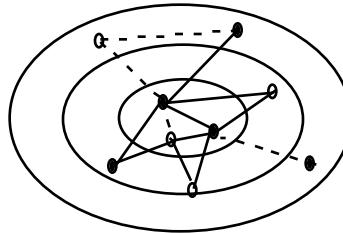


Figure 1: A sociogram

The filled circles represent the women and the empty circles the men. The dashed line represents a relationship of hostility and the full lines represent a friendly relation. In addition, this graphic represents a certain hierarchy in the group, as the individuals in the center of the sociogram have more relationships as compared to the others in the external circles 10.

In our case, we have four groups with different behaviors and parameters (does not include the gender parameter). The autonomous virtual humans walk on the limit of the circles, like in the sociogram and these circles are delimited by points which represent the goals for each group, as we can see in the next figure.

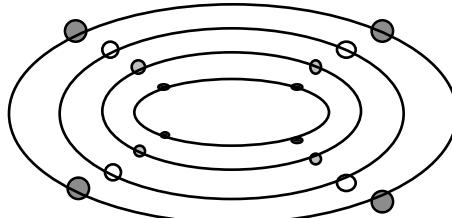


Figure 2: The sociogram in our case

The virtual humans walk in the direction of their goals. Each group has one different color allowing to show when the virtual humans change groups. We can see in Figures 3 and 4 some images of this simulation. Figure 3, we can see the start of the simulation where all autonomous virtual humans are in the initial (randomized assigned) positions. In Figure 4, the virtual humans walk within the limits of their group in the sociogram.

Using the parameters specified in section 3, we have modeled a sociogram including time information which allow us to see changes in crowd behavior during the simulation. In this example, the flocking behavior is represented by walking in a specific region of each group (each circle).

The second example is a crowd which visits a museum. In this simple example, the crowd is formed by different groups. In the beginning of the simulation all virtual humans of the same group visit the museum within the time of their group including, for example, the time to see a specific work of art. However, during the simulation, we can observe agents which change groups and assume the time of those from their new group to visit and walk. In this case, we consider that the virtual humans change groups as a result of both individual and group relationships.

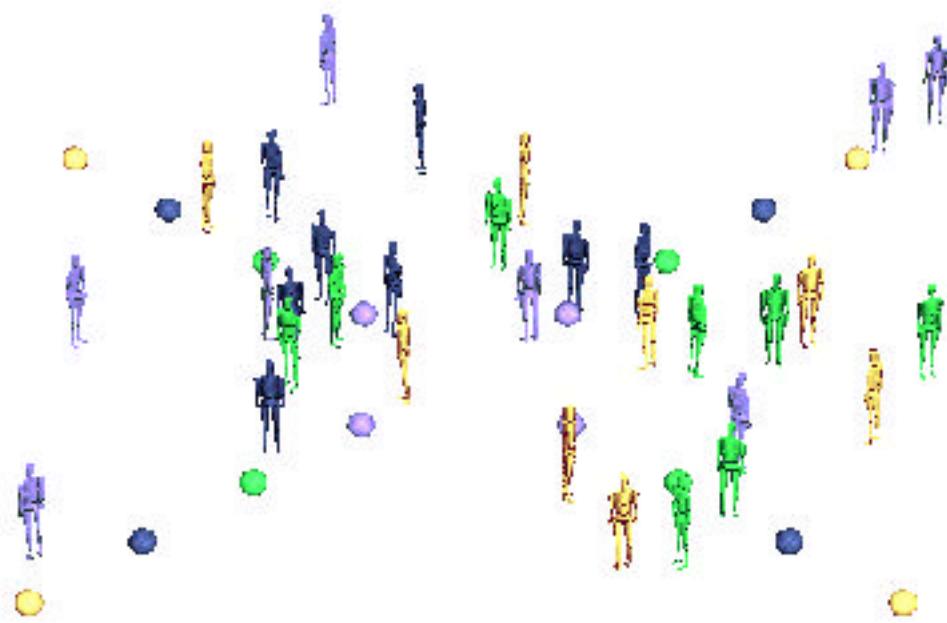


Figure 3: Initial population in sociogram

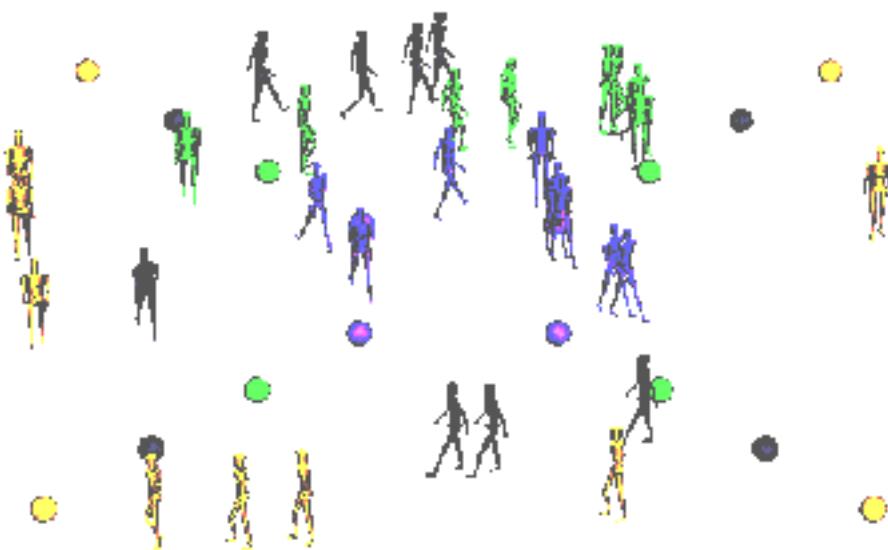


Figure 4: Formed groups in sociogram

Figures 5 and 6 show one example of this simulation. Figure 5 shows the beginning of the simulation, the virtual humans are in their randomized initial position. After one hundred interactions, the agents are gathered in groups and walk in the museum. The autonomous virtual humans of the same group look during a similar time to a specific work of art. Figure 7 shows one image of this simulation integrated in a DIVE environment in COVEN Project 15.

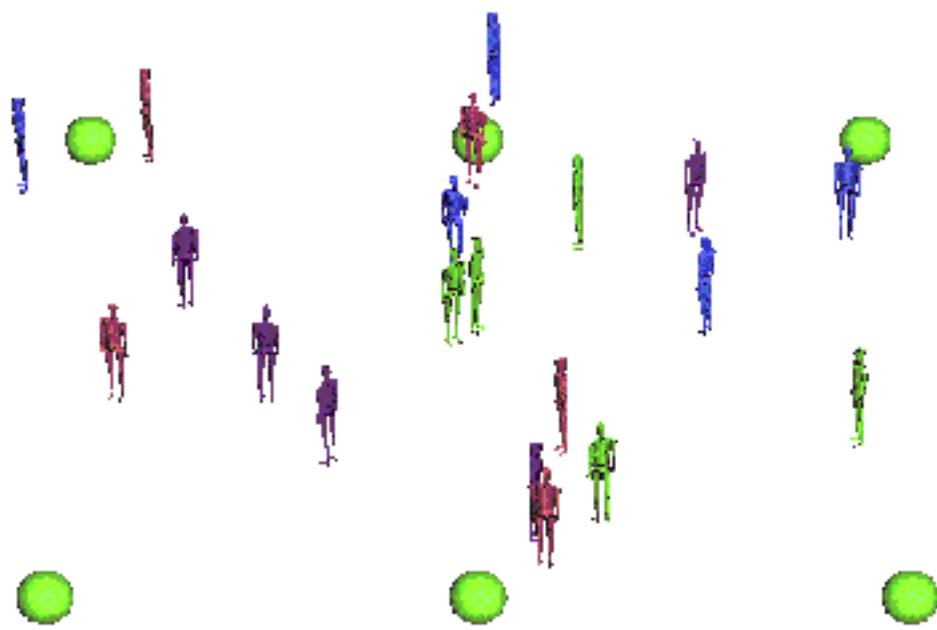


Figure 5: Initial population visiting a Museum.

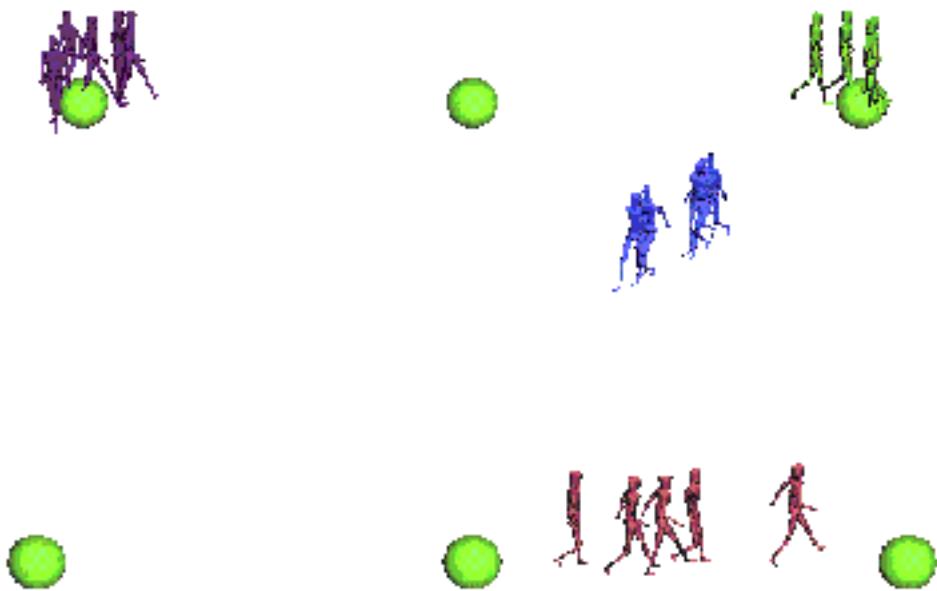


Figure 6: Formed groups in the Museum.

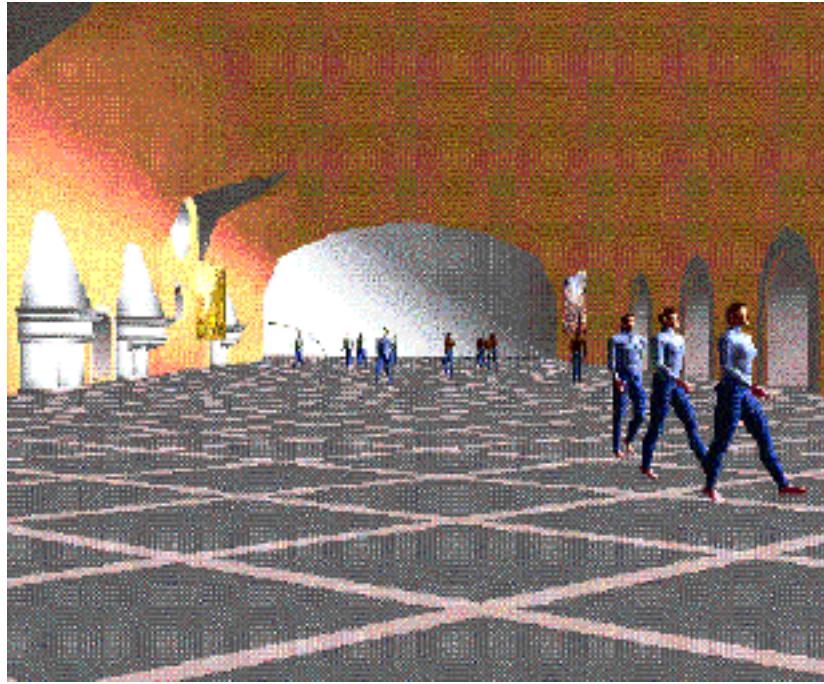


Figure 7: Simulation integrated in DIVE

## 5. Collision Analysis

The main question that we want to answer in this section is the possibility of using the crowd model as an advantage. That is, if we have a crowd model with many autonomous virtual humans, we may decide whether the collision is necessary or not and depending on the cases. Here, we can use some concepts of multiresolution in order to decide which method of collision must be applied. But for this, we need some information to compare different kinds of collision avoidance. Therefore, we implemented 2 types of collision avoidance which we present in this section with a comparison of the results.

### 5.1 Collision Avoidance Type 1

This is a very simple method for avoiding collision. Using simple mathematical equations (intersection of two lines and distance from two points) to determine the future positions from the current ones, thus, we are able to detect a possible collision event. Before the collision occurs, we decide (through some rules of priority) to stop one virtual human and let the other one pass. Unless if the vectors of the virtual humans' movements are collinear, all other situations work just with this condition.

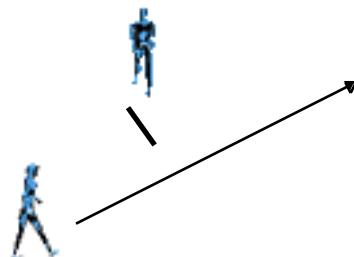


Figure 8: Virtual human stops for the other

In the collinear cases, it is necessary one simple vector analysis to know which agent we must stop or if the agent can have its directions changed (section 5.2).

The following execution flow shows the priority rules:

```

For each pair of agents
  If vectors are not collinear
    If linear velocities are different
      Stop the slower agent
    else
      Choose one agent in a random way
  else
    If the vectors are convergent
      Choose one agent in a random way
      Change its movement direction
    else
      Increase the linear velocity of the agent
      which walks in the front of the other

```

In the following images we can see some examples of the collinear cases.

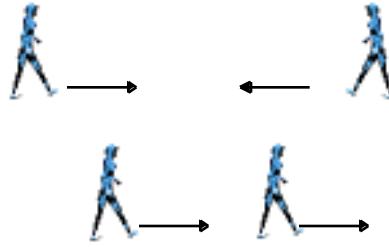


Figure 9: The special collision cases

In the situations of Figure 9, as we can see in the priority rules, it is necessary to decide which virtual human must wait or change the direction. In this latter case, we used the method which will be presented in the next section.

## 5.2 Collision Avoidance Type 2

This other collision method is also very simple, but it is based on the directions changes. Instead of waiting for the other one, the autonomous virtual human can predict the collision event (knowing the position of next virtual humans) through a simple geometric computing (intersection of two lines). Thus, it can avoid the collision by changing its directions through its angular velocity changes. After a specific period of time, the virtual human returns to its last angular velocity (which was stored in the data structures), and also comes back to its previous direction, it represents the goal seeking group behavior.

We assumed:

(ang\_vel)A = func(position\_all\_next\_agents),  
*It means that the new angular velocity from the agent A is a function of the position vectors of the agents in a near distance.*

The next images show a sequence of the collision detection method type 2. We can see the agent avoiding collision by changing his angular velocity.

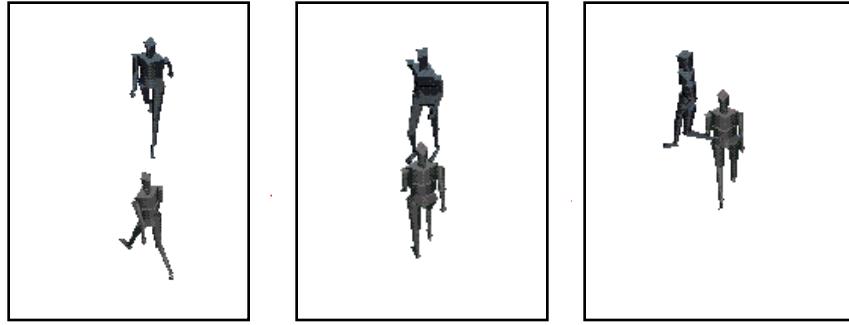
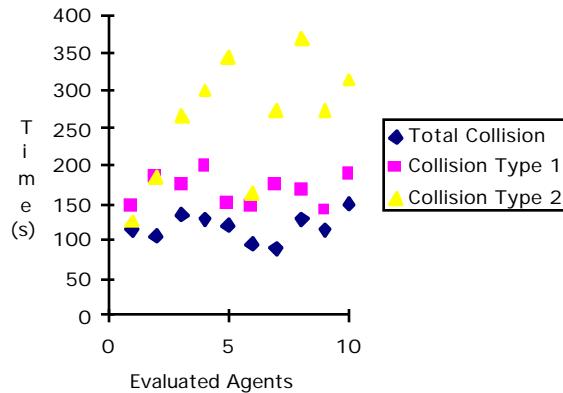


Figure 10: Avoid collision sequence

Sometimes, however, the virtual human does not have enough time to modify its direction because the collision position is too close to the current position. In this case, we are using the last method (section 5.1) as a solution, that is, we decide which virtual human must wait (priority rules presented in section 5.1).

We do not intend to present in this work a new collision algorithm but just to present some useful considerations for the crowd case. Thus, we evaluated these two collision methods and a total collision (i.e., no collision avoidance). The following graphic presents the time data for ten autonomous virtual humans which start at an initial aleatory position and must arrive to a specific final position.



Graphic 1: Comparison of the collision methods for 10 agents

The y axis is the time spent by each virtual human to arrive to their goal. The x axis shows each evaluated agent. As their initial positions are totally random, we can explain the time spent by agent 1 (its initial position was close from its goal position, by coincidence). As the initial population and the behaviors are the same in all three cases, we can compare their time data. The next table presents some results.

Situation	Mean Time (sec)
Total Collision	117.6
Collision Avoidance Type 1	166.9
Collision Avoidance Type 2	261.8

Table 1

As it can be seen in Table 1, in terms of computational time, collision type 2 is 55% more expensive than total collision and 36% more expensive than the collision type 1.

Based on this analysis we specified our multiresolution model. We defined the position limits of the autonomous virtual humans and the collision method implemented. The next figure shows one possible model: consider the distance from the observer to the virtual humans as a high value and the three agents represent the positions limits of the different specific collision methods.

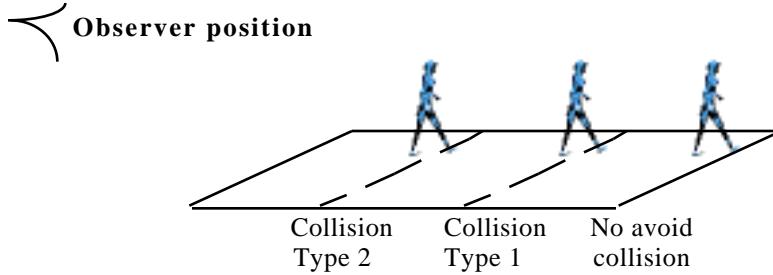
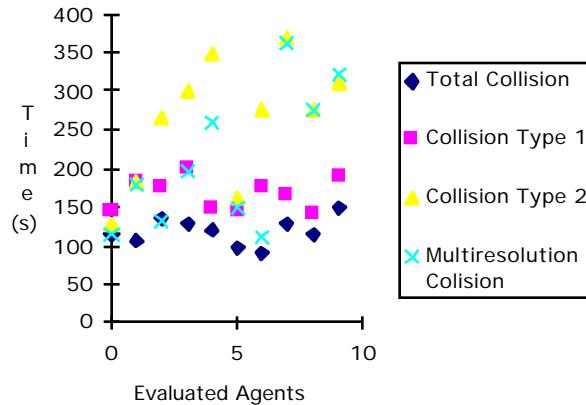


Figure 11: The model of multiresolution collision

In this case, the virtual humans that do not avoid collision must be a long distance from the viewer. We need this kind of multiresolution because we believe that when we have thousands of autonomous virtual humans, a detailed collision response may not be seen and thus become unnecessary. We must emphasize here that the dynamic movement of these three groups will be different because in case of changing trajectory or just waiting for another agent, the time and the path followed by each agent can change. In spite of this, the agents will seek their goal. We have measured the time for multiresolution collision as showed in Graphic 2 and Table 2.



Graphic 2: Comparison with the multiresolution collision

Situation	Mean Time (sec)
Total Collision	117.6
Collision Avoidance Type 1	166.9
Collision Avoidance Type 2	261.8
Multiresolution Collision	209.7

Table 2

For the same population of graphic 1, the time spent avoiding collision is 20% smaller than the collision type 2 in the multiresolution method. We concluded that this is a good method but it will be used when we have many autonomous virtual humans, thus we can not see the collision problems originated from the totally-not-avoiding collision method even if the agents were far away.

## 6. Applied Methodology

Our system was developed in C Language and we have used the libraries SCENELib and BODYlib 8 dedicated to the creation of 3D objects scenes and human actors. We have also used the AGENTlib and BEHAVIORlib 9, the first one is responsible for the coordination of perception and action, and the second enables the definition of a behavior as a hierarchical decomposition in simpler behaviors performed either sequentially or concurrently. We do not use a synthetic vision approach 14 because this feature spends a lot of computational time, thus, we have used here a simple model of perception that just needs the position and orientation vectors of the autonomous virtual humans. In the next graphic (figure 12), we show the system architecture.

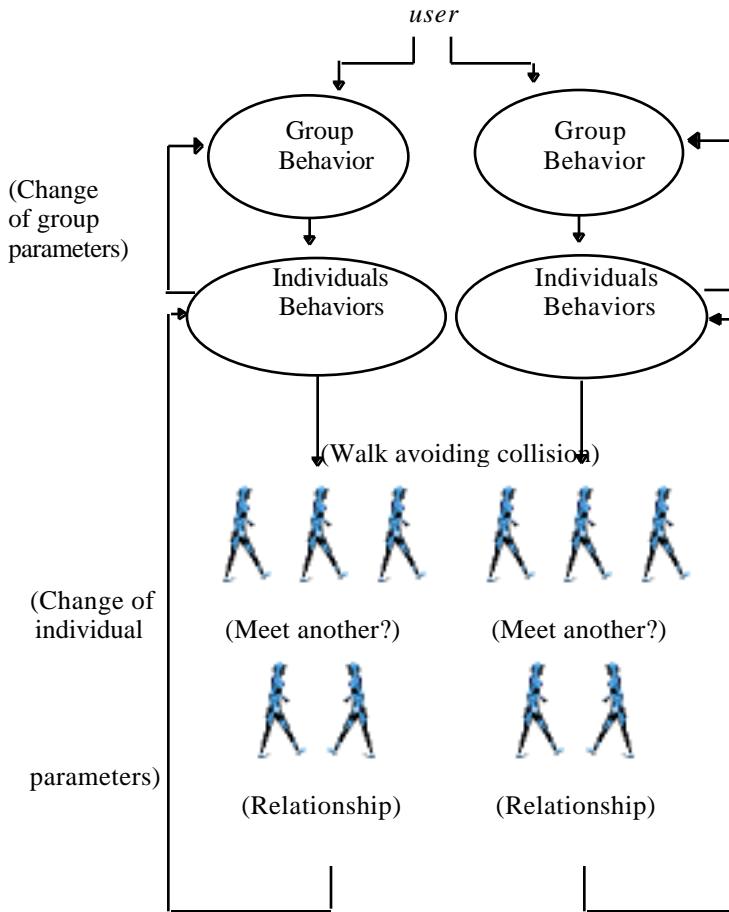


Figure 12: The system architecture

Figure 12 presented the execution flow of our system. As we could see in the last sections, the user specifies the group behavior, which is distributed to all the virtual humans creating the individual parameters. Through these parameters, each agent acts according to three possible behaviors: walk, collision avoidance and relationship with the other, for each time that one virtual human meets an other. This relationship can change the individual parameters and the group behavior, e.g., when one virtual human changes a group and becomes the leader within this new group, the other autonomous virtual humans will try to follow him. In addition, the groups seek the goals and maintain the flocking movement.

## 7. Conclusions

We have described a model for the creation of a behavior based on inter-groups relationships. We think that our results show that we can create a crowd through the group's behavior and drive it through few parameters. Our parameters are still very simple given that we intend to simulate a generic

crowd. However, we believe that our model will allow the creation of a totally generic crowd model in the future using mainly the inter-relationships between the autonomous virtual humans and the goals schema (interest points).

As a direct application, we can imagine the museum example for a real simulation where the goal is finding the best locations for the most visited works of art. Or in the sociogram example, we can have a statistical measure of the population behavior.

In the future we intend to model a panic situation in which all virtual humans will have their goals changed to an unpredictable direction. In this case, it is necessary to have a good model of adaptation of the autonomous virtual humans in a unknown environment.

## 8. Acknowledgments

The authors are grateful to Dr. Ronan Boulic and Pascal Bécheiraz for fruitful discussions and to Dr. Maristela Monteiro for sharing her experience in the sociological and statistical aspects. The research was sponsored by CAPES - Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (Brazilian office of Education and Science), the Swiss National Research Foundation and the Federal Office of Education and Science in the framework of the European project ACTS COVEN.

## 9. References

1. C. Reynolds. "Flocks, Herds, and Schools: A Distributed Behavioral Model". Proc. SIGGRAPH'87, Computer Graphics, v.21, n.4. July, 1987.
2. W. Reeves. "Particle Systems - A Technique for Modeling a Class of Fuzzy Objects". Proc. SIGGRAPH'83, Computer Graphics, v.2, n.2. July, 1983.
3. X. Tu and D. Terzopoulos. "Artificial Fishes: Physics, Locomotion, Perception, Behavior". Proc. SIGGRAPH'94, Computer Graphics, July, 1994.
4. J.H. Yin, S. Velastin and A. C. Davies. "Measurement of Crowd Density using Image Processing". VII European Signal Processing Conference, EUSIPCO-94, Edinburgh, UK, Sept. 1994.,
5. T. Calvert and et. al . "Towards the Autonomous of Multiple Human Figures". IEEE Computer Graphics & Applications, v. pp. 69-75, 1994.
6. E. Bouvier and Pascal Guilloteau.. "Crowd simulation in immersive space management". 3Rd EUROGRAPHICS Workshop on Virtual Environments, Monte Carlo (1996).
7. E. Bouvier, E. Cohen and L. Najman. "From crowd simulation to airbag deployment: particle systems, a new paradigm of simulation". Journal of Electronic Imaging 6(1), 94-107 (January 1997).
8. R. Boulic, T. Capin, Z. Huang, P. Kalra, B. Lintermann, N. Magnenat-Thalmann, L. Moccozet, T. Molet, I. Pandzic, K. Saar, A. Schmitt, J. Shen and D. Thalmann. "The HUMANOID Environment for interactive Animation of Multiple Deformable Human Characters". Proceedings of EUROGRAPHICS'95, p.337-348 (Maastricht, The Netherlands, August 28 september, 1995).
9. P. Bécheiraz and D. Thalmann. "A Model of Nonverbal Communication and Interpersonal Relationship between Virtual Actors". Proc. of the Computer Animation'96, Geneva, Switzerland, pp. 58-67. May, 1996.
10. Hellmuth Benesh. "Atlas de la Psychologie". Librairie Générale Française, 1995, pour l'édition française.
11. J. S. McClelland. "The Crowd and The Mob". Book printed in Great Britain at the University Press, Cambridge. 1989.
12. M. E. Roloff. "Interpersonal Communication - The Social Exchange Approach". SAGE Publications, v.6, London. 1981.
13. P. Mannoni. "La Psychologie Collective". Presses Universitaires de France, Paris. 1985.
14. O. Renault, N.M.Thalmann, D. Thalmann. "A Vision Based Approach to Behavioral Animation". The Journal of Visualization and Computer Animation, v.1, n.1, pp.18-21.
15. T. K. Capin, I. S. Pandzic, H. Noser, N. Thalmann, D. Thalmann. "Virtual Human Representation and Communication in VLNET Networked Virtual Environments". IEEE Computer Graphics and Applications, Special Issue on Multimedia Highways, March 1997.

## Synthetic Characters: Behaving in Character



Bruce Blumberg  
Synthetic Characters  
Group  
MIT Media Lab  
[bruce@media.mit.edu](mailto;bruce@media.mit.edu)  
[www.media.mit.edu/~bruce](http://www.media.mit.edu/~bruce)

MIT Media Laboratory

## Vision for the Synthetic Characters Group

- Great interactive animated characters based on deep models of
  - behavior
  - personality
  - movement
- revealed through compelling interaction and evocative camera and lighting control

MIT Media Laboratory

## **It's hard to build great interactive characters that stay in character...**

- Need to get **it** right
  - Kids “know” how these characters move and think.
- Rich personality, e.g. what makes Daffy, Daffy
  - goals and motivations
  - way of responding to events
  - way of moving and expressing “thoughts”

MIT Media Laboratory

## **Technical hurdles to building characters that stay in character**

- Modeling personality, drives and emotions
- Making sense of the world, given too few bits
- Incorporating learning and memory
- Modeling expressive, re-usable movement
- Building a control system which does the right thing given all of the above

MIT Media Laboratory

## **Approach: combine ideas from multiple disciplines**

- Animal behavior
- Character animation
- Artificial Intelligence
- Artificial Life
- Computer graphics and modeling
- ...

MIT Media Laboratory

## **What contributes to truly great interactive characters?**

**Motion, Behavior, Emotion**

MIT Media Laboratory

## **Motion, Behavior and Emotion**

- Expressiveness of “Disney” characters;
- Control system which insures character always stays “in character” despite dynamic and unpredictable world
  - Motion
  - Behavior
  - Emotion

MIT Media Laboratory

## **What contributes to truly great interactive characters?**

**Motion, Behavior, Emotion**

**Lights,  
Camera &  
Sound**

MIT Media Laboratory

## Lights, camera, & staging

- Cinematography provides well understood tools for
  - directing viewer's attention
  - revealing motivational and emotional state of the character
- What is not well understood is how to do this in a real-time, non-scripted, dynamic environment

MIT Media Laboratory

## What contributes to truly great interactive characters?

Motion, Behavior, Emotion

Interaction

Lights,  
Camera &  
Sound

MIT Media Laboratory

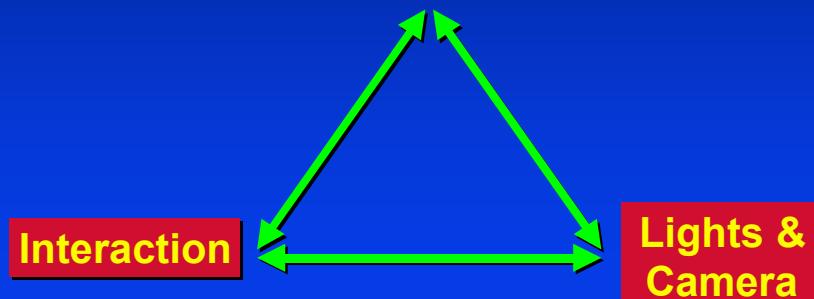
## Interaction

- How do we make the interface disappear so that the user interacts with the character not the computer?
- Challenges
  - Increasing the “semantic bandwidth” between the person and the character?
  - Interacting with characters who have a mind of their own?

MIT Media Laboratory

## The interaction of the parts represents a non-trivial challenge

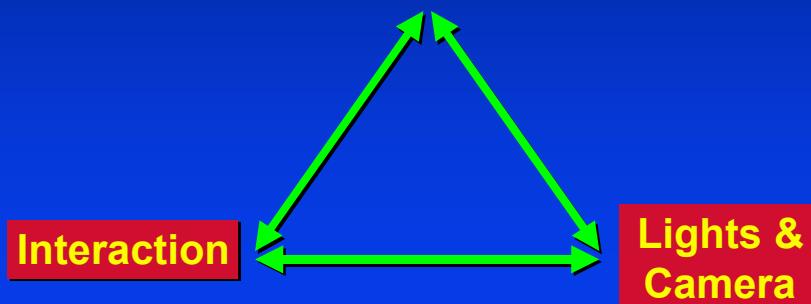
Motion, Behavior, Emotion



MIT Media Laboratory

**Focus of this talk will be on behavior, emotion and learning**

**Motion, Behavior, Emotion**



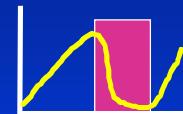
MIT Media Laboratory

**Character decides what to do based on story, state of world & drives**

**Perception**



**Drives & Emotions**



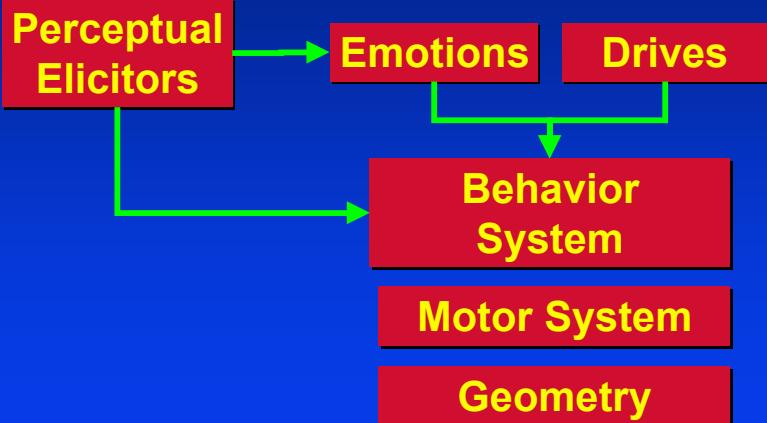
**Behaviors**

**Actions**

**Behaviors are responsible for deciding what to do**

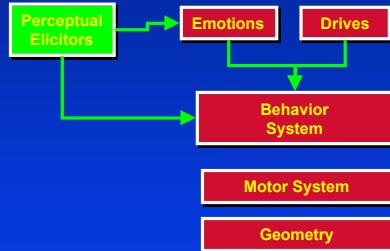
MIT Media Laboratory

## Overview of architecture



MIT Media Laboratory

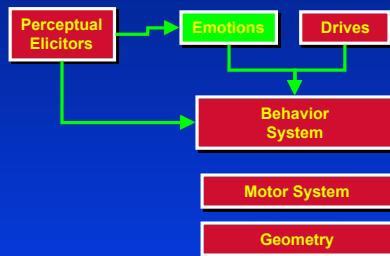
## Elicitors elicit relevant behaviors and emotions based on sensory input



- A “Snake” detector will increase Fear and perhaps elicit the RUN-AWAY behavior

MIT Media Laboratory

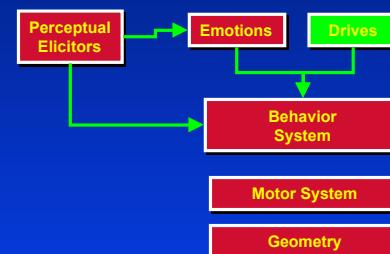
## Emotions bias what behaviors are performed & quality of motion



- High level of ANGER may result in the BITE behavior becoming active and actions being performed “angrily”

MIT Media Laboratory

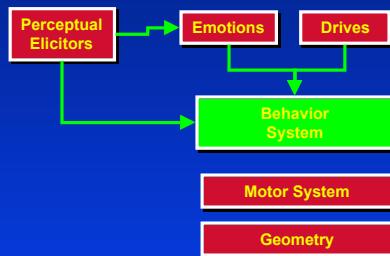
## Drives bias what behaviors are performed



- A high level of HUNGER will predispose the character to engage in behaviors which reduce the level of hunger

MIT Media Laboratory

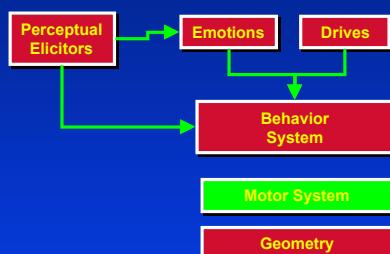
## Behaviors attempt to satisfy drives and emotions given state of world



- They are organized into groups of competing behaviors. These behaviors represent different strategies for satisfying a given drive.

MIT Media Laboratory

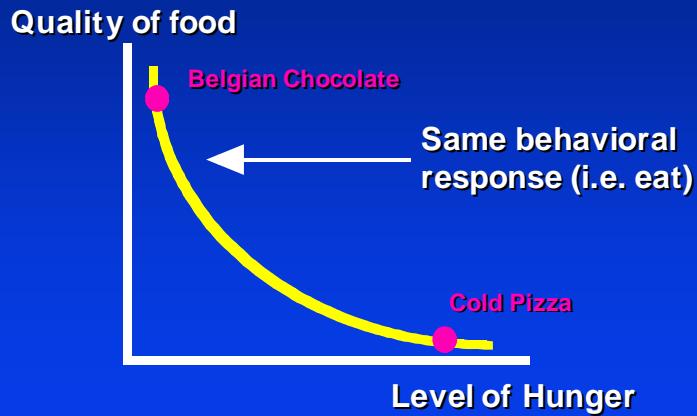
## The Behavior System depends on the Motor System to move the character



- A MOVE-TO behavior, if active, will invoke the “WALK” motor skill and pass it an emotional modifier (e.g. “Angrily”)

MIT Media Laboratory

## Perceptions & motivations needed for spontaneous, natural behavior



MIT Media Laboratory

## We use a value-based approach

- Perceptual elicitors, drives and emotions ultimately output values which represent their relative importance.
- Each Behavior computes a value based on the value of its relevant elicitors, drives & emotions
- Behaviors compete on the basis of their respective values

MIT Media Laboratory

## **Advantages of a value-based approach**

- Natural way to think about drives, emotions and sensory input
- Useful parallels with reinforcement learning, neural-nets (i.e. easy to see how learning can be integrated into the model.)
- Simple to implement, fast at runtime

MIT Media Laboratory

## **System is composed for 4 key types of objects**

- Sensors
- Transducers
- Accumulators
- Groups

MIT Media Laboratory

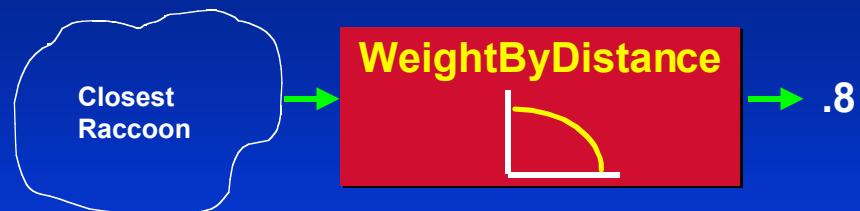
## Sensors act as filters on sensory input



- Sensors filter on features
- Sensors may be chained together

MIT Media Laboratory

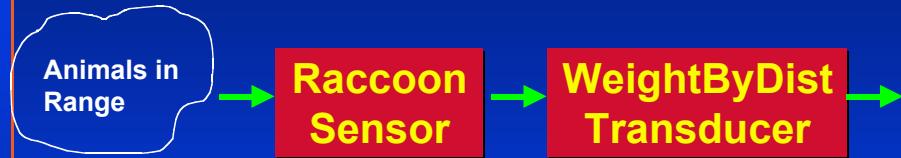
## Transducers transduce a value from an object



- Transducers are needed to map sensory input into values.
- Input to a transducer is typically a sensor

MIT Media Laboratory

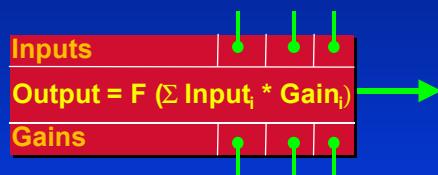
## Perceptual Elicitor represented via chains of Sensors & Transducers



- Sensors filter on features
- Transducer transduce a value

MIT Media Laboratory

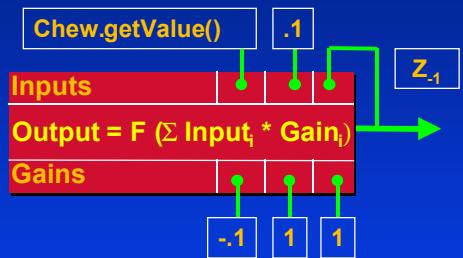
## Accumulators calculate a value based on its inputs and gains



- Inputs & gains are anything that provides a value
- Func() can be any arbitrary function
- Building block for behaviors, drives & emotions

MIT Media Laboratory

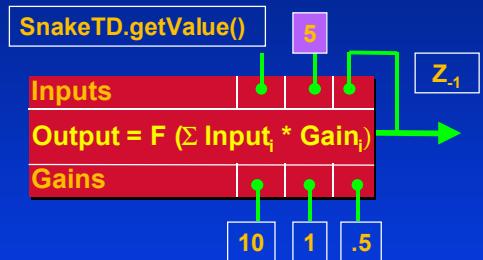
## Drives expressed via an accumulator



- $V_{(t)} = -.1 * \text{Chew.getValue()} + 1 * .1 + 1 * V_{(t-1)}$

MIT Media Laboratory

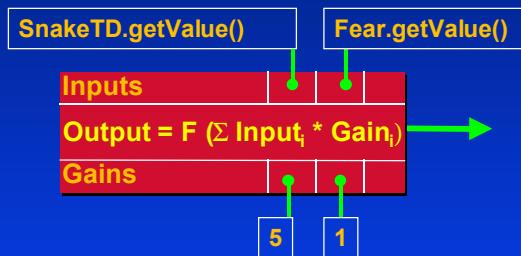
## Emotions expressed via an accumulator



- $V_{(t)} = 10 * \text{SnakeTD.getValue()} + 1 * 5 + .5 * V_{(t-1)}$
- Bias models innate tendency to be in mood
- Elicitors may be reactive and/or cognitive

MIT Media Laboratory

## Behaviors expressed via an accumulator



- $V_{(t)} = 10 * \text{SnakeTD.getValue()} + 1 * \text{Fear.getValue()}$
- Behavior may side-effect drives & emotions
- Behavior may invoke actions as a side-effect

MIT Media Laboratory

## Groups act as containers for competing behaviors...

### Feeding Group

**Eat      Roll-over      Shake      Lie-down**

- Behaviors may represent different strategies for satisfying a drive
- Group is responsible for arbitration so that the most appropriate behavior becomes active

MIT Media Laboratory

## Groups may be organized into loose hierarchies

### Feeding Group

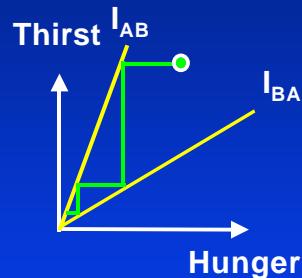
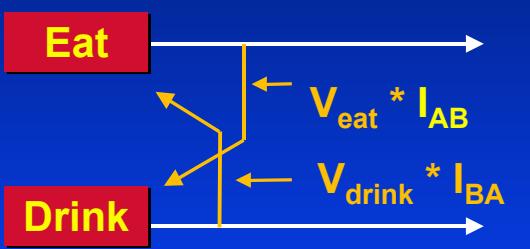
Eat      Find-food

### Find Food Group

Move-to-food      Wander      Avoid

MIT Media Laboratory

## Cross-exclusion groups use mutual inhibition in winner-take-all scheme



Inhibitory gains serve as “knobs” to control persistence

MIT Media Laboratory

## Cross-Exclusion Groups good for arbitrating emotions

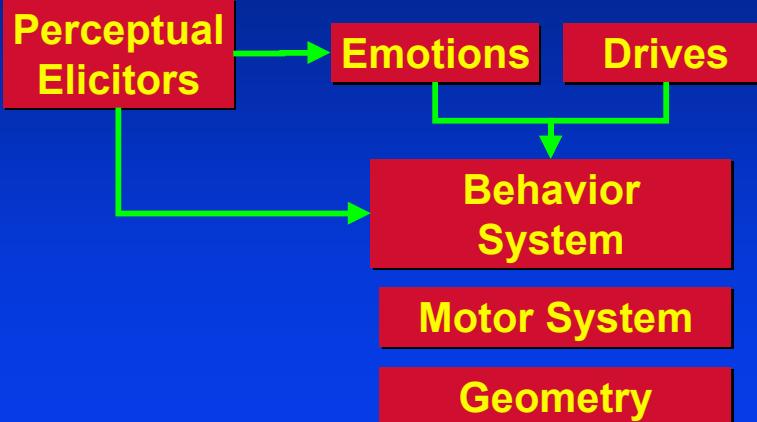
### Emotion Group

Happy Sad Fear Anger Disgust Startle

- Insures only 1 emotion is expressed at a time.
- Temperament modeled through biases & gains
- Character-specific pattern of moods modeled via inhibitory gains

MIT Media Laboratory

## Overview of architecture



MIT Media Laboratory

## **Benefits of this approach...**

- Robust reactive behavior.
- Adaptive and personalizable behavior
- Behavior that doesn't feel scripted
- Control at higher levels of abstraction
- Interactive characters which behave in character...

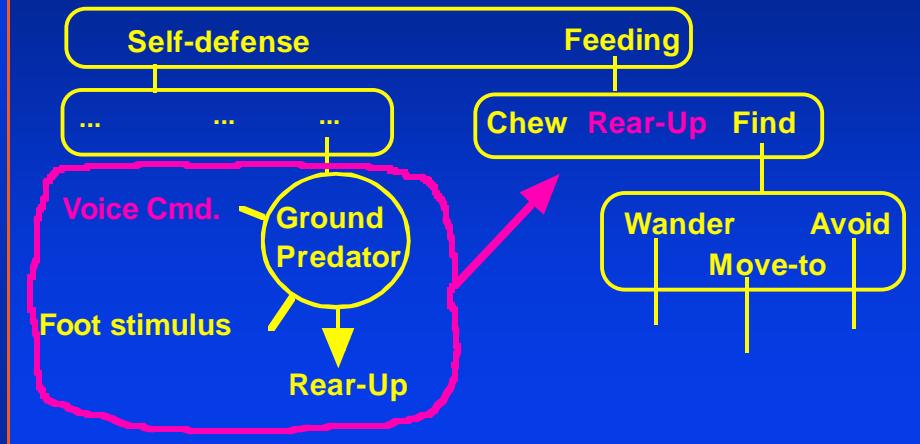
MIT Media Laboratory

## **The possible integration of learning is an important aspect of this approach**



MIT Media Laboratory

## Background: Lorenz's perspective on animal training



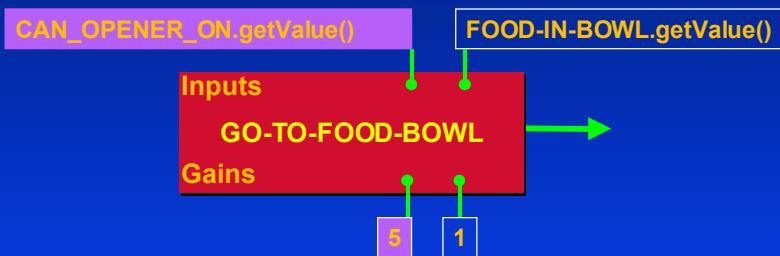
MIT Media Laboratory

## Useful things to learn: New clues to old contexts

Stimulus	Door Squeak →	
Behavior		Food In Bowl →
Outcome		Go to Bowl →
		Reduce Hunger

MIT Media Laboratory

## Learning new elicitors to improve reliability of existing behaviors



- Classical conditioning
- Learn association and weight

MIT Media Laboratory

## Useful things to learn : New ways to satisfy existing drives

Stimulus Hand Extended →

Behavior SIT →  
Outcome Dog Bone

MIT Media Laboratory

## Learn new combinations of elicitors and behaviors

### Feeding Group

Eat

Find-food

Hand-Extended

Sit

1.2

- Operant conditioning
- Learn association between elicitor and behavior
- Learn causal relationship with change in drive

MIT Media Laboratory

## Lurking Problem: Is it an operant or classical contingency?

Stimulus    Door Squeak →

Food In Bowl →

Behavior       Lick Paw →

Go to Bowl →

Outcome

Reduce Hunger

“Reduce Hunger” group

Door  
Squeak

Lick  
Paw

?

Food in Bowl

GoTo

Door Squeak

MIT Media Laboratory

## Approach

- Drives and emotions “drive” learning (i.e. they act as the reinforcement signal)
- Look for reliable conjunctions of behavior and stimulus
- Use temporal difference learning to learn weight for new Elicitors
- Use a simple metric of reliability to distinguish between operant and classical contingencies

MIT Media Laboratory

## Build elicitors which signal interesting conjunctions of behavior and stimulus

- Looks at short-term memory for candidates
- Builds 2 elicitors per conjunction: ( B && S) and ((!B) && S)

Stimulus



Behavior



(B && S) Active

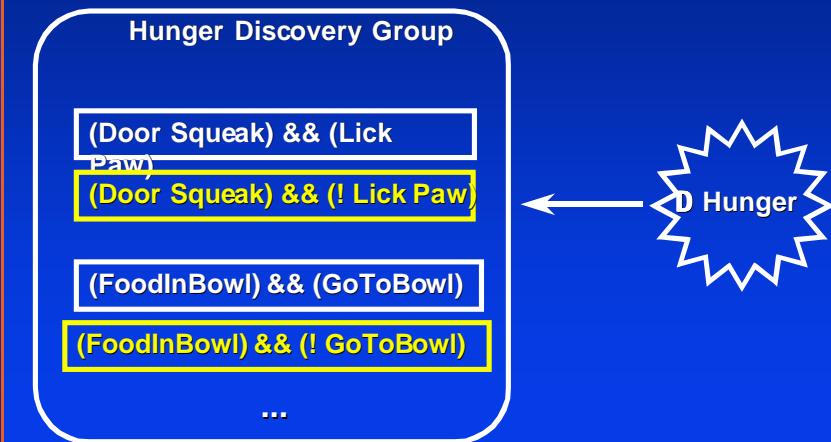


((!B) && S) Active



MIT Media Laboratory

## Candidate elicitors compete to explain D in value of drive



MIT Media Laboratory

## They compete using Sutton & Barto Temporal Difference Learning Model

- **MaxValue for each elicitor learned via the Sutton & Barto Temporal Difference Model:**

$$\Delta \text{maxValue} = \text{LearningRate} * (\text{actual reward} - \text{predicted reward}) * (\text{temporal trace of elicitor})$$

- **Also learns the reliability of each elicitor:**

$$\text{Reliability} = \frac{\# (\text{Active} \&\& \text{Rewarded})}{\# (\text{Active})}$$

MIT Media Laboratory

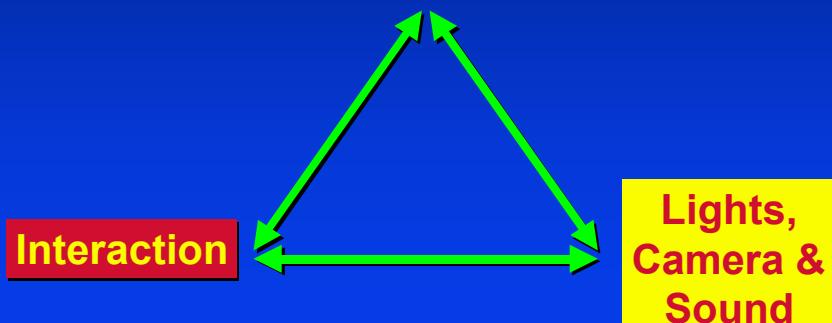
## Decision Rules

- Operant Contingency
  - Learned value of  $(B \And S) > \text{threshold}$
  - Reliability( $B \And S$ )  $>$  Reliability( $(\neg B) \And S$ )
- Classical Contingency
  - Learned value of one  $((\neg B_i) \And S) > \text{threshold}$
  - For all  $B_i$ : Reliability  $((\neg B_i) \And S) > (B_i \And S)$

MIT Media Laboratory

## What makes truly great interactive characters?

Motion, Behavior, Emotion



MIT Media Laboratory

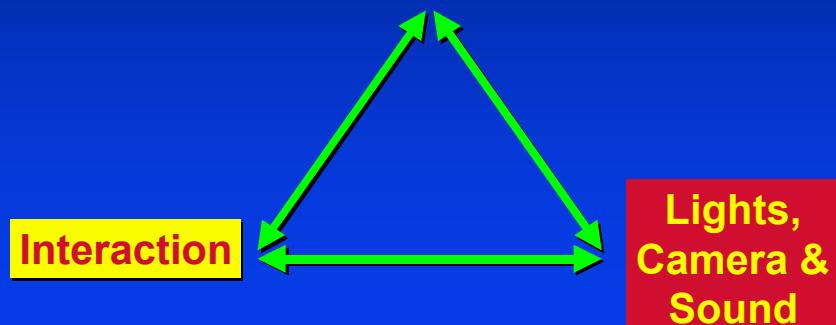
## **Lighting and camera control needs to be character-centered**



MIT Media Laboratory

## **What makes truly great interactive characters?**

**Motion, Behavior, Emotion**



MIT Media Laboratory

## **Interaction should be iconic, tangible and matched with character**

- Should encourage guest to read more, not less, into character...



MIT Media Laboratory

## **Swamped! A plush toy interface for autonomous characters**



**A rich platform from which to investigate**

- Interactive characters
- Automatic camera control
- Iconic, tangible interfaces

MIT Media Laboratory

## What makes truly great interactive characters?

Motion, Behavior, Emotion

Interaction

Lights,  
Camera &  
Sound

MIT Media Laboratory

## Synthetic Characters: Behaving in Character



Come see us in Enhanced Realities!!!

MIT Media Laboratory

## Multi-Level Direction of Autonomous Creatures for Real-Time Virtual Environments

Bruce M. Blumberg and Tinsley A. Galyean

bruce@media.mit.edu, tag@media.mit.edu

MIT Media Lab

E15-311, 20 Ames St.

Cambridge Ma. 02139

### ABSTRACT

There have been several recent efforts to build behavior-based autonomous creatures. While competent autonomous action is highly desirable, there is an important need to integrate autonomy with "directability". In this paper we discuss the problem of building autonomous animated creatures for interactive virtual environments which are also capable of being directed at multiple levels. We present an approach to control which allows an external entity to "direct" an autonomous creature at the motivational level, the task level, and the direct motor level. We also detail a layered architecture and a general behavioral model for perception and action-selection which incorporates explicit support for multi-level direction. These ideas have been implemented and used to develop several autonomous animated creatures.

### 1. INTRODUCTION

Since Reynold's seminal paper in 1987, there have been a number of impressive papers on the use of behavioral models to generate computer animation. The motivation behind this work is that as the complexity of the creature's interactions with its environment and other creatures increases, there is an need to "endow" the creatures with the ability to perform autonomous activity. Such creatures are, in effect, autonomous agents with their own perceptual, behavioral, and motor systems. Typically, authors have focused on behavioral models for a specific kind of creature in a given environment, and implemented a limited set of behaviors. There are examples of locomotion [2, 5, 7, 14, 16], flocking [18], grasping [9], and lifting [2]. Tu and Terzopoulos's Fish [20] represent one of the most impressive examples of this approach.

Advances in behavioral animation are critically important to the development of creatures for use in interactive virtual environments. Research in autonomous robots [4, 8, 12] supports the need to couple real-time action with dynamic and unpredictable environments. Their insights only serve to strengthen the argument for autonomous animated creatures.

Pure autonomy, perhaps, should not be the ultimate goal. Imagine making an interactive virtual "Lassie" experience for children. Suppose the autonomous animated character playing Lassie did a fine job as a autonomous dog, but for whatever reason was ignoring the child. Or suppose, you wanted the child to focus on some

---

Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

©1995 ACM-0-89791-701-4/95/008

aspect of the environment which was important to the story, but Lassie was distracting her. In both cases, you would want to be able to provide external control, in real-time, to the autonomous Lassie. For example, by increasing its "motivation to play", it would be more likely to engage in play. Alternatively, Lassie might be told to "go over to that tree and lie down" so as to be less distracting.

Thus, there is a need to understand how to build animated characters for interactive virtual environments which are not only capable of competent autonomous action but also capable of responding to external control. We call this quality "directability." This is the fundamental problem addressed in this paper.

This paper makes 3 primary contributions to the body of literature regarding animated autonomous characters. Specifically, we describe:

- An approach to control which allows an external entity to "direct" a virtual character at a number of different levels.
- A general behavioral model for perception and action-selection in autonomous animated creatures but which also supports external control.
- A layered architecture which supports extensibility, re-usability and multiple levels of direction.

An experimental toolkit which incorporates these ideas has been successfully used to build a number of creatures: a virtual dog used in an interactive virtual environment, and several creatures used in an interactive story telling environment.

The remainder of the paper is organized as follows. In section 2 we present a more detailed problem statement and summarize the key contributions of our approach. In section 3 we present an overview of the general architecture. In sections 4, 5 and 6 we discuss the motor, sensory and behavior systems in more depth. In section 7 we discuss how directability is integrated into our architecture. Finally, in section 8 we discuss some aspects of our implementation and give examples of its use.

### 2. PROBLEM STATEMENT

An autonomous agent is a software system with a set of goals which it tries to satisfy in a complex and dynamic environment. It is autonomous in the sense that it has mechanisms for sensing and interacting with its environment, and for deciding what actions to take so as to best achieve its goals[12]. In the case of an autonomous animated creature, these mechanisms correspond to a set of sensors, a motor system and associated geometry, and lastly a behavior system. In our terminology, a creature is an animate object capable of goal-directed and time-varying behavior.

Deciding on the "right" action or set of actions is complicated by a number of factors. For example, due to the problems inherent in sensing and perception, a creature's perception of its world is likely to be incomplete at best, and completely erroneous at worst.

There may be competing goals which work at cross-purposes (e.g. moving toward food may move the creature away from water). This can lead to dithering in which the creature oscillates among competing activities. On the other hand, an important goal may be un-obtainable, and pursuit of that goal may prevent the satisfaction of lower priority, but attainable goals. External opportunities need to be weighed against internal needs in order to provide just the right level of opportunistic behavior. Actions may be unavailable or unreliable. To successfully produce competent autonomous action over extended periods of time, the Behavior System must provide solutions to these problems, as well as others.

However, as mentioned earlier, strict autonomy is not the goal. We need, in addition, to direct the creature at a number of different levels. Three levels of input, (motivational, task, and direct) are outlined in Figure 1. Additionally, commands at the direct level need to be able to take three imperative forms:

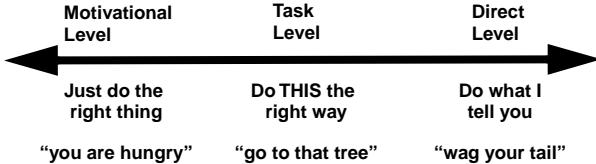
- Do it, independent of the Behavior System.
- Do it, if the Behavior System doesn't object.
- Suggest how an action should be performed, should the Behavior System wish to perform that action.

Thus, the behavior and motor systems must be designed and implemented in such a way that it is possible to support these levels and types of direction at run-time.

Building autonomous animated creatures is inherently an iterative process. This is particularly true since we are in the early phases of understanding how to build them. Ideally, a common approach should be taken for the specification of geometry through to behavior so that a developer need only learn a single framework. Lastly, an embedded interpreter is required to facilitate testing, as well as run-time direction.

## 2.1 Multiple Levels of Control

We provide an approach to control which allows an external entity to "direct" an autonomous animated creature at a number of different levels. These levels are detailed in Figure 1. By providing



**Figure 1:** Here we articulate three levels at which a creature can be directed. At the highest level the creature would be influenced by changing its current motivation and relying on it to react to this change. If you tell it to be hungry it will go off looking for food. At the task level you give it a high level directive and you expect it to carry out this command in a reasonable manner (for example walking around a building instead of through it.) At the lowest level you want to give a creature a command that directly changes its geometry.

the ability to "direct" the creature at multiple levels the animator or developer can choose the appropriate level of control for a given situation. Both Badler and Zeltzer have proposed similar decomposition of control [2, 23].

## 2.2 A General Behavior Model

We propose a distributed behavioral model, inspired by work in Ethology and autonomous robot research, for perception and action-selection in autonomous animated creatures but which also supports external control. The contributions of this model include:

- A general model of action-selection which provides greater control over temporal patterns of behavior than previously described approaches have offered.
- A natural and general way to model the effect of external stimuli and internal motivation.
- An approach in which multiple behaviors may suggest actions to be performed and preferences for how the

actions are to be executed, while still maintaining the advantages of a winner-take-all architecture.

- An implementation which supports motivational and task level direction at run-time.

We also describe a robotics inspired approach to low-level autonomous navigation in which creatures rely on a form of synthetic vision to perform navigation and obstacle avoidance.

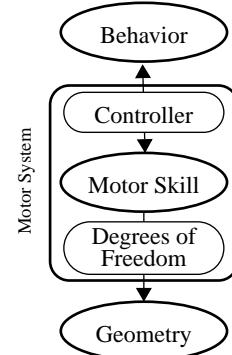
## 2.3 A Layered Architecture

A 5-layered architecture for autonomous animated creatures is described. Several important abstraction barriers are provided by the architecture:

- One between the Behavior System and the Motor Skills, which allows certain behaviors (e.g. "move-toward") to be independent of the Motor Skills which perform the desired action (e.g. "drive" vs. "walk") in a given creature.
- One between the Motor Skills and geometry which serves as both an abstraction barrier and a resource manager.

The result is an architecture which encourages re-usability and extensibility, while providing the necessary foundation to support autonomous action with interactive direction.

## 3. ARCHITECTURE



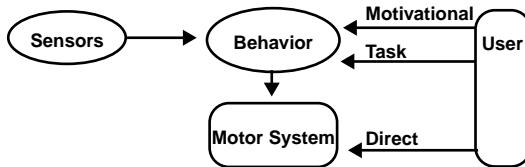
**Figure 2:** Block diagram of a creature's architecture. The basic structure consists of the three basic parts (Geometry, Motor Skills and Behavior) with two layers of abstraction between these parts (Controller, and Degrees of Freedom).

Figure 2 shows the basic architecture for a creature. The geometry provides the shapes and transforms that are manipulated over time for animation. The Motor Skills provide atomic motion elements which manipulate the geometry in order to produce coordinated motion. "Walking" or "Wagging the tail" are examples of Motor Skills. Motor Skills manipulate the geometry with no knowledge of the environment or state of a creature, other than that needed to execute the skill. At the top rests the Behavior System of a creature. This element is responsible for deciding what to do, given its goals and sensory input and triggering the correct Motor Skills to achieve the current task or goal. In addition to these three parts, there are two layers of insulation, the controller and the degrees of freedom (DOFs), which are important to making this architecture generalizable and extensible.

Behaviors implement high level capabilities such as, "find food and eat", or "sit down and shake", as well as low level capabilities such as "move to" or "avoid obstacle" by issuing the appropriate motor commands (i.e "forward", "left", "sit", etc.) to the controller. Some behaviors may be implemented in a creature-independent way. For example, the same "move to" behavior may be applicable to any creature with basic locomotive skills (e.g. forward, left, right,...) although each may use different Motor Skills to perform the required action. It is the controller which provides this common interface to the Motor Skills by mapping a generic command ("forward") into the correct motor skill(s) and parameters for

a given creature. In this way, the same behavior may be used by more than one type of creature.

Figure 3 shows the sources of input to the creature. Sensors are



**Figure 3:** There are two sources of input to a creature. First are sensors associated with the creature. These sensors are used by the Behavior System to enable both task level and autonomous behavior. The other source of input is from the user (or application using the creature.) This input can happen at multiple levels, ranging from simply adjusting a creature's current motivational state to directly turning a motor skill on or off.

elements of a creature which the creature uses to interrogate the environment for relevant information. The creature may also take additional input from the user or the application using the creature. These directives can enter the creature's computational model at the three different levels.

#### 4. MOTOR SYSTEM

We use the term "motor system" to refer to the three layers that lie between the Behavior System and the geometry, Figure 2. These parts include the Motor Skills in the center, and the abstraction and interface barriers on either side of the Motor Skills. Together these three layers of the architecture provide the mapping from motor commands to changes in the geometry over time.

The motor system is designed to meet the following 5 important criteria:

- Act as an abstraction barrier between high-level commands (e.g. "forward") and the creature specific implementation (e.g. "walking").
- Support multiple imperative forms for commands.
- Provide a generic set of commands which all creatures can perform.
- Minimize the amount of "house-keeping" required of the Behavior System.
- Provides resource management so as to support coherent, concurrent motion.

Within these three layers of the motor system, the controller provides the high level abstraction barrier and the support of multiple imperative forms. Motor skills can be inherited allowing basic skills to be shared amongst creatures. Also, Motor Skills are designed to minimize the "house-keeping" that an external user or Behavior System must do. It is the degree of freedom abstraction barrier that serves as the resource manager.

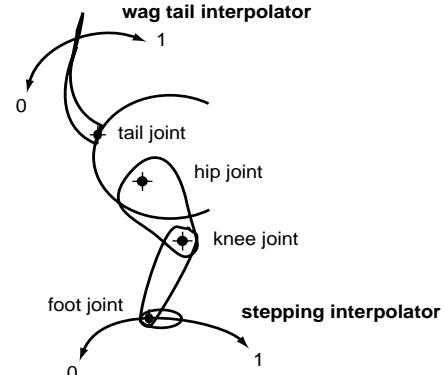
##### 4.1 Degrees of Freedom (DOFs)

Degrees of Freedom (DOFs) are "knobs" that can be used to modify the underlying geometry. They are the mechanism by which creatures are repositioned and reshaped. For example, DOFs might be used to wag the tail, move a joint, or reposition an entire leg. DOFs serve 2 important functions:

- Resource management. Provides a locking mechanism so that competing Motor Skills do not conflict.
- An abstraction barrier. Utilizes interpolators to re-map simple input values (0 to 1) to more complex motion.

The resource management system is a simple one. Each DOF can be locked by a motor skill, restricting it by anyone else until unlocked. This locking provides a mechanism for insuring coherent, concurrent motion. As long as two or more Motor Skills do not conflict for DOFs they are free to run concurrently. Alternatively, if a motor skill requests DOFs that are already locked it will be informed it cannot run currently.

When functioning as an abstraction barrier a DOF provides a mechanism to map a simple input value (often a number between 0 and 1) to another space via interpolators and inverse kinematics such as in, Figure 4. It is this abstraction that allows a motor skill



**Figure 4:** DOFs in a creature can provide interfaces to the geometry at several different levels. For example, joints (and therefore the associated transformations) can be directly controlled or indirectly as is the case with this leg. Here inverse kinematics is used to move the foot. An additional level of abstraction can be added by using interpolators. The interpolator on the leg provides a "one knob" interface for the motor skill. By giving a number between 0 and 1 the motor skill can set the location of the leg along one stepping cycle. Likewise the tail can be wagged with only one number.

to position a leg along a stepping cycle via one number. Note that when a high level DOF (the stepping DOF in this example) is locked the lower level DOFs it utilizes (the leg joints) are in turn locked.

##### 4.2 Motor Skills

A motor skill utilizes one or more DOFs to produce coordinated movement. Walking, turning, or lower head are all examples of Motor Skills. A motor skill can produce complicated motion, and the DOFs' locking mechanism insures that competing Motor Skills are not active at the same time. In addition, Motor Skills present an extremely simple interface to upper layers of the architecture. A motor skill can be requested to turned on, or to turn off. In either case, arguments may be passed as part of the request.

Motor skills rely heavily on degrees of freedom to do their work. Each motor skill declares which DOFs it needs in order to perform its task. It can only become active if all of these DOFs are unlocked. Once active, a motor skill adjusts all the necessary DOFs with each time-step to produce coordinated animation.

Most Motor Skills are "spring-loaded." This means that if they have not been requested to turn on during an update cycle, they begin to move their DOFs back toward some neutral position and turn off within a few time-steps. The advantage of this approach is that a behavior, which turns on a skill, need not be concerned with turning it off at the correct time. The skill will turn itself off, thereby reducing the amount of "bookkeeping." Because Motor Skills are "spring-loaded" the Behavior System is required to specify which skills are to be active with each time-step. This may seem like a burden but it is consistent with a reactive behavior system which re-evaluates what actions it should perform during every update cycle. It should also be noted that this spring-loaded feature can be turned off, to facilitate sources of direction other than the Behavior System.

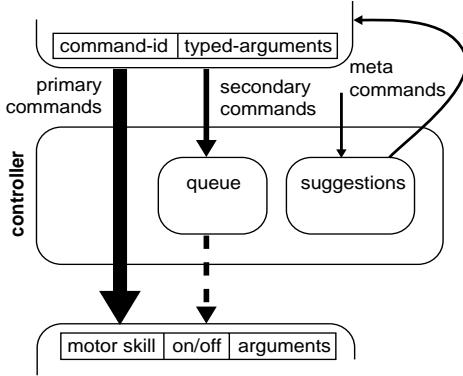
There are a number of basic Motor Skills which all creatures inherit, such as ones for setting the position or heading of the creature.

##### 4.3 Controller

The controller is a simple but significant layer in the architecture which serves an important function as an abstraction barrier

between the Behavior System and the underlying Motor Skills. The primary job of the controller is to map commands such as “forward”, “turn”, “halt”, “look at” etc. into calls to turn on or turn off the appropriate motor skill(s). Thus, “forward” may result in the “walk” motor skill being turned on in the dog but the “move” motor skill in the case of the car. This is an important function because it allows the Behavior System or application to use one set of commands across a potentially wide-range of creatures, and lets the motor system of each creature to interpret them differently but appropriately.

The controller accepts commands in the form of a data structure called a motor command block. This data structure specifies the command and any arguments. In addition, a motor command block can store return arguments, allowing functions that inquire about the state of a creature (its position, its velocity) to be treated by the same mechanism as all other commands. A command block can be issued to the controller as one of three imperative forms: primary command - to be executed immediately; secondary - to be queued at a lower priority; and as a meta command - suggesting how another command should be run, Figure 5. These different impera-



**Figure 5:** An incoming command is represented in a motor command block consisting of a command id and an optional list of typed arguments. If these arguments are not supplied then defaults stored in the controller are used. Any given command will turn on or off one or more Motor Skills, while also providing any necessary arguments. Commands take two levels of importance. Primary commands are executed right away, while secondary commands are queued. These queued commands are executed at the end of each time cycle, and only if the necessary resources (DOFS) are available. It is expected that these secondary commands will be used for suggested but not imperative actions. The last type of input into the controller is in the form of a meta-command. These commands are stored as suggestion of how to execute a command. For example, “if you are going to walk I suggest that you walk slowly.” These are only stored in the controller and it is the responsibility of the calling application (or user) to use or ignore a suggestion.

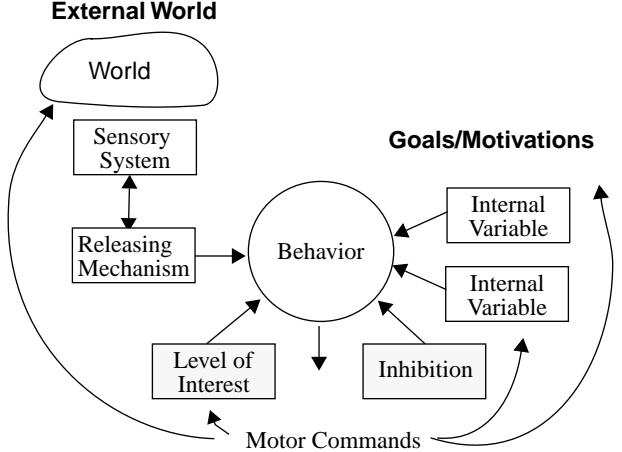
These forms are used extensively by the Behavior System (see section 6.6). They allow multiple behaviors to simultaneously express their preferences for motor actions.

## 5. SENSING

There are at least three types of sensing available to autonomous animated creatures:

- Real-world sensing using real-world “noisy” sensors.
- “Direct” sensing via direct interrogation of other virtual creatures and objects.
- “Synthetic Vision” in which the creature utilizes vision techniques to extract useful information from an image rendered from their viewpoint.

While it is important to support all three types of sensing, we have found synthetic vision to be particularly useful for low-level navigation and obstacle avoidance. Several researchers, including Renault [17], Reynolds[18], and Latombe[10] have suggested similar approaches.



**Figure 6:** The purpose of a Behavior is to evaluate the appropriateness of the behavior, given external stimulus and internal motivations, and if appropriate issue motor commands. Releasing Mechanisms act as filters or detectors which identify significant objects or events from sensory input, and which output a value which corresponds to the strength of the sensory input. Motivations or goals are represented via Internal Variables which output values which represents the strength of the motivation. A Behavior combines the values of the Releasing Mechanisms and Internal Variables on which it depends and that represents the value of the Behavior before Level of Interest and Inhibition from other Behaviors. Level of Interest is used to model boredom or behavior-specific fatigue. Behaviors must compete with other behaviors for control of the creature, and do so using Inhibition (see text for details). There are a variety of explicit and implicit feedback mechanisms.

## 5.1 Synthetic Vision For Navigation

Horswill [8] points out that while “vision” in general is a very hard problem, there are many tasks for which it is possible to use what he calls “light-weight” vision. That is, by factoring in the characteristics of the robot’s interaction with the environment and by tailoring the vision task to the specific requirements of a given behavioral task, one can often simplify the problem. As a result, vision techniques developed for autonomous robots tend to be computationally cheap, easy to implement, and reasonably robust.

Synthetic vision makes sense for a number of reasons. First, it may be the simplest and fastest way to extract useful information from the environment (e.g. using vision for low-level obstacle avoidance and navigation versus a purely analytical solution). This may be particularly true if one can take advantage of the rendering hardware. Moreover, synthetic vision techniques will probably scale better than analytical techniques in complex environments. Finally, this approach makes the creature less dependent on the implementation of its environment because it does not rely on other creatures and objects to respond to particular queries.

Our approach is simple. The scene is rendered from the creature’s eye view and the resulting image is used to generate a potential field from the creature’s perspective (this is done in an approach similar to that of Horswill). Subsequently, a gradient field is calculated, and this is used to derive a bearing away from areas of high potential. Following Arkin [1], some behaviors within the Behavior System represent their pattern of activity as a potential fields as well (for example, moveto). These potential fields are combined with the field generated by the vision sensor to arrive at a compromise trajectory.

This sensor is a simple example of using a technique borrowed from robotics. It was simple to implement, works well in practice, and is general enough to allow our virtual dog to wander around in new environments without modification.

## 6. BEHAVIOR SYSTEM

The purpose of the Behavior System is to send the “right” set of control signals to the motor system at every time-step. That is, it must weigh the potentially competing goals of the creature, assess

the state of its environment, and choose the set of actions which make the “most sense” at that instant in time. More generally, it provides the creature with a set of high-level behaviors of which it is capable of performing autonomously in a potentially unpredictable environment. Indeed, it is this ability to perform competently in the absence of external control which makes high level motivational or behavioral control possible.

Action-selection has been a topic of some interest among Ethologists and Computer Scientists alike, and a number of algorithms have been proposed [3, 4, 12, 19-22]. Earlier work [3], presented a computational model of action-selection which draws heavily on ideas from Ethology. The algorithm presented below is derived from this work but incorporates a number of important new features. The interested reader may consult [3] for the ethological justification for the algorithm. The remainder of this section describes the major components of the Behavior System, and how it decides to “do the right thing”

### 6.1 Behaviors

While we have spoken of a Behavior System as a monolithic entity, it is in fact a distributed system composed of a loosely hierarchical network of “self-interested, goal-directed entities” called Behaviors. The granularity of a Behavior’s goal may vary from very general (e.g. “reduce hunger”) to very specific (e.g. “chew food”). The major components of an individual Behavior are shown in Figure 6. This model of a distributed collection of goal-directed entities is consistent with ethological models as well as recent theories of the mind [15].

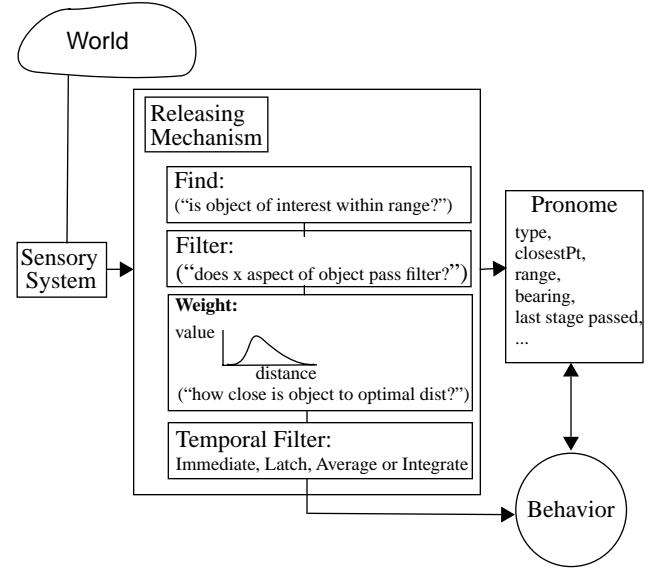
Behaviors compete for control of the creature on the basis of a value which is re-calculated on every update cycle for each Behavior. The value of a Behavior may be high because the Behavior satisfies an important need of the creature (e.g. its Internal Variables have a high value). Or it may be high because the Behavior’s goal is easily achievable given the Behavior’s perception of its environment (e.g. its Releasing Mechanisms have a high value).

Behaviors influence the system in several ways: by issuing motor commands which change the creature’s relationship to its environment, by modifying the value of Internal Variables, by inhibiting other Behaviors, or by issuing suggestions which influence the motor commands issued by other Behaviors.

Behaviors are distinguished from Motor Skills in two ways. First, a Behavior is goal-directed whereas a Motor Skill is not. For example, “Walking” in our model is a Motor Skill. “Moving toward an object of interest” is a Behavior. Second, a Behavior decides when it should become active, whereas a Motor Skill runs when something else decides it should be active. Typically, Behaviors rely on Motor Skills to perform the actions necessary to accomplish the Behavior’s goals.

### 6.2 Releasing Mechanisms and Pronomes

Behaviors rely on objects called “Releasing Mechanisms” to filter sensory input and identify objects and/or events which are relevant to the Behavior, either because they are important to achieving the Behavior’s goal, or because their presence determines the salience of the Behavior given the creature’s immediate environment. Releasing Mechanisms output a continuous value which typically depends on whether the stimuli was found, on its distance and perhaps on some measure of its quality. This is important because by representing the output of a Releasing Mechanism as a continuous quantity, the output may be easily combined with the strength of internal motivations which are also represented as continuous values. This in turn allows the creature to display the kind of behavior one finds in nature where a weak stimulus (e.g. day-old pizza) but a strong motivation (e.g. very hungry) may result in the same behavior as a strong stimulus (e.g. chocolate cake) but weak motivation (e.g. full stomach).



**Figure 7:** Releasing Mechanisms identify significant objects or events from sensory input and output a value which represents the strength of the stimulus. By varying the allowed maximum for a given Releasing Mechanism, a Behavior can be made more or less sensitive to the presence of whatever input causes the Releasing Mechanism to have a non-zero value. A Releasing Mechanism has 4 phases (Find, Filter, Weight and Temporal Filtering), as indicated above, each of which is implemented by callbacks. Releasing Mechanisms can often share the same generic callback for a given phase. Temporal Filtering is provided to deal with potentially noisy data.

While Releasing Mechanisms may be looking for very different objects and events, they typically have a common structure. This is described in more detail in Figure 7. The importance of this is that it is possible to share functionality across Releasing Mechanisms.

In addition to transducing a value from sensory input, a Releasing Mechanism also fills in a data structure available to the Behavior called a Pronome [15]. The Pronome acts like a pronoun in English: The use of Pronomes makes it possible for the Behavior to be written in terms of “it”, where “it” is defined by the Behavior’s Pronome. Thus, a “stopNearAndDo” Behavior can be implemented without reference to the kind of object it is stopping near. Pronomes may be shared among behaviors, thus allowing the construction of a generic “find and do” hierarchy. While the motivation for Pronomes comes from theories of mind [15] as opposed to ethology, they make sense in an ethological context as well. In any event, the use of Pronomes greatly facilitates the integration of external control, and simplifies the construction of behavior networks by providing a level of abstraction.

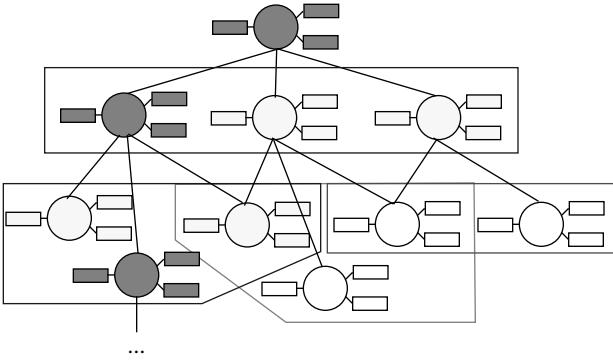
### 6.3 Internal Variables

Internal Variables are used to model internal state. Like Releasing Mechanisms, Internal Variables express their value as a continuous value. This value can change over time based on autonomous growth and damping rates. In addition, Behaviors can potentially modify the value of an Internal Variable as a result of their activity.

Both Releasing Mechanisms and Internal Variables may be shared by multiple Behaviors.

### 6.4 Behavior Groups

Behaviors are organized into groups of mutually inhibiting behaviors called Behavior Groups as shown in Figure 8. While we find a loose hierarchical structure useful this is not a requirement (i.e. all the Behaviors can be in a single Behavior Group). Behavior Groups are important because they localize the interaction among Behaviors which facilitates adding new Behaviors.



**Figure 8:** Behaviors are organized into groups of mutually inhibiting Behaviors called Behavior Groups. These Behavior Groups are in turn organized in a loose hierarchical fashion. Behavior Groups at the upper levels of the hierarchy contain general types of behaviors (e.g. “engage-in-feeding”) which are largely driven by motivational considerations, whereas lower levels contain more specific behaviors (e.g. “pounce” or “chew”) which are driven more by immediate sensory input. The arbitration mechanism built into the algorithm insures that only one Behavior in a given Behavior Group will have a non-zero value after inhibition. This Behavior is then active, and may either issue primary motor commands, or activate the Behavior Group which contains its children Behaviors (e.g. “search-for-food”, “sniff”, “chew” might be the children of “engage-in-feeding”). The dark gray behaviors represent the path of active Behaviors on a given tick. Behaviors which lose to the primary Behavior in a given Behavior Group may nonetheless influence the resulting actions of the creature by issuing either secondary or meta-commands.

## 6.5 Inhibition and Level of Interest

A creature has only limited resources to apply to satisfying its needs (e.g. it can only walk in one direction at a time), and thus there needs to be some mechanism to arbitrate among the competing Behaviors. Moreover, once a creature is committed to satisfying a goal, it makes sense for it to continue pursuing that goal unless something significantly more important comes along.

We rely on a phenomena known as the “avalanche effect” [15] to both arbitrate among Behaviors in a Behavior Group and to provide the right amount of persistence. This is done via mutual inhibition. Specifically, a given Behavior A will inhibit a Behavior B by a gain  $I_{AB}$  times Behavior A’s value. By (a) restricting the inhibitory gains to be greater than 1, (b) by clamping the value of a Behavior to be 0 or greater, and (c) requiring that all Behaviors inhibit each other, the “avalanche effect” insures that once the system has settled, only one Behavior in a Behavior Group will have a non-zero value. This model of inhibition was first proposed, in an ethological context by Ludlow [11], but see Minsky as well.

This model provides a robust mechanism for winner-take-all arbitration. It also provides a way of controlling the relative persistence of Behaviors via the use of inhibitory gains. When the gains are low, the system will tend to dither among different behaviors. When the gains are high, the system will show more persistence.

The use of high inhibitory gains can, however, result in pathological behavior in which the creature pursues a single, but unattainable goal, to the detriment of less important, but achievable ones. Ludlow addressed this problem by suggesting that a level of interest be associated with every Behavior. It is allowed to vary between 0 and 1 and it has a multiplicative effect on the Behavior’s value. When Behavior is active the level of interest decreases which in turn reduces the value of the Behavior regardless of its intrinsic value. Eventually, this will allow another Behavior to become active (this is known as time-sharing in the ethological literature). When the behavior is no longer active, its level of interest rises.

Inhibitory gains and level of interest provide the designer with a good deal of control over the temporal aspects of behavior. This is an important contribution of this algorithm.

## 6.6 Use of Primary, Secondary and Meta-commands

Being active means that a Behavior or one of its children has top priority to issue motor commands. However, it is extremely important that less important Behaviors (i.e. those which have lost the competition for control) still be able to express their preferences for actions. This is done by allowing Behaviors which lose to issue suggestions in the form of secondary and meta-commands as described earlier. These suggestions are posted prior to the winning Behavior taking its action, so it can utilize the suggestions as it sees fit.

For example, a dog may have a behavior which alters the dog’s characteristics so as to reflect its emotional state. Thus, the behavior may issue secondary commands for posture as well as for the desired ear, tail and mouth position, and use a meta-command for the desired gait. The use of a meta-command for gait reflects the fact that the behavior may not know whether the dog should go forward or not. However, it does know how it wants the dog to move, should another behavior decide that moving makes sense.

Despite the use of secondary and meta-commands, the winning behavior still has ultimate say over what actions get performed while it is active. It can over-rule a secondary command by removing it from the queue or by executing a Motor Skill which grabs a DOF needed by a given secondary command. In the case of meta-commands, the winning behavior can choose to ignore the meta-command, in which case it has no effect.

## 6.7 The Algorithm

The action-selection algorithm is described below. The actual equations are provided in appendix A.

On each update cycle:

(1) All Internal Variables update their value based on their previous value, growth and damping rates, and any feedback effects.

(2) Starting at the top-level Behavior Group, the Behaviors within it compete to become active. This is done as follows:

(3) The Releasing Mechanisms of each Behavior update their value based on the current sensory input. This value is then summed with that of the Behavior’s Internal Variables and the result is multiplied by its Level Of Interest. This is repeated for all Behaviors in the group.

(4) For each Behavior in the group, the inhibition due to other Behaviors in the group is calculated and subtracted from the Behavior’s pre-inhibition value and clamped to 0 or greater.

(5) If after step (4) more than one Behavior has a non-zero value then step (4) is repeated (using the post-inhibition values as the basis) until this condition is met. The Behavior with a non-zero value is the active Behavior for the group.

(6) All Behaviors in the group which are not active are given a chance to issue secondary or meta-commands. This is done by executing a suggestion callback associated with the Behavior.

(7) If the active Behavior has a Behavior Group as a child (i.e. it is not a Behavior at the leaf of the tree), then that Behavior Group is made the current Behavior Group and the process is repeated starting at step (3). Otherwise, the Behavior is given a chance to issue primary motor commands via the execution of a callback associated with the Behavior.

## 7. INTEGRATION OF DIRECTABILITY

Having described the Behavior and Motor Systems we are now in a position to describe how external control is integrated into this architecture. This is done in a number of ways.

First, motivational control is provided via named access to the Internal Variables which represent the motivations or goals of the Behavior System. By adjusting the value of a given motivational variable, the creature can be made more or less likely to engage in Behaviors which depend on that variable.

Second, all the constituent parts of a Behavior are also accessible at run-time, and this provides another mechanism for exerting

behavioral control. For example, by changing the type of object for which the Releasing Mechanism is looking, the target of a given Behavior can easily be altered (e.g. “fire hydrants” versus “user’s pants leg”). In addition, a Behavior may be made more or less opportunistic by adjusting the maximum allowed “strength” of its Releasing Mechanisms. A Behavior can be made in-active by setting its level of interest to zero.

Third, the Behavior System is structured so that action-selection can be initiated at any node in the system. This allows an external entity to force execution of a particular part or branch of the Behavior System, regardless of motivational and sensory factors which might otherwise favor execution of other parts of it. Since branches often correspond to task-level collections of Behaviors, this provides a form of task-level control.

Forth, it is easy to provide “imaginary” sensory input which in turn may trigger certain behaviors on the part of the creature. For example, objects may be added to the world which are only visible to a specific creature. The advantage of this technique is that it does not require the external entity to know anything about the internal structure of the creature’s Behavior System.

The mechanisms described above for controlling the Behavior System naturally support both prescriptive and proscriptive control. For example, by adjusting the level of a motivational variable which drives a given branch of the Behavior System, the director is expressing a weighted preference for or against the execution of that behavior or group of behaviors.

The multiple imperative forms supported by the Motor Controller allow the director to express weighted preferences directly at the motor level. For example, at one extreme, the director may “shut off” the Behavior System and issue motor commands directly to the creature. Alternatively, the Behavior System can be running, and the director may issue persistent secondary or meta-commands which have the effect of modifying or augmenting the output of the Behavior System. For example, the external entity might issue a secondary command to “wag tail”. Unless this was explicitly over-ruled by a Behavior in the Behavior System, this would result in the Dog wagging its tail. External meta-commands may also take the form of spatial potential field maps which can be combined with potential field maps generated from sensory data to effectively attract or repel the creature from parts of its environment.

## 8. IMPLEMENTATION

These ideas have been implemented as part of an object-oriented architecture and toolkit for building and controlling autonomous animated creatures. This toolkit is based on Open Inventor 2.0. Most of the components from which one builds a creature, including all of the components of the Behavior System, and the action-selection algorithm itself are derived from Inventor classes. This allows us to define most, of a creature and its Behavior System via a text file using the Inventor file format. This is important for rapid prototyping and quick-turnaround, as well as to facilitate the use of models generated via industry-standard modelers. We also make extensive use of Inventor’s ability to provide named access to field variables at run-time. This is important for integration of external control. Callbacks are used to implement most of the custom functionality associated with specific Releasing Mechanisms and Behaviors. This coupled with extensive parameterization reduces the need to create new subclasses. Lastly, an embedded Tcl interpreter is provided for interactive run-time control.

We have developed several creatures using this tool kit. For the Alive project [13], we have developed “Silas T. Dog” an autonomous animated dog which interacts with a user in a 3D virtual world in a believable manner. Silas responds to a dozen or so gestures and postures of the user, and responds appropriately (e.g. if the user bends over and holds out her hand, Silas moves toward the

outstretched hand and eventually sits and shakes his paw). The dog always looks at its current object of interest (head, hand, etc.), and when it is sad or happy, its tail, ears and head move appropriately. Silas has a number of internal motivations which he is constantly trying to satisfy. For example, if his desire to fetch is high, and the user has not played ball with him, he will find a ball and drop it at the person’s feet. Similarly, he will periodically take a break to get a drink of water, or satisfy other biological functions.

Silas represents roughly 3000 lines of C++ code, of which, 2000 lines are for implementing his 24 dog specific Motor Skills. He responds to 70 motor commands. His Behavior System is comprised of roughly 40 Behaviors, 11 Behavior Groups, 40 Releasing Mechanisms and 8 Internal Variables. Silas runs at 15Hz on a Onyx Reality Engine with rendering and sensing time (i.e. the second render for his synthetic vision) comprising most of the update time. The evaluation of the Behavior System itself typically takes less than 6-8 milliseconds.

We have also developed a number of creatures which are used in the context of an interactive story system [6]. This system features a computational director which provides “direction” to the creatures so as to meet the requirements of the story. For example, at the beginning of the story, a dog hops out of a car and wanders around. If the user, who is wearing a head-mounted display, does not pay attention to the dog, the director will send the dog over to the user. If the user still does not pay attention, the director effectively tells the dog: “the user’s leg is a fine replacement for a hydrant, and you really have to...”. The resulting behavior on the part of the dog usually captures the user’s attention.



Figure 9: Silas and his half brother Lucky.

## 9. CONCLUSION

Autonomy and directability are not mutually exclusive. We have detailed an architecture and a general behavioral model for perception and action-selection which can function autonomously while accepting direction at multiple levels. This multi-level direction allows a user to direct at whatever level of detail is desirable. In addition, this blend of autonomy and direction is demonstrated with several creatures within the context of two applications.

## Acknowledgments

The authors would like to thank Professors Pattie Maes, Alex P. Pentland and Glorianna Davenport for their support of our work. Thanks go, as well, to the entire ALIVE team for their help. In particular, thanks to Bradley Rhodes for suggesting the use of pronomes. We would also like to thank Taylor Galyean and Marilyn Feldmeier for their work modeling the setting and “Lucky.” This work was funded in part by Sega North America and the Television of Tomorrow Consortium.

## Appendix A.

### Behavior Update Equation:

$$v_{it} = \text{Max} \left[ \left( l_{it} \cdot \text{Combine} \left( \sum_k rm_{ki} \sum_j iv_{jt} - \sum_m n_{mi} \cdot v_{mt} \right) \right) 0 \right]$$

Where at time  $t$  for Behavior  $i$ ,  $v_{it}$  is its value;  $l_{it}$  is the level of interest;  $rm_{ki}$  and  $iv_{jt}$  are the values of Releasing Mechanism  $k$ , and Internal Variable  $j$ , where  $k$  and  $j$  range over the Releasing Mechanisms and Internal Variables relevant to Behavior  $i$ ;  $n_{mi}$  ( $n > 1$ ) is the Inhibitory Gain that Behavior  $m$  applies against Behavior  $i$ ;  $v_{mt}$  is the value of Behavior  $m$ , where  $m$  ranges over the other Behaviors in the current Behavior Group.  $\text{Combine}()$  is the function used to combine the values of the Releasing Mechanisms and Internal Variables for Behavior  $i$  (i.e addition or multiplication).

### Internal Variable Update Equations:

$$iv_{it} = (iv_{i(t-1)} \cdot damp_i) + growth_i - \sum_k effects_{kit}$$

Where at time  $t$  for Internal Variable  $i$ ,  $iv_{it}$  is its value;  $iv_{i(t-1)}$  is its value on the previous time step;  $damp_i$  and  $growth_i$  are damping rates and growth rates associated with Internal Variable  $i$ ; and  $effects_{kit}$  are the adjustments to its value due to the activity of Behavior  $k$ , where  $k$  ranges over the Behaviors which directly effect its value when active.

$$effects_{kit} = (modifyGain_{ki} \cdot v_{k(t-1)})$$

Where  $effects_{kit}$  is the effect of Behavior  $k$  on Internal Variable  $i$  at time  $t$ ;  $modifyGain_{ki}$  is the gain used by Behavior  $k$  against Internal Variable  $i$  and  $v_{k(t-1)}$  is the value of Behavior  $k$  in the preceding time step.

### Level of Interest Update Equation:

$$li_{it} = \text{Clamp}(((l_{i(t-1)} \cdot damp_i) + growth_i - (v_{i(t-1)} \cdot bRate_i)), 0, 1)$$

Where  $li_{it}$  is the Level Of Interest of Behavior  $i$  at time  $t$ , and  $bRate_i$  is the boredom rate for Behavior  $i$ .  $\text{Clamp}(x,y,z)$  clamps  $x$  to be between  $y$  and  $z$ . Note Level Of Interest is just a special case of an Internal Variable.

### Releasing Mechanism Update Equation:

$$rm_{it} = \text{Clamp}(\text{TemporalFilter}(t, rm_{i(t-1)}, \text{Find}(s_{it}, dMin_i, dMax_i) \cdot \text{Filter}(s_{it}) \cdot \text{Weight}(s_{it}, dOpt_i)), min_i, max_i)$$

Where  $rm_{it}$  is the value of Releasing Mechanism  $i$  at time  $t$ ;  $s_{it}$  is the relevant sensory input for  $i$ ;  $dMin_i$  and  $dMax_i$  are minimum and

maximum distances associated with it;  $\text{Find}()$  returns 1 or 0 if the object of interest is found within  $s_{it}$  and within  $dMin_i$  to  $dMax_i$ ;  $\text{Filter}()$  returns 1 or 0 if the object of interest matches some additional criteria;  $\text{Weight}()$  weights the strength of the stimulus based on some metric such as optimal distance  $dOpt_i$ ;  $\text{TemporalFilter}()$  applies a filtering function (latch, average, integration, or immediate) over some period  $t$ ; and  $\text{Clamp}()$  clamps the resulting value to the range  $min_i$  to  $max_i$ .

## REFERENCES

- Arkin, R.C., Integrating Behavioral, Perceptual, and World Knowledge in Reactive Navigation, in *Designing Autonomous Agents*, P. Maes, Editor. 1990, MIT Press, Cambridge, pp.105-122.
- Badler, N.I., C. Phillips, and B.L. Webber, *Simulating Humans: Computer Graphics, Animation, and Control*. 1993, Oxford University Press, New York.
- Blumberg, B. Action-Selection in Hamsterdam: Lessons from Ethology. in *Third International Conference on the Simulation of Adaptive Behavior*. Brighton, England, 1994, MIT Press. pp.108-117.
- Brooks, R., A Robust Layered Control System for a Mobile Robot. 1986. *IEEE Journal of Robotics and Automation* RA-2. pp. 14-23.
- Bruderlin, Armin and Thomas W. Calvert. Dynamic Animation of Human Walking. Proceedings of SIGGRAPH 89 (Boston, MA, July 31-August 4, 1989). In *Computer Graphics* 23, 3 (July 1989), 233-242.
- Galyean, T. A. *Narrative Guidance of Interactivity*, Ph.D. Dissertation, Massachusetts Institute of Technology, 1995.
- Girard, Michael and A. A. Maciejewski. Computational Modeling for the Computer Animation of Legged Figures. Proceedings of SIGGRAPH 85 (San Francisco, CA, July 22-26, 1985). In *Computer Graphics* 19, 263-270.
- Horswill, I. A Simple, Cheap, and Robust Visual Navigation System. in *Second International Conference on the Simulation of Adaptive Behavior*. Honolulu, HI, 1993. MIT Press, pp.129-137.
- Koga, Yoshihito, Koichi Kondo, James Kuffner, and Jean-Claude Latombe. Planning Motion with Intentions. Proceedings of SIGGRAPH 94 (Orlando, FL, July 24-29, 1994). In *Computer Graphics Proceedings, Annual Conference Series*, 1994, ACM, SIGGRAPH, pp. 395-408.
- Latombe, J. C., *Robot Motion Planning*. 1991, Kluwer Academic Publishers, Boston.
- Ludlow, A., The Evolution and Simulation of a Decision Maker, in *Analysis of Motivational Processes*, F.T.&T. Halliday, Editor. 1980, Academic Press, London.
- Maes, P., Situated Agents Can Have Goals. *Journal for Robotics and Autonomous Systems* 6(1&2), 1990, pp. 49-70.
- Maes, P., T. Darrell, and B. Blumberg. The ALIVE System: Full-body Interaction with Autonomous Agents. in *Proceedings of Computer Animation'95 Conference*, Switzerland, April 1995, IEEE Press, pp. 11-18.
- McKenna, Michael and David Zeltzer. Dynamic Simulation of Autonomous Legged Locomotion. Proceedings of SIGGRAPH 90 (Dallas, TX, August 6-10, 1990). In *Computer Graphics* 24, 4 (August 1990), pp.29-38.
- Minsky, M., *The Society of Mind*. 1988, Simon & Schuster, New York.
- Raiert, Marc H. and Jessica K. Hodgins. Animation of Dynamic Legged Locomotion. Proceedings of SIGGRAPH 91 (Las Vegas, NV, July 28-August 2, 1991). In *Computer Graphics* 25, 4 (July 1991), 349-358
- Renault, O., N. Magnenat-Thalmann, D. Thalmann. A vision-based approach to behavioral animation. *The Journal of Visualization and Computer Animation* 1(1), 1990, pp.18-21.

18. Reynolds, Craig W. Flocks, Herds, and Schools: A Distributed Behavioral Model. Proceedings of SIGGRAPH 87 (Anaheim, CA, July 27-31, 1987). In *Computer Graphics* 21, 4 (July 1998), 25-34.
19. Tinbergen, N., *The Study of Instinct*. 1950, Clarendon press, Oxford.
20. Tu, Xiaoyuan and Demetri Terzopoulos. Artificial Fishes: Physics, Locomotion, Perception, Behavior. Proceedings of SIGGRAPH 94 (Orlando, FL, July 24-29, 1994). In *Computer Graphics* Proceedings, Annual Conference Series, 1994, ACM, SIGGRAPH, pp. 43-50.
21. Tyrell, T. The Use of Hierarchies for Action Selection, in *Second International Conference on the Simulation of Adaptive Behavior*. 1993. MIT Press, pp.138-147.
22. Wilhelms J., R. Skinner. A 'Notion' for Interactive Behavioral Animation Control. *IEEE Computer Graphics and Applications* 10(3) May 1990, pp. 14-22.
23. David Zeltzer, Task Level Graphical Simulation: Abstraction, Representation and Control, in *Making Them Move*. Ed. Badler, N., Barsky, B., and Zeltzer D. 1991, Morgan Kaufmann Publishers, Inc.

# No Bad Dogs: Ethological Lessons for Learning in Hamsterdam\*

**Bruce M. Blumberg**

MIT Media Lab

E15-311, 20 Ames St.

Cambridge Ma. 01463

bruce@media.mit.edu

**Peter M. Todd**

Max Planck Inst. for Psychological Research,

Center for Adaptive Behavior and Cognition

Leopoldstrasse 24, 80802 Munich Germany

ptodd@mpipf-muenchen.mpg.de

**Pattie Maes**

MIT Media Lab

E15-305, 20 Ames St.

Cambridge Ma. 01463

pattie@media.mit.edu

## Abstract

We present an architecture for autonomous creatures that allows learning to be combined with action selection, based on ideas from ethology. We show how temporal-difference learning may be used within the context of an ethologically inspired animat architecture to build and modify portions of the behavior network, and to set fundamental parameters including the strength associated with individual Releasing Mechanisms, the time course associated with appetitive behaviors, and the learning rates to be used based on the observed reliability of specific contingencies. The learning algorithm has been implemented as part of the Hamsterdam toolkit for building autonomous animated creatures. When implemented in Silas, a virtual dog, the algorithm enables Silas to be trained using classical and instrumental conditioning.

## 1 Introduction

Action selection and learning represent two significant areas of research in behavior-based AI [Maes94], and advances in both areas are essential if we are to build animats that demonstrate robust adaptive behavior in dynamic and unpredictable environments. However, most work in this area to date has focused on one problem or the other. Several researchers [Maes90a, Blumberg94, Blumberg95, Tyrrell94, Tu94] have proposed ethologically inspired models of action selection that purport to handle many of the challenges associated with building animats which must juggle multiple goals at one time. However, this work did not incorporate learning, and the proposed architectures generally involved hand-built behavior networks and “not-a-few” parameters that had to be tweaked. In contrast, there has been a great deal of impressive work in learning [Watkins89, Whitehead92, Kaelbling92, Lin92, Mahadevan91, Sutton91], but the focus of this work has been on single-goal systems and on issues of optimality. Moreover, while it is now understood how to learn an optimal policy in certain kinds of worlds, it is also recognized that this is just the tip of the iceberg and that addressing the issues of “perceptual aliasing” and the “curse of dimensionality” are perhaps more important than the details of the underlying learning algo-

rithm itself. In other words, the structure in which learning takes place is as critical as the way the learning actually occurs. Among the few projects that pull these two research strands together, Klopff [Klopff93] developed a model that accounts for a wide range of results from classical and instrumental conditioning, and Booker [Booker88] and Maes [Maes90b] have integrated learning into more complete action selection algorithms. But much remains to be done.

In this paper, our contribution is to show how certain types of associative learning may be integrated into the action selection architecture previously proposed by Blumberg [Blumberg94, Blumberg95]. The underlying learning algorithm we use is Sutton and Barto’s Temporal Difference model [Sutton90], which builds and modifies portions of the behavior network in our ethologically inspired system. Furthermore, this algorithm sets several of the fundamental variables associated with our system, thus addressing some of the previous criticisms about proliferating free parameters. In particular, our system will:

- learn that an existing behavior can lead to the fulfillment of some previously-unassociated motivational goal when performed in a novel context (i.e., in more ethological language, the system will learn new appetitive behaviors).
- learn the context in which an existing behavior will directly fulfill a different motivational goal (i.e. learn new releasing mechanisms to be associated with a consummatory behavior).

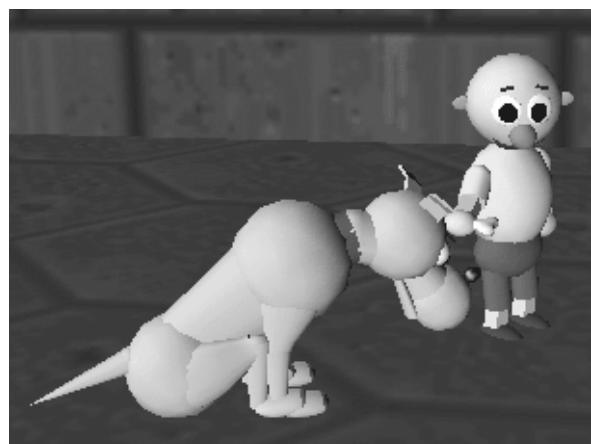


Figure 1: Silas T. Dog and his faithful teacher Dr. J.T. Puppet

\* Apologies to Barbara Woodhouse [Woodhouse82].

- learn new releasing mechanisms that are found to be useful predictors of the future state of the world (i.e. show classical conditioning).

This approach to action selection combined with learning has been implemented and incorporated into the architecture for “Silas T. Dog,” an autonomous, animated virtual dog featured in the ALIVE interactive virtual reality project [Maes95]. It has been successfully used to teach this old dog new tricks. By giving Silas the ability to learn, we hope to demonstrate the scientific feasibility of the coherent ethology-based model of animal behavior described in this paper. But this work also shows the usefulness of adding adaptability and user-trainability to autonomous VR agents intended to interact with people in an engaging and convincing manner. In this paper, we first discuss the important lessons from ethological research that have inspired the design of this learning system (Section 2), and then describe the basic action selection system underlying Silas’s behavior (in Section 3). We proceed to the new learning mechanisms in Section 4, and indicate their impact on behavior in Section 5. Finally, we discuss the implications and uses of this approach in Section 6.

## 2 Lessons from Ethology

Like the previous work on action selection in Hamsterdam [Blumberg94], our present efforts to incorporate learning are inspired by decades of ethological research [Dickinson94, Gallistel90, Gallistel94, Gould94, Lorenz73, Sutton90, McFarland93, Plotkin93, Shettleworth94, Davey89]. Here we briefly list the main take-home messages we have received from this literature and used with Silas to date.

**Learning complements evolution:** Learning is a mechanism for adapting to significant spatial and temporal aspects of an animal’s environment that vary predictably, but at a rate faster than that to which evolution can adjust (e.g. predictable changes occurring within a generation). The most adaptive course in this case is to evolve mechanisms that facilitate learning of these rapidly-varying features. Thus, evolution determines much of what can be learned and the manner in which it is learned. While this is sometimes viewed as imposing negative “constraints on learning,” the innate structure in which learning occurs most often has the effect of dramatically simplifying the learning process [Gallistel90, Gallistel94, Gould94, Lorenz73, Miller90, McFarland93, Plotkin93, Shettleworth94, Todd91].

**Motivations and goals drive learning:** Animals learn things that facilitate the achievement of biologically significant goals. Thus, motivations for those goals drive learning. It is far easier to train a hungry animal using food as a reward than a sated one [Gould94, Lorenz73, McFarland93]. The components of the lesson to be learned—that is, what things or actions facilitate a particular goal—are usually, but not always, contiguous in some sense with the satisfaction of the goal. For instance when a dog gets a cookie, the thing to learn is what behavior it performed just before getting that cookie.<sup>1</sup> (Taste-aversion learning occurring over a much longer time-period is an important example of non-temporally contiguous learning that we also wish to model.)

**Animals learn new appetitive behaviors:** Operant and instrumental learning can be viewed as an animal’s discovery of new appetitive behaviors—that is, new behaviors that bring them closer to attaining some goal [Lorenz73]. Specifically, operant conditioning occurs when an animal finds that an existing behavior (one that was typically performed in another context for another purpose) performed in a new stimulus situation will reliably lead to a state of the world in which performance of a consummatory behavior will successfully satisfy one or more motivational variables. For example, a dog that learns to roll over on command in order to get a cookie has added the “roll over in the context of the spoken word ROLL” behavior to its appetitive behaviors for feeding.

**Animals learn the value of stimuli and behaviors:** Stimuli that identify motivationally significant contexts, and appetitive behaviors that lead to goal-satisfying contexts, both have learnable values associated with them. This is an underlying assumption of many associative models of classical or operant conditioning [Sutton90, Klopff93, Montague94]. It is also a necessary assumption for integrating learning into an ethologically inspired model, such as that used in Hamsterdam, in which external and internal factors are expressed as real-valued quantities.

**Animals learn more than associations:** It is not enough simply to learn that some action X is a useful appetitive behavior. It is also useful to learn an expected time course for X, that is, how long to engage in action X before concluding that reinforcement is not forthcoming and that another behavior may be more successful. Gallistel [Gallistel90,94] presents evidence that in certain cases animals can learn the information relevant to this kind of time course knowledge, including the number of events, the time of their occurrence and the interval between them. Within an associative model, such as the one we develop here, it is also necessary to augment the system’s knowledge with such additional statistics in order to allow learning of a behavior’s expected time course.

---

1 Credit assignment is a hard problem for both animals and machines. Many of the innate mechanisms which facilitate learning in animals do so by solving, in effect, the credit assignment problem. This is done by either making the animal particularly sensitive to the relevant stimuli, or predisposed to learn the lesson when it is easiest to do so (e.g. song learning) [Gould94, McFarland93, Plotkin93]. With respect to animal training, trainers stress the importance of giving reinforcement within 2-4 seconds of the performance of a desired behavior and it is very difficult to train animals to perform sequences of actions, except by chaining together individually trained actions [Rogerson92, Lorenz94]. However, by breaking a sequence of desired actions into individually trained steps, the trainers are solving the credit assignment problem for the animal. Learning and training are flip sides of the same coin: learning tries to make sense of the world and put it into as simple a framework as possible, and training tries to make the world as simple as possible for the learning mechanisms to grasp.

### 3 Computational Model

In this section, we summarize our ethologically inspired computational model for action selection and learning in autonomous animated creatures. The underlying architecture for motor control and action selection is described in more detail in [Blumberg94, Blumberg95]. Here we will only describe enough of the underlying architecture to make it clear how learning is integrated into this system.

#### 3.1 Overview

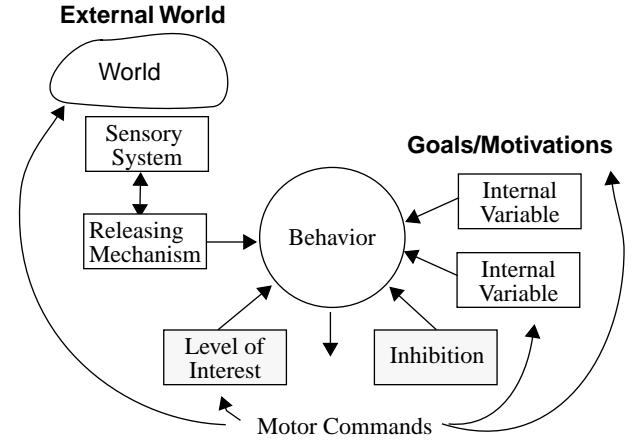
The basic structure of a creature in our system consists of the three basic parts (Geometry, Motor Skills and Behavior System) with two layers of abstraction between these parts (Controller, and Degrees of Freedom). The Geometry provides the shapes and transforms that are manipulated over time to produce the animated motion. The Motor Skills (e.g. “walking,” “wagging tail”) provide atomic motion elements that manipulate the geometry in order to produce coordinated motion. Motor Skills have no knowledge of the environment or state of the creature, other than that needed to execute their skill. Above these Motor Skills sits the Behavior System, which is responsible for deciding what to do, given a creature’s goals and sensory input. The Behavior System triggers the correct Motor Skills to achieve the current task or goal. Between these three parts are two layers of insulation, the Controller and the Degrees of Freedom (DOFs), which make this architecture generalizable and extensible. The DOFs, Controller, and Motor Skills together constitute what we call the Motor System.

A key feature of the Motor System is that it allows multiple behaviors to express their preferences for motor actions simultaneously. It is structured in such a way that Motor Skills that are complementary may run concurrently (e.g. walking and wagging the tail), but actions that are incompatible are prevented from doing so. Furthermore, the motor system recognizes 3 different imperative forms for commands: primary commands (i.e. do the action if the system is able to), secondary commands (i.e. do it if no other action objects) and meta-commands (i.e. do it this way). This allows multiple behaviors to express their preferences for motor actions.

There are at least three types of sensing available to autonomous creatures in our system:

- Real-world sensing using input from physical sensors.
- “Direct” sensing via direct interrogation of other virtual creatures and objects.
- “Synthetic vision” using computer vision techniques to extract useful information from an image rendered from the creature’s viewpoint. This is used for low-level navigation and obstacle avoidance, based on ideas from Horswill [Horswill93], and [Reynolds87].

Most of the learning in our system relies on “direct” sensing, in which objects in Silas’s world make information about their state available to Silas and other objects. For example, in the ALIVE interactive virtual reality system [Maes96], a user’s gestures in the “real world” are converted into state variables associated with the user’s proxy “crea-



**Figure 2:** The purpose of a Behavior is to evaluate its relevance given external stimulus and internal motivations, and if appropriate issue motor commands. Releasing Mechanisms act as filters which identify significant objects or events from sensory input, and output a value which represents the strength of the sensory input. Motivations or goals are represented via Internal Variables which output values which represent their strength. A Behavior combines the values of the Releasing Mechanisms and Internal Variables on which it depends to yield its value before Level of Interest and Inhibition from other Behaviors. Level of Interest is used to model boredom or behavior-specific fatigue. Behaviors compete, via mutual inhibition, with other Behaviors for control of the creature.

ture” in Silas’s world. Via direct sensing Silas can “sense” the state of this proxy creature, and thus react to the gestures of the user in the real world.

The purpose of the Behavior System is to send the “right” set of control signals to the motor system at every time-step. The “right set” is determined by weighing the current competing goals of the creature, assessing the state of the environment, and choosing the behavior (and its associated set of actions) that best satisfies some of the creature’s goals at this instant in time. More generally, the Behavior System coordinates the creature’s use of its available high-level behaviors in a potentially unpredictable environment.

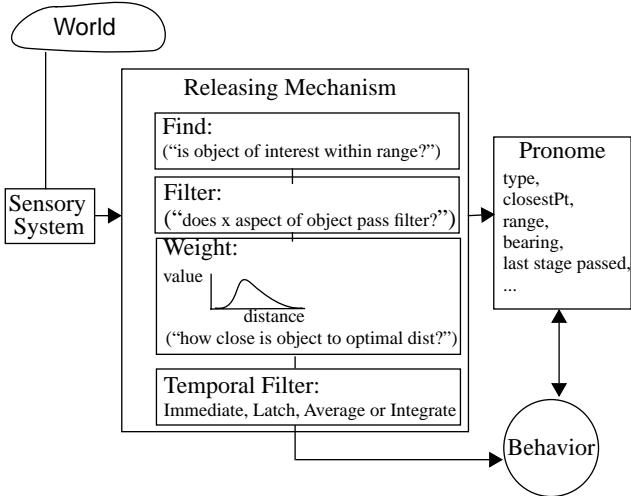
We will now turn to a discussion of the major components of the Behavior System and briefly describe the role of each component.

#### 3.2 Behaviors

The Behavior System is a distributed network of self-interested, goal-directed entities called Behaviors. The granularity of a Behavior’s goal can range from very general (e.g. “reduce hunger”) to very specific (e.g. “chew food”). The major components of an individual Behavior are shown in Figure 2.

Each Behavior is responsible for assessing its own current relevance or value, given the present state of internal and external factors. On the basis of this assessed value, each Behavior competes for control of the creature. The current value of a Behavior may be high because it satisfies an important need of the creature (as indicated by high values of the associated Internal Variables—see section 3.4), or because its goal is easily achievable given the current state of the environment (as indicated by high values of the associated Releasing Mechanisms—see Section 3.3).

Behaviors influence the system in several ways: by issuing motor commands which change the creature’s relation-



**Figure 3:** Releasing Mechanisms identify significant objects or events from sensory input and output a value which represents its strength. By varying the allowed maximum for a given Releasing Mechanism, a Behavior can be made more or less sensitive to the presence of the relevant stimuli. A Releasing Mechanism also fills in a data structure called a Pronome [Minsky86]. The Pronome can be thought of as representing the current focus of attention, which can be shared across behaviors.

ship to its environment, by modifying the value of Internal Variables, by inhibiting other Behaviors, or by issuing suggestions which influence the motor commands issued by other Behaviors.

Behaviors are distinguished from Motor Skills in two ways. For example, Behaviors are goal-directed whereas Motor Skills are not. For example, “Walking” is a Motor Skill whereas “Moving toward object of interest” is a Behavior which invokes the “Walking” Motor Skill to accomplish its goal. Second, Motor Skills do not decide when to become active, but rather rely on another entity to invoke them when appropriate. Motor Skills are similar to the Ethological idea of “Fixed Action Patterns” in that once initiated they do not require sensory input (other than proprioceptive input) and they have a specific time course [Lorenz76]. For the purposes of learning, the set of Motor Skills is assumed to be fixed. Thus, Motor Skill learning is an important type of learning which we do not address in this system at present. However, see [Giszter94] for an interesting example of motor learning which uses Maes’ algorithm.

### 3.3 Releasing Mechanisms

Objects called Releasing Mechanisms (see Figure 3) filter the creature’s sensory input to identify objects and events whose presence helps determine the current appropriateness of each possible Behavior. Releasing Mechanisms output a continuous value that typically depends on:

- the presence of the stimulus object or event (e.g. a person is nearby).
- more specific features associated with the stimulus (e.g. the person’s hand is down and extended).
- the distance to the stimulus relative to some ideal distance (e.g. the hand is one foot away, and I like hands that are two feet away).

By representing the output of a Releasing Mechanism as a continuous quantity, this value may be easily combined with the strength of motivations (from Internal Variables) that are

also represented as continuous values. This combination in turn allows the creature to display the kind of trade-off behavior one finds in nature where a weak stimulus (e.g. week-old pizza) but a strong motivation (e.g. very hungry) may result in the same behavior as a strong stimulus (e.g. chocolate cake) but weak motivation (e.g. full stomach).

Releasing Mechanisms return a value between some minimum and maximum, with the maximum value occurring when the Releasing Mechanism’s object of interest appears at some optimal distance from the creature. But how can we set this maximum value for a given Releasing Mechanism? The value of the Releasing Mechanism should reflect the usefulness of its associated Behavior to the creature, and hence its maximum should be equal to the time-discounted usefulness of its object of interest (which the Behavior will exploit). For example, for a food-sensitive Releasing Mechanism, its maximum value should be equivalent to the time-discounted value of the food source to the creature. We use the time-discounted value to reflect the fact that it will take a finite period of time to consume the food. Thus, with everything else being equal, a food source that provides more energy per time-tick will be favored over one which provides less energy per tick, even if the total amount of energy from the two food sources is the same. This perspective is central to our approach to learning.

Ethologists generally view Releasing Mechanisms as filtering external stimuli. We generalize this and allow Releasing Mechanisms to be inward looking as well, so for example, a Releasing Mechanism might be sensitive to the performance of a given behavior. This latter type of Releasing Mechanism is called a Proprioceptive Releasing Mechanism (PRM) to distinguish it from External Releasing Mechanisms (ERM) which are outward looking.

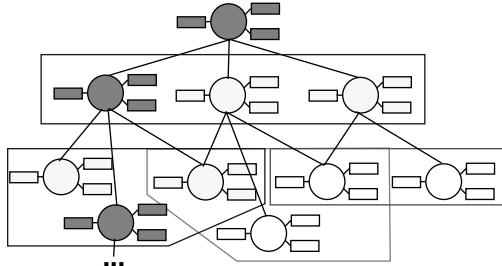
### 3.4 Internal Variables

Internal Variables are used to model internal state such as level of hunger or thirst. Like Releasing Mechanisms, Internal Variables are each expressed as a continuous value. This value can change over time based on autonomous increase and damping rates. In addition, certain Behaviors, such as “eating,” modify the value of an Internal Variable as a result of their activity. Ethologically speaking, these are “consummatory” behaviors, which when active have the effect of reducing the further motivation to perform that behavior. When a consummatory Behavior is active, it reduces the value of its associated motivational variable by an amount equal to some gain multiplied by the Behavior’s value. Thus, the higher the Behavior’s value, the greater the effect on the motivational Internal Variable.

Much of the learning that occurs in nature centers around the discovery either of novel appetitive behaviors (i.e. behaviors that will lead to a stimulus situation in which a consummatory behavior is appropriate), or of stimulus situations themselves in which a consummatory behavior should become active. More generally, one could say that the motivations that these consummatory behaviors satisfy drive much of the learning in animals. We adopt just such a perspective in our model: motivational Internal Variables are viewed as entities that actively drive the attempt to discover

new strategies (i.e. appetitive behaviors) that insure their satisfaction.

Based on this perspective, we can use the change in the value of a motivational variable due to the activity of a consummatory behavior as a feedback (or reinforcement) signal for the learning process. As we stated earlier in this section, changes in motivational Internal Variables are proportional to the value of the Behavior that changes them, which is in turn proportional to the value of the Releasing Mechanism that triggers it and the current level of motivation reflected in the Internal Variable value. Thus, more succinctly, motivational change-based reinforcement is proportional to both the quality of the stimulus (i.e., the value of the underlying Releasing Mechanism) and the level of motivation (i.e., the value of the changing Internal Variable).



**Figure 4:** Behaviors are organized into groups of mutually inhibiting behaviors called Behavior Groups. Behavior Groups in turn are organized in a loose hierarchical fashion. Behavior Groups at the upper levels of the hierarchy contain general behaviors (e.g. “engage-in-feeding”) which are largely driven by motivational considerations, whereas lower levels contain more specific behaviors (e.g. “pounce” or “chew”) which are driven largely by immediate sensory input. The algorithm’s arbitration mechanism insures that only one Behavior in a given Behavior Group will have a non-zero value after inhibition. This Behavior is then active, and may either issue primary motor commands, or activate the Behavior Group which contains its children behaviors (e.g. “search-for-food”, “sniff”, “chew” might be the children of “engage-in-feeding”). The dark gray nodes represent the path of active Behaviors on a given tick. Losing Behaviors in a given Behavior Group may nonetheless influence the resulting actions of the creature by issuing either secondary or meta-commands.

### 3.5 Behavior Groups

Behaviors are organized into mutually inhibiting groups called, simply, Behavior Groups (similar to Minsky’s cross-exclusion groups [Minsky86]). These are illustrated in Figure 4. While we find a loose hierarchical structure useful this is not a requirement (i.e. all the Behaviors can be in a single Behavior Group). Behavior Groups are important because they localize the interaction among Behaviors which in turn facilitates adding new Behaviors.

Typically, the Behavior Group at the top of the system is composed of Behaviors that, in turn, have their own subsumed Behavior Groups specialized for addressing a particular need of the creature (e.g. feeding or mating). These subgroups very often have the same structure, consisting of a consummatory behavior and one or more appetitive behaviors. The different appetitive behaviors represent alternative strategies for bringing the creature into a situation in which the consummatory behavior may be used. Associated with each of the appetitive behaviors are one or more Releasing Mechanisms that signal the appropriateness of this behavior given the current environment (i.e., the likelihood that this appetitive behavior will lead to a situation in which the consummatory behavior may become active).

Learning a new appetitive behavior is accomplished by adding this Behavior to an existing Behavior Group. For example, consider a dog learning that sitting when her owner says “Sit” will often lead to a reward. In this case, the “sitting” Behavior together with a Releasing Mechanism that fires on the verbal command “Sit” are both copied into the Behavior Group associated with feeding, and thus performing this trick becomes one more strategy that the dog can use for getting food.

### 3.6 Inhibition and Level of Interest

Any creature has only limited resources to apply to satisfying its needs (e.g. it can only walk in one direction at a time), and thus there needs to be some mechanism to arbitrate among the competing Behaviors within a Behavior Group, since they cannot all be in charge at once. Moreover, once a creature is committed to satisfying a particular goal, it makes sense for it to continue pursuing that goal unless something significantly more important comes along.

We follow Minsky [Minsky86] and Ludlow [Ludlow76,80] in relying on a model of mutual inhibition both to arbitrate among competing behaviors in a Behavior Group and to provide control for behavioral persistence. Mutual inhibition usually (see [Blumberg94,Blumberg95]) leads to what Minsky terms the “avalanche effect,” in which even weakly superior Behaviors can quickly come to dominate all the others. Ludlow provides two further important insights. The first is to recognize that by adjusting the inhibitory gains, one can vary the relative persistence of one Behavior versus the others. In general, the greater the gain, the more persistent the winning Behavior will be. The second insight is to associate a Level of Interest with every Behavior, indicating how “interested” the creature is in pursuing this Behavior right now. When the Behavior is active, its Level of Interest decreases, whether or not the Behavior achieved its goal. This in turn reduces the value of the Behavior regardless of its intrinsic value (i.e. its value before inhibition and Level of Interest were taken into account). Eventually, this will allow other Behaviors to become active. Level of Interest variables thus help the creature to avoid situations in which it obsesses on a single unattainable goal, performing the same currently ineffective Behavior over and over without result, and ignores less important but achievable goals.

Level of Interest also has a clear meaning in the context of an appetitive Behavior: it controls how long the creature should persist in the Behavior before giving up. For example, if the creature has learned from experience that a reward usually follows within X ticks of commencing some Behavior, then it should base the amount of time it will pursue this Behavior on this knowledge (i.e., it should accordingly adjust the amount by with the Level of Interest is decreased after each performance of the Behavior).

## 4 Integration of Learning

As indicated in Section 1, we want Silas to be able to learn: new contexts for goal-satisfying with existing behaviors; new releasing mechanisms for consummatory behaviors; and new releasing mechanisms with greater predictive

power that previously-used releasing mechanisms. The first two of these items are thinly veiled descriptions of what is usually referred to as operant or instrumental conditioning. In particular, learning new contexts for existing behaviors is essentially what animal training is all about (what Lorenz [Lorenz73] refers to as “conditioned action” or “conditioned appetitive behavior.”). That is, the animal learns an association between an behavior and an outcome, and also learns the stimulus configuration upon which this association is contingent. The third type of learning corresponds more closely to “classical conditioning” in that there is already a built-in association between a stimulus and a response (i.e. a Releasing Mechanism and a Behavior), and what is learned are one or more novel Releasing Mechanisms that have an incremental predictive value (e.g. can predict an outcome further in the future). In the first two cases, a change in an underlying motivational variable acts as the reinforcement or feedback signal. By contrast, in the third case feedback is provided by an existing Releasing Mechanism. In the rest of this section we outline how these types of associative learning can be integrated into the action selection architecture described in Section 3.

#### 4.1 Conceptual Overview of Learning Process

Fundamentally, the animal is trying to learn one of a number of potential associations:

- ( $S \rightarrow O$ ): Context S leads to outcome O, regardless of behavior
- ( $A \rightarrow O$ ): Behavior A leads to outcome O, regardless of context
- ( $S \rightarrow (A \rightarrow O)$ ): Behavior A leads to outcome O, in context S

Here, outcomes mean a significant change in a motivational variable. Given these possible associations in the world, what should the learning system pay attention to and make associations with when a particular outcome occurs?

Temporal and spatial contiguity answers the first part of this question. That is, when a motivational variable changes, the learning system looks at Silas’s recent behaviors and at the recently changed features of (nearby) objects of interest, to see if any of these may be associated with the outcome. Essentially, Silas says “let’s check short term memory and start watching the last  $N$  behaviors and objects of interest to see if they continue to be associated with changes in this motivational variable from now on.”

We use Sutton and Barto’s temporal difference model of Pavlovian conditioning [Sutton90] to actually learn the prevailing associations. This model effectively forces alternative “explanations” (i.e., behaviors or object features) to compete to “explain” (i.e., make the best prediction of) the time-discounted changes in the value of the reinforcement variable (i.e., the outcome). Sutton and Barto’s model is attractive because it is simple, yet it explains most of the results of Pavlovian conditioning experiments. It results in Silas learning a value for a particular behavior or stimulus that reflects its time-discounted association with changes in the value of some motivational variable.

In the first two types of associations, the change in the value of the underlying motivational variable (i.e., the outcome) acts as the reinforcement variable that drives learning,

making simple first-order associations. In the third case ( $S \rightarrow (A \rightarrow O)$ ), the learned value of A (in producing outcome O) acts as the reinforcement variable for learning O’s association with S. This allows Silas to learn second-order associations.

The fundamental insight that allows us to integrate this learning algorithm into our existing action selection algorithm is that the “value” to be learned for a behavior or stimulus is represented by the MaxValue associated with the Releasing Mechanism sensitive to that behavior or stimulus. In order to treat behaviors as learnable “cues” in the same way as external stimuli, we use PRMs (Proprioceptive Releasing Mechanism) to signal when a given behavior is active. In all cases, then, we are learning the appropriate MaxValue associated with a given Releasing Mechanism.

When the learned MaxValues of the relevant Releasing Mechanisms get above a threshold, this means that the association is “worth remembering,” and should be made a permanent part of the Behavior network. For example, in the case of a ( $S \rightarrow (A \rightarrow O)$ ) association worth remembering, this means that in context S, signalled by one or more Releasing Mechanisms, performance of behavior A has reliably led to outcome O, which corresponds to the performance of a consummatory behavior that reduces the value of the underlying motivational variable. More concretely, in the case of sitting (A) on command (S) for food (O), the sitting behavior and the set of releasing mechanisms (e.g., an outstretched hand) that indicate it’s time for Silas to do his trick will be added to the Behavior Group that is principally concerned with feeding. In addition, the Releasing Mechanisms that signal S will be added to the top-level behavior that owns the Feeding Behavior Group. This last step is necessary because S, the sitting-trick context, is now a relevant context for engaging in Feeding.

Because in this example ( $S \rightarrow (A \rightarrow O)$ ) represents a new appetitive strategy, it is also important to learn the appropriate time-course for the actions involved. That is, Silas should learn how long he should engage in the particular behavior of sitting, waiting for a cookie, before giving up. This information is computed as a by-product of the learning process and is used to change the Level Of Interest parameter of the behavior over time (see Section 4.2.6).

#### 4.2 Extensions to Support Learning

To add learning to the action selection mechanisms presented in Section 3, we have developed several new structures that we now describe in turn.

##### 4.2.1 Short-Term Memory

The Behavior System maintains FIFO memories that are essential for learning. One memory keeps track of the last  $N$  unique behaviors that have been active, and another keeps track of the last  $N$  objects of interest (currently we use  $N = 10$ ). The active behavior on a time-tick is the leaf behavior which ultimately won during action selection. The object of interest is the pronome associated with that behavior. Note that in both cases the memory is based on unique instances and not time. This idea is taken from Killeen: “Decay of short-term memory is very slow in an undisturbed environ-

ment and dismally fast when other events are interpolated...Such distractors do not so much subvert attention while time elapses but rather by entering memory they move time along” [Killeen94].

Foner and Maes [Foner94] also emphasize the importance of contiguity in learning. However, because short-term memory in our system is based on unique behaviors or objects of interest, our focus of attention is not strictly equivalent to temporal or spatial contiguity. In addition, we will probably move to a system in which each motivational variable has its own memory-list of potentially significant objects or events, again stripped of their temporal information. This would then make it possible to model long-time-span learning phenomena such as taste aversion.

#### 4.2.2 Reinforcement Variables

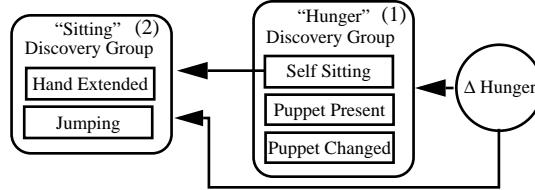
A Reinforcement Variable is either an Internal Variable (i.e. a motivational variable), or a Releasing Mechanism (either an ERM or a PRM) whose value is used as a feedback signal for learning. For example, hunger would be a Reinforcement Variable if the animal wished to learn new appetitive strategies to reduce its level of hunger. Alternatively, the PRM associated with the “Begging” behavior would be the Reinforcement Variable associated with learning the context in which Begging was a successful appetitive behavior for reducing hunger.

#### 4.2.3 Discovery Groups and the Learning Equation

A Discovery Group is a simply a collection of Releasing Mechanisms (either PRMs or ERMs) for which the animal wants to discover the appropriate association with a Reinforcement Variable. The Releasing Mechanisms in a Discovery Group effectively compete to “explain” the value of the Reinforcement Variable, and as they learn their association with the Reinforcement Variable, they change their maximum value accordingly. Thus, a Releasing Mechanism that has no apparent association with the state of a Reinforcement Variable will wind up with a maximum value close to zero. By contrast, a Releasing Mechanism which has a strong association will wind up with a maximum value that approaches the time-discounted value of the Reinforcement Variable.

We now turn to the question of how Discovery Groups are formed, that is, how creatures “decide” what to watch as potential sources of important associations. The first point to be made is that it is there is no “centralized” learning mechanism. Rather, each motivational variable is responsible for learning its own set of appetitive behaviors and their respective contexts using the mechanisms described in the previous sections. This means that each motivational variable has its own collection of Discovery Groups for which it is the Reinforcement Variable. Thus, exogenous changes (i.e. changes directly due to the performance of a behavior) in the motivational variable will serve as the feedback for the learning process in these Discovery Groups. This process is summarized in Figure 5.

The second point is that significant changes in the level of the motivational variable initiate the process of identifying candidates (i.e. ERMs or PRMs) to add to the Discovery



**Figure 5:** Discovery Groups (DG) are built and used by Reinforcement Variables (e.g. motivational variables) to explain significant changes in their value. For example, when hunger undergoes a significant change, it looks at short-term memory for the most recent behaviors performed and objects of interest encountered and adds the appropriate RMs (e.g. “Self-Sitting”) to its primary DG (1). Within a DG, the Barto-Sutton TD learning algorithm is used to discover the appropriate MaxValue for each RM. In order to learn the appropriate context for a behavior which appears to be important, a subsidiary DiscoveryGroup (2) may be constructed which has both the motivational variable and the proprioceptive RM for that behavior as its reinforcement variables. It will be populated with RMs which are sensitive to those features (e.g. “Hand-Extended”) of the recently encountered objects of interest which appear to change shortly before the motivational variable undergoes its change.

Groups. The fundamental assumption is one of contiguity: i.e., that one or more of the creature’s recent behaviors and/or one or more of the features associated with the creature’s recent objects of interest may have a causal or predictive association with the change in the motivational variable. In the current system, this means choosing candidates from the contents of short-term memory.

Discovery Groups and Reinforcement Variables are intended to be an abstraction of the temporal difference associative models proposed by researchers such as Klopf [Klopf93] or Sutton and Barto [Sutton90] to explain the results of classical conditioning. In their context, the Reinforcement Variable corresponds to the Unconditional Stimulus (UCS), and the Discovery Group corresponds to the collection of Conditional Stimuli (CS) that are competing to “explain” the value of the UCS.

We follow the Sutton and Barto TD learning model in this paper. Thus, within a Discovery Group we use the following equation to learn the association between a Reinforcement Variable and one or more Releasing Mechanisms:

$$\Delta V_{it} = \beta \left( \left[ \lambda_t + \gamma \left[ \sum_j V_{j(t-1)} \cdot A_{jt} \right] - \left[ \sum_j V_{j(t-1)} \cdot A_{j(t-1)} \right] \right] \alpha \bar{A}_{it} \right)$$

Where:

$V_{it}$  = MaxValue of RM i at time t

$\lambda_t$  = Value of reinforcement variable at time t

$\bar{A}_{it}$  = Trace for RM i at time t

$A_{it}$  = 1 if RM i passed find/filter and weight > .90, 0 otherwise

$\beta$  = Learning Rate

$\gamma$  = Temporal Discount Rate

$\alpha$  = Trace Rate

$\delta$  = Trace Decay Rate

Where the trace is defined as:

$$\bar{A}_{it} = \bar{A}_{i(t-1)} + \delta(A_{i(t-1)} - \bar{A}_{i(t-1)})$$

The first equation specifies how the MaxValue associated with a given RM in the Discovery Group will change in response to the value of the Reinforcement Variable. The amount it will change is proportional to an error defined to be the feedback actually received plus the discounted prediction of future feedback by all the Releasing Mechanisms in the Discovery Group less the prediction of feedback by all the Releasing Mechanisms on the previous time step. In

other words it changes by an amount proportional to the amount of actual and predicted feedback unexplained by the set of Releasing Mechanisms in the Discovery Group. In effect, they are competing for value, where the value is the time-discounted value of feedback.

The second equation is used to specify a “stimulus” or memory trace for a given Releasing Mechanism. It basically does temporal averaging of the state of the Releasing Mechanism, where A is either 1 or 0 (i.e. active or inactive) and A bar ranges between 0 and 1. This term is used by Sutton and Barto to explain temporally discontinuous associations between a CS and a UCS (e.g. when the CS turns off prior to the start of the UCS).

#### 4.2.4 A Variable Learning Rate

A variable learning rate is a necessary addition to the learning equation in order to explain two results found in animal learning. First, prior exposure to a CS without a correlated UCS delays learning a later association between the two. Second, it is well known that partial reinforcement, while lengthening training times, also lengthens the time to extinction. Neither of these effects would be predicted by the standard fixed-rate learning equation. Gallistel [Gallistel90] and Cheng [Cheng95] base their arguments against associative models in part on just these reasons.

To rescue associative learning, we propose that the learning rate should be proportional to some measure of the observed reliability of receiving, or not receiving, feedback. If the reliability is high -- e.g., a behavior is always rewarded, or always not rewarded -- then it makes sense to have a high learning rate, i.e. to pay attention to each observation. On the other hand, if the observed reliability is low -- e.g., a behavior is sometimes rewarded, and sometimes not -- then the learning rate should be low, reflecting the uncertainty associated with any one observation.

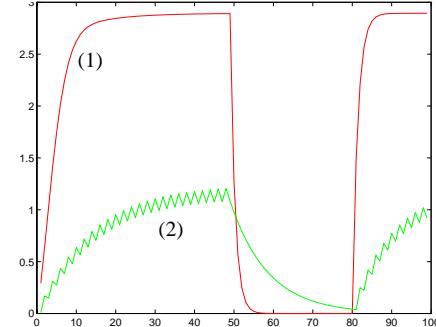
To implement this intuition, we base the learning rate on a moving average of what we call the “Reliability Contrast” which is defined to be:

$$RC = \frac{\#(\text{active \& feedback}) - \#(\text{active \& no feedback})}{\#(\text{active})} / \#(\text{active})$$

In other words, the RC is the difference between the observed probability of receiving feedback at least once while the behavior is active and the observed probability of receiving no feedback while the behavior is active. The RC ranges from reliably-unreliable (-1.0)—e.g. you will reliably not receive feedback to reliably-reliable (1.0)—e.g. you will reliably receive feedback. The equation for the learning rate is then:

$$\beta = \beta_{\min} + \text{fabs}(RC)(1 - \beta_{\min})$$

This variable learning rate allows us to accommodate one of the two weaknesses of the learning equation, namely the partial reinforcement effect. It will also accommodate the phenomena of rapid re-acquisition after extinction because in extinction the learning rate is high, and so it responds to the change rapidly. Both effects are shown in Figure 6. It does not, however, accommodate the phenomena of prior exposure reducing the learning rate. We also do not yet model the transition from a learned response to essentially a habit (i.e. when the learning rate decays to 0 and learning stops).



**Figure 6:** The graph shows the effect of a variable learning rate on the time-course in the associative strength of a stimulus under acquisition ( $t < 50$ ), extinction ( $50 < t < 80$ ), and reacquisition ( $t > 80$ ) in 2 feedback ratio scenarios ((1) = rewarded each time, and (2) = rewarded every other time). The learning rate is calculated in each case on a moving average of the reliability contrast. Since the reliability contrast is lower in the case of the variable reward scenario (2), its learning rate is lower and thus its decay in extinction is more gradual.

#### 4.2.5 Learning the Level of Interest

As mentioned earlier, in a world of uncertain rewards an animal needs to learn not just new appetitive behaviors, but also how long to engage in a given appetitive behavior before giving up and trying something else. We use a very simple approach to this problem: as part of the learning process, the Behavior incorporates its expected time to feedback and the associated variance. The Level of Interest for the Behavior is then held constant (and high) over the expected time to reward, and from that point on is allowed to decay linearly to zero over a period of time.

### 5 Implementation & Results

The algorithm described in the previous sections has been implemented as part of the Amsterdam toolkit for building and controlling autonomous animated creatures. We have developed several creatures using this toolkit, including the ALIVE [Maes96] project’s previously-mentioned virtual dog, Silas, a Puppet, and a Hamster [Blumberg95, Blumberg94]. Here we describe some of the initial results we have obtained for Silas and the hamster learning new positive and negative environmental associations.

Silas has 24 dog-specific Motor Skills and responds to 70 motor commands. In his current simplified form for testing learning, Silas initially has a Feeding Behavior Group which is empty except for a consummatory “eat” behavior—that is, this Behavior Group has no appetitive behaviors. Silas also has an exploratory Behavior Group that causes him to approach the user and randomly engage in a variety of different behaviors (e.g. sitting, begging, lying-down etc...).

We have used the learning algorithm described in the previous section to successfully train Silas to respond to gestures made by another autonomous creature, the Puppet (see Figure 1). The moment the Puppet detects that Silas is performing a desired behavior<sup>2</sup> (e.g. sitting), he performs the training gesture (e.g. hand-extended-down), and 10 ticks later (approximately 1/2 second), gives Silas a dog bone. Silas successfully learns that sitting when the Puppet’s hand is extended is a useful appetitive strategy for getting food, and thus adds the “Sit” behavior and its associated RM for

“hand-extended” to its “Feeding” system.

As the next step in teaching Silas to sit, the Puppet (an unexperienced dog-trainer) changes his training gesture to be the combination of “hand extended” and “jumping.” However, as a result of blocking, Silas attributes little value to the “jumping” component of the combined gesture, because it has no incremental predictive value beyond the “hand extended” component. But when the Puppet changes his training paradigm yet again and begins to jump several ticks before he extends his hand, Silas rapidly learns the new significance of “jumping”. This is shown in Figure 7.

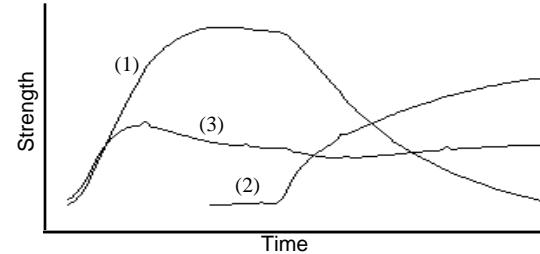
This experiment shows that the learning system described here can produce some of the phenomena associated with classical conditioning (not altogether surprising given our learning equation). Silas also learns the expected time to reward and adjusts the Level of Interest associated with “sitting” in this context accordingly, as discussed in Section 4.2.5.

In another experiment, we have a Hamster which is engaged in foraging in an open pen. The floor of the pen can be “electrified” and its color can change. The experiment is set up so that the floor-color will go from off to green to yellow to red. When the color is red, a shock is delivered for up to 40 ticks as long as the Hamster remains within a certain radius of the center of the pen. If the Hamster leaves the critical area, the shock is immediately turned off. The Hamster has a built-in response to shock: it will randomly choose to either “freeze” or to “run away.” It also has a built-in association that the shock comes from the environment (i.e. when it is shocked the environment becomes its object of interest). The motivational variable is “relief from pain.” Since the Hamster’s speed is such that it can leave the pen in substantially less time than the shock’s duration, running away should be the preferred action, and it should have an increasing tendency to run away as the floor-color progresses from green to red.

Over the course of 20 trials (i.e. 20 individual shocks), the Hamster learns that running away brings a quicker reduction in pain than does freezing, and so running away becomes the preferred option. In addition, it learns the association between the time course of the colors and being shocked and begins to flee when it senses the green light, or the yellow light at the latest. As a result, it successfully learns to avoid getting shocked.

We have also performed an experiment similar to that described by Montague [Montague94], in which our hamster learns to preferentially approach one food source over another based on the relative value of the alternative food sources. However, this needs to be combined with a motiva-

<sup>2</sup> Typically a dog trainer would issue the command “Sit” and then manipulate the dog into a sitting position [Rogerson92,Lorenz94]. Hence, the command precedes performance. To accomplish this in our system the Puppet considers the Dog to be performing a behavior the moment the behavior becomes active. However, the Dog (i.e. its PRMs) does not consider itself to be performing a behavior until both the behavior and its underlying motor skills are active. The effect is that from the Dog’s perspective the command precedes its perception of its own performance.



**Figure 7:** This shows the change in MaxValue for the “extendedRight” ERM (1), “jumping” ERM (2) and the “Sit” PRM (3) as a result of being associated with food during training. Initially, the Puppet uses a single gesture (“extendedRight”), but midway through training begins to simultaneously combine another gesture, “jumping”. As a result of blocking, little associative strength is allocated to the “jumping” PRM (flat portion of 2) until the Puppet changes its training paradigm again and “jumping” precedes “extendedRight” by 3 ticks. As a result, “jumping” rapidly gains strength at the expense of “extendedRight”.

tionally-based exploration strategy so that the Hamster will periodically test the less-preferred source to see if its value has changed. These results, along with those for Silas’s training, are still early and informal, but together they illustrate the power of the approach taken in this paper.

## 6 Areas for future work

Using the learning algorithm presented here we plan on demonstrating further learning effects from the ethological and psychological literature. For example, we want to demonstrate different phenomena including taste aversion, outcome devaluation, habit formation, learned helplessness and superstitious behavior. From the ethological literature, we want to demonstrate socially-facilitated learning [Gould94, Shettleworth95] and integrate an exploration vs. exploitation strategy which is under the influence of motivational variables. We would also like to be able to demonstrate chaining to produce behavioral sequences (e.g. as in a circus routine).

Learning new motor skills (or combinations of motor skills) represents a major area of behavioral innovation which is not addressed here and thus represents a limitation in the current work. However, the architecture of the Motor System would make it straightforward to add this type of learning.

## 7 Conclusion

Our contribution in this paper is to show how certain types of learning may be integrated into the animat architecture previously proposed by Blumberg [Blumberg94, Blumberg95]. The underlying algorithm is Sutton and Barto’s Temporal Difference model, but we show how it may be used and interpreted within the context of our ethologically inspired model to build and modify portions of the behavior network, and to set several of the fundamental parameters associated with our system.

It is not the intent of the authors to “push” one learning algorithm vs. another, but rather to stimulate thought on how best to integrate learning techniques such as TD learning into ethologically inspired multi-goal action selection architectures. This is done particularly with an eye towards building autonomous creatures such as “Silas” which must interact with, and learn from, users in natural ways.

We also hope that we have given the reader a sense of how ideas from ethology may be profitably incorporated into architectures for autonomous animated creatures.

## References

- Blumberg, B. (1994). Action-Selection in Hamsterdam: Lessons from Ethology. In: *From Animals To Animats, Proceedings of the Third International Conference on the Simulation of Adaptive Behavior*, Cliff, D., P. Husbands, J.A. Meyer, and S.W. Wilson, eds. MIT Press, Cambridge Ma.
- Blumberg, B. and T. Galyean (1995). Multi-level Direction of Autonomous Creatures for Real-Time Virtual Environments. In: *Proceedings of SIGGRAPH 95*.
- Booker L. (1988). Classifier Systems that Learn Internal World Models. *Machine Learning Journal*, Volume 1, Number 2,3.
- Brooks, R. (1986). A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation RA-2*.
- Cheng, P. and K.J. Holyoak (1995). Complex Adaptive Systems as Intuitive Statisticians: Causality, Contingency, and Prediction. In: *Comparative Approaches to Cognitive Science*, Roitblat, H.L. and J.A. Meyer, eds. MIT Press, Cambridge Ma.
- Davey G. (1989). *Ecological Learning Theory*. Routledge Inc., London.
- Dickinson, A. (1994). Instrumental Conditioning. In: *Animal Learning and Cognition*, Mackintosh, N.J. ed. Academic Press, San Diego.
- Foner, L.N. and P. Maes (1994). Paying Attention to What's Important: Using Focus of Attention to Improve Unsupervised Learning. In: *From Animals To Animats, Proceedings of the Third International Conference on the Simulation of Adaptive Behavior*, Cliff, D., P. Husbands, J.A. Meyer, and S.W. Wilson, eds. MIT Press, Cambridge Ma.
- Gallistel, C.R. (1990). *The Organization of Learning*. MIT Press, Cambridge Ma.
- Gallistel, C.R. (1994). Space and Time. In: *Animal Learning and Cognition*. Mackintosh, N.J. ed. Academic Press, San Diego.
- Giszter, S. (1994). Reinforcement Tuning of Action Synthesis and Selection in a Virtual Frog. In: *From Animals To Animats, Proceedings of the Third International Conference on the Simulation of Adaptive Behavior*, Cliff, D., P. Husbands, J.A. Meyer, and S.W. Wilson, eds. MIT Press, Cambridge Ma.
- Gould, J.L. and C.G. Gould (1994). *The Animal Mind*. Scientific American Library, New York.
- Horswill, I. (1993). A Simple, Cheap, and Robust Visual Navigation System. In: *Second International Conference on the Simulation of Adaptive Behavior*. Honolulu, HI. MIT Press
- Kaelbling, L. (1992). *Learning in Embedded Systems*, MIT Press, Cambridge Ma.
- Killeen, P.R. (1994). Mathematical principles of reinforcement. *Behavioral and Brain Sciences*, 17,105-172.
- Klopff, A.H., J.S. Morgan, and S.E. Weaver (1993). A Hierarchical Network of Control Systems that Learn: Modeling Nervous System Function During Classical and Instrumental Conditioning. *Adaptive Behavior*, Vol. 1, No. 3.
- Lin, L.J., and T.M. Mitchell (1992). *Memory Approaches to Reinforcement Learning in Non-Markovian Domains*, CMU-CS-92-138, Dept. of Computer Science, Carnegie-Mellon University.
- Lorenz, K. (1973). *Foundations of Ethology*. Springer-Verlag, New York.
- Lorenz, K. (1994). *Man Meets Dog*. Kodansha America, New York.
- Ludlow, A. (1976). *The Behavior of a Model Animal*. *Behavior*, Vol. 58.
- Ludlow, A. (1980). The Evolution and Simulation of a Decision Maker. In: *Analysis of Motivational Processes*, Halliday F.T. &T. eds. Academic Press, London.
- Maes, P. (1990a). Situated Agents Can Have Goals. *Journal of Robotics and Autonomous Systems* 6(1&2).
- Maes P. & R. Brooks (1990b). Learning to Coordinate Behaviors. In: *Proceedings of AAAI-90*.
- Maes, P. (1994). Modeling Adaptive Autonomous Agents. *Artificial Life*, Vol. 1, Numbers 1&2.
- Maes, P., T. Darrell, B. Blumberg, and A. Pentland (1996). The ALIVE System: Wireless, Full-Body Interaction with Autonomous Agents, (to appear in the ACM Special Issue on Multimedia and Multisensory Virtual Worlds, spring 1996)
- Mahadevan S. and J. Connell (1991). Automatic Programming of Behavior-Based Robots using Reinforcement Learning. In: *Proceedings of the Ninth National Conference on Artificial Intelligence*. MIT Press, Cambridge Ma.
- McFarland, D. (1993). *Animal Behavior*. Longman Scientific and Technical, Harlow UK.
- Miller, G.F., and P.M. Todd (1990). Exploring adaptive agency I: Theory and methods for simulating the evolution of learning. In: *Proceedings of the 1990 Connectionist Models Summer School*, Touretzky D.S., J.L. Elman, T.J. Sejnowski, and G.E. Hinton, eds. Morgan Kaufmann, San Mateo, Ca.
- Minsky, M. (1988). *The Society of Mind*. Simon & Schuster, New York.
- Montague, P.R., P. Dayan, and T.J. Sejnowski (1994). Foraging in an uncertain environment using predictive Hebbian learning. In: *Advances in Neural Information Processing 6*, Cowan, J.D., Tesauro, G., and Alspector, J. eds. Morgan Kaufmann, San Mateo, Ca.
- Plotkin, H.C. (1993). *Darwin Machines and the Nature of Knowledge*. Harvard University Press, Cambridge.
- Reynolds, C. W. (1987). Flocks, Herds, and Schools: A Distributed Behavioral Model. In: *Proceedings of SIGGRAPH 87*.
- Rogerson, J. (1992). *Training Your Dog*. Howell Book House, London.
- Shettleworth, S.J. (1994). Biological Approaches to the Study of Learning. In: *Animal Learning and Cognition*, Mackintosh, N.J. ed. Academic Press, San Diego.
- Sutton, R., and A.G. Barto (1990). Time-Derivative Models of Pavlovian Reinforcement. In: *Learning and Computational Neuroscience: Foundations of Adaptive Networks*. Gabriel, M. and J. Moore, eds. MIT Press, Cambridge Ma.
- Sutton R. (1991). Reinforcement Learning Architectures For Animats. In: *From Animals To Animats, Proceedings of the First International Conference on the Simulation of Adaptive Behavior*, Meyer, J.A. and S.W. Wilson, eds. MIT Press, Cambridge Ma.
- Todd, P.M., and Miller, G.F. (1991). Exploring adaptive agency II: Simulating the evolution of associative learning. In: *From animals to animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, Meyer J.A., and S.W. Wilson, eds. MIT Press, Cambridge Ma.
- Tu, Xiaoyuan and D. Terzopoulos (1994). Artificial Fishes: Physics, Locomotion, Perception, Behavior. In: *Proceedings of SIGGRAPH 94*.
- Tyrrell T. (1993). *Computational Mechanisms for Action Selection*. Ph.D. Thesis, Centre for Cognitive Science, University of Edinburgh.
- Watkins C. (1989). *Learning from Delayed Rewards*. Ph.D. Thesis, King's College, Cambridge.
- Whitehead S.D. (1992). *Reinforcement Learning for the Adaptive Control of Perception and Action*. Technical Report 406, University of Rochester Computer Science Dept.
- Woodhouse, B. (1982). *No bad dogs: the Woodhouse way*. Summit Books, New York.

# **Artificial Life meets Entertainment: Lifelike Autonomous Agents**

**Pattie Maes**  
**MIT Media Lab, Rm. E15-305**  
**20 Ames Street**  
**Cambridge, MA 02139**  
**pattie@media.mit.edu**

---

---

The relatively new field of Artificial Life attempts to study and understand biological life by synthesizing artificial life forms. To paraphrase Chris Langton, the founder of the field, the goal of Artificial Life is to "model life as it could be so as to understand life as we know it". Artificial Life is a very broad discipline which spans such diverse topics as artificial evolution, artificial ecosystems, artificial morphogenesis, molecular evolution and many more. [6] offers a nice overview of the different research questions studied by the discipline. Artificial Life shares with Artificial Intelligence its interest in synthesizing adaptive autonomous agents. Autonomous agents are computational systems that inhabit some complex, dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks that they are designed for.

The goal of building an autonomous agent is as old as the field of Artificial Intelligence itself. The Artificial Life community has initiated a radically different approach towards this goal which focuses on fast, reactive behavior, rather than knowledge and reasoning, as well as adaptation and learning. Its approach is largely inspired by Biology, and more specifically the field of Ethology, which attempts to understand the mechanisms which animals use to demonstrate adaptive and successful behavior.

Autonomous agents can take many different forms depending on the nature of the environment that they inhabit. If the environment is the real physical environment, then the agent takes the form of an autonomous robot. Alternatively, one can build 2D or 3D animated agents which inhabit simulated physical environments. Finally, so-called "knowbots", software agents or interface agents are disembodied entities which inhabit the digital world of computers and computer networks [9]. There are obvious applications for all these types of agents. For example, autonomous robots have been built for surveillance, exploration and other tasks in environments that are unaccessible or dangerous for human beings. There is a long tradition of building simulated agents for training purposes. Finally, more recently, interface agents have been proposed as one mechanism to help computer users deal with work and information overload [9].

One potential application area of agent research which has received surprisingly little interest so far is entertainment. This area may become much more important in the coming years, since the traditional main funding source of agent research, the defense industry, has been scaling down. Entertainment is an extremely large industry that is only expected to grow in the near future. Many forms of entertainment employ characters that act in some environment. This is the case for video games, simulation rides, movies, animation, animatronics, theater, puppetry, certain toys and even party lines. Each of these entertainment forms could potentially benefit from the casting of autonomous semi-intelligent agents as

entertaining characters. Entertainment is a fun and very challenging application area which will push the limits of agent research.

### The Challenge of Modeling Entertaining Characters

Several forms of commercial entertainment currently incorporate automated entertaining characters. Most of these characters are extremely simple: they demonstrate very predictable behavior and do not seem very convincing. This is in particular the case for characters with whom a person can interact in real-time, for example, video game characters. When automated characters show sophisticated behavior, it is typically completely mechanical and non-interactive and the result of a painstaking and laborious process. An example of the latter is the behavior of the dinosaurs in the movie Jurassic Park.

The last couple of years, a few exceptions have emerged. A number of researchers have applied agent technology to produce animation movies. Rather than scripting the exact movements of an animated character, the characters are modeled as agents which perform actions in response to their perceived environment. Reynolds [11] modeled flocks of birds and schools of fish by specifying the behavior of the individual animals that made up the flock. The same algorithms were used to generate some of the behavior of the bats in the movie Batman II. Terzopoulos [12] has modeled very realistic fish behavior including mating, feeding, learning and predation (figure 1). His models have been employed to make entertaining, short, animated movies.



**Figure 1: Realistic Fish behavior modeled by Terzopoulos et.al. to produce short animated movies.**

---

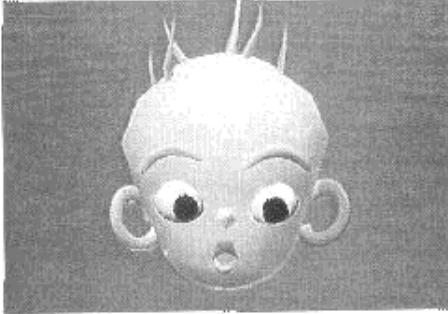
---

In addition to the previous work, some researchers have used agent models to build interactive, real-time animation systems. Bates' Woggles world [1] allows the user to interact with a world of creatures called Woggles. In this pioneering work, a user interacts with the world and its creatures using the mouse and keyboard to directly control the movements and behavior of one of the Woggles. The Woggles have several internal needs and a wide range of emotions, which results in fairly complex interactions. Fisher's Menagerie system [4] allows a user to interact with animated agents in real-time by using a head mounted display and a head tracking device. The agents in Menagerie are typically engaged in a single high-level behavior such as flocking. Tosa [13] used neural networks to model an artificial baby that reacts in emotional ways to the sounds made by a user looking into its crib (figure 2). Finally, the ALIVE system described below allows a user to enter a virtual world and use his/her whole body to

interact with animated autonomous agents (figure 4).

---

---



*Figure 2: Tosa's artificial "baby" reacts to sounds made by a user looking into its crib.*

---

---

In addition to computer animated agents, one notable original project is Julia [10], an autonomous agent that lives in a text based MUSE system (Multi-User Simulation Environment). Julia's behavioral repertoire includes moving around in and mapping of the environment, discourse with players, overhearing conversations and gossiping about them later on, relaying messages among players and helping players with navigation problems. Julia has moods, feelings and attitudes towards players and a good memory. She remembers what people have told her, have done to her, when she last encountered them, etc. Figure 3 includes some sample transcripts of interactions with Julia. Julia is an instance of what Mauldin has termed a "Chatterbot". A Chatterbot has several different modules for dealing with the different functions required to automate a player in a MUD. The conversation module is implemented as a prioritized layer of mini-experts, which are collections of patterns and associated potential responses. Chatterbots go beyond Weizenbaum's Eliza system in that they use more "tricks" and have more sophisticated memories of past events, conversations, etc.

---

---

```
Space-Ace says, "So why won't you kiss me, Julia?"  
Julia says, "Sorry, I'm not into geeks, Space-Ace."  
Space-Ace says, "Julia, I'm not a geek"  
Julia says, "Then lose the pocket protector, Space-Ace."  
Space-Ace says, "Julia, I don't have a pocket protector"  
Julia says, "Congratulations."  
Space-Ace says, "So kiss me, Julia"  
Julia says, "I'd rather be overhauled by a Yugo mechanic, Space-Ace."  
Space-Ace says, "julia, kiss me"  
Julia says, "Why, did your sheep leave you?"  
Space-Ace says, "julia, kiss me"  
Julia says, "Take a long walk through an unlinked exit, Space-Ace  
(copyright © 1990 by Ashne)."
```

```
Julia is offended by Lenny's suggestion.  
Julia winks at Lenny.  
Julia eyes Lenny carefully...
```

Julia grins at Lenny.  
Julia nods to Lenny.  
[...]

**Figure 3: Excerpts from interactions with "Julia", Mauldin's autonomous conversing agent which inhabits a text-based multi-user environment.**

---

---

Building these entertaining agents requires the same basic research questions that are central to all agent research to be addressed, namely perception, action selection, motor control, adaptation and communication. The agent has to perceive its environment, which is often dynamic and unpredictable, especially when a user is able to affect it. It has to decide what to do next so as to make progress towards the tasks it is designed to achieve. Relevant actions have to be translated in concrete motor commands. Over time, the agent has to change and improve its behavior on the basis of its past experience. Finally, the agent has to be able to communicate to other agents in the world, both human and artificial ones. The key problem is to come up with an architecture which integrates all these functionalities and which results in behavior that is fast, reactive, adaptive, robust, autonomous and last but not least, "lifelike". Life-like behavior is that which is non-mechanistic, non-predictable and spontaneous. The architectures of many of the successful entertaining agents listed above share a surprising number of features:

- The agents are modeled as distributed, decentralized systems consisting of small competence modules. Each competence module is an "expert" at achieving a particular, small, task-oriented competence. There is no central reasoner, nor a central internal model. The modules interface to one another via extremely simple messages. Each of the competence modules is directly connected to relevant sensors and effectors. As a result, the behavior produced is robust, adaptive to changes, fast and reactive.

Complex behavior is the result of interaction dynamics (feedback loops) at three different levels: interactions between the agent and the environment, between the different modules inside the agent and between multiple agents. For example, a simple Braitenberg creature which "loves" the user can be built by making it move in the direction of the user with a speed proportional to the distance from the user. As an example of complex multi-agent interaction, Reynolds' creatures demonstrate flocking behavior through the use of simple local rules followed by each of the creatures in the flock.

- The architecture includes a lot of redundant methods for the same competence. Multiple levels of complexity/sophistication ensure fault-tolerance, graceful degradation and non-mechanistic behavior. For example, Julia has several methods for responding to an utterance addressed to her: she can try to understand the utterance and generate a meaningful reply, or, if that fails, she can quote someone else on the same topic, or, if that fails, she can start a different conversation topic.

A more comprehensive discussion of the research questions in autonomous agent research and the characteristics of typical architectures for autonomous agents can be found in [7].

Apart from the more standard research questions, the design of entertaining agents also requires more novel (to a large degree to the Artificial Intelligence community and definitely to the Artificial Life one) questions to be dealt with, for example, how to model emotions, intentions, social behavior and discourse [1]. Typically these issues are even more important than making the agent very intelligent,

since, to quote Bates, "the actual requirement is to achieve a persistent appearance of awareness, intention and social interaction" [1]. Even though these topics may turn out to be of central importance to modeling and understanding intelligence, they have hardly been studied in Artificial Intelligence to date.

Finally, building entertaining agents requires the agent researcher to think more about the user. The researcher is forced to address the psychology of the user: how will the typical user perceive the virtual characters, what behavior will s/he engage in, what misconceptions and confusing situations may arise, and so on. Other disciplines such as human-computer interaction, animation, sociology, literature and theater are particularly helpful in answering these questions. For example, animation teaches us that users typically perceive faster moving characters as being younger, upbeat and more intelligent. Literature and theater teach us that it is easier for users to quickly grasp stereotype characters, such as the "shrink", in the Eliza program.

## The ALIVE project

A more detailed description of a particular project aimed at building entertaining agents may convey the research challenges and application opportunities of entertaining agents in a more convincing way.

ALIVE [8] is a virtual environment which allows wireless full-body interaction between a human participant and a virtual world which is inhabited by animated autonomous agents. ALIVE stands for "Artificial Life Interactive Video Environment". One of the goals of the ALIVE project is to demonstrate that virtual environments can offer a more "emotional" and evocative experience by allowing the participant to interact with animated characters.

The ALIVE system was demonstrated and tested in several public forums. It was demonstrated for 5 days at the SIGGRAPH-93 Tomorrow's Realities show in Anaheim, California and for 3 days at the AAAI-94 Art Show in Seattle, Washington. The system is installed permanently at the MIT Media Laboratory in Cambridge, Massachusetts. It will feature in the Ars Electronica Museum, currently under construction in Linz, Austria and the ArcTec electronic arts bienale in Tokyo, Japan.

In the style of Myron Krueger's Videoplace system, the ALIVE system offers an unencumbered, full-body interface to a virtual world [5]. The ALIVE user moves around in a space of approximately 16 by 16 feet. A video camera captures the user's image, which is composited into a 3D graphical world after the user's image has been isolated from the background. The resulting image is projected

onto a large screen which faces the user and acts as a type of "magic mirror" (figure 4): the user sees him/herself surrounded by objects and agents. No goggles, gloves, or wires are needed for interaction with the virtual world. Computer vision techniques are used to extract information about the person, such as his/her 3D location and the position of various body parts, as well as simple gestures performed. ALIVE combines active vision and domain knowledge to achieve robust and real-time performance [8].

---

---



**Figure 4:** The ALIVE system allows a user to use natural gestures to interact with a virtual world inhabited by animated autonomous agents such as this dog.

---

---

The user's location as well as his/her hand and body gestures affect the behavior of the agents in the virtual world. The user receives visual and auditory feedback about the agents' internal state and reactions. Agents have a set of internal needs and motivations, a set of sensors to perceive their environment, a repertoire of activities which they can perform and a physically-based motor system that allows them to move in and act on the environment. A behavior system decides in real-time which activity the agents engage in so as to meet their internal needs and to take advantage of opportunities presented by the current state of the environment.

The system allows not only for the obvious direct-manipulation style of interaction, but also for a more powerful, indirect style of interaction in which gestures can have more complex meaning. The meaning of a gesture is interpreted by the agents based on the situation the agent and user find themselves in. For example, when the user points away (figure 5) and thereby sends a character away, that character will go to a different place in the environment depending on where the user is standing (and which direction s/he is pointing). In this manner, a relatively small set of gestures can be employed to mean many different things in many different situations.

---

---



**Figure 5:** Gestures are interpreted by the agents based on the context. Here, the dog walks away in the direction the user is pointing.

---

---

The ALIVE system incorporates a tool, called "Hamsterdam" [3], for modeling semi-intelligent autonomous agents that can interact with one another and with the user. Hamsterdam produces agents that respond with a relevant activity on every time step, given their internal needs and motivations, past history and the perceived environment with its attendant opportunities, challenges and changes.

Moreover, the pattern and rhythm of the chosen activities is such that the agents neither dither between multiple activities, nor persist too long in a single activity. They are capable of interrupting a given activity if a more pressing need or an unforeseen opportunity arises. The Hamsterdam activity model is based on animal behavior models proposed by Ethologists. In particular, several concepts proposed in Ethology such as behavior hierarchies, releasers, fatigue, and so on, have proven to be crucial in guaranteeing the robust and flexible behavior required by autonomous interacting agents [3]. The ALIVE systems shows that animated characters that are based on Artificial Life models can look convincing (i.e. allow suspension of disbelief).

When using Hamsterdam to build an agent, the designer specifies the sensors of the agent, its motivations or internal needs, and its activities and actions. Given that information, the Hamsterdam software automatically infers which of the activities is most relevant to the agent at a particular moment in time according to the state of the agent, the situation it finds itself in and its recent behavior history. The observed behaviors or actions of the agent are the final result of numerous activities competing for control of the agent. The activities compete on the basis of the value of a given activity to the agent at that instant, given the perceived environment, the internal needs of the agent, and the agent's recent history. The details of the behavior model and a discussion of its features is reported upon in [3].

The ALIVE system consists of different virtual worlds between which the user can switch by pressing a virtual button. Each world is inhabited by different agents: one world is inhabited by a Puppet, a second one by a Hamster and Predator and a third one by a Dog. The Puppet follows the user around (in 3D) and tries to hold the user's hand. It also imitates some of the actions of the user (sitting down, jumping, etc.). It will be sent away when the user points away and will come back when the user waves. The Puppet employs facial expressions to convey its internal state. For example, it pouts when the user sends it away and smiles when the user motions it to come back. It giggles when the user touches its belly.

The Hamster avoids objects, follows the user around and begs for food. The Hamster rolls over to have its stomach scratched if the user bends over and pats it. If the user has been patting the Hamster for a while, its need for attention is fulfilled and some other activity takes precedence (e.g. looking for food). The user is able to feed the Hamster by picking up food from a virtual table and putting it on the floor. The user is also able to let the Predator out of its cage and into the Hamster's world. The Predator tries to chase and kill the Hamster. The Predator views the user as a predator and attempts to avoid and flee from the user. However, as it gets more hungry, it will become more bold and dare to come closer to the user. Both the Predator and the Hamster are successful at reconciling their multiple internal needs (avoiding the predator, finding food, not running into obstacles, etc.).

The most sophisticated character built so far is a dog called Silas. Silas's behavioral repertoire currently includes following the user, sitting (when asked by the user), going away (when sent away by the user) and performing other tricks such as jumping, fetching a ball, lying down and shaking. It also will chase the Hamster if that one is introduced in the dog world. Along with visual sensors and feedback, the Dog world also uses simple sound input and output. In addition to the camera, a directional microphone is facing the user. The resulting signal is fed into a simple pitch tracker and high pitch (e.g. clapping,

whistling, high voice) and low pitch (low voice) are interpreted respectively as positive and negative input from the user to the Dog. The Dog also provides auditory output, which consists of a variety of prerecorded samples.

By observing thousands of users interact with the agents in ALIVE, several things have been learned. First of all, the gestures which the user can engage in should be intuitive with respect to the domain and should provide immediate feedback. To user interface designers, the latter will seem obvious, but to Artificial Life and Artificial Intelligence researchers it isn't. Examples of "natural" gestures are petting for creatures or pointing and waving for the virtual Puppet. Whenever a gesture is successfully perceived by an agent, the user should receive immediate feedback, either in terms of movement and/or facial and body expression (e.g. the Hamster rolls over when being patted, the Puppet smiles when being tickled, etc.). This helps the user develop an understanding for the space of recognized gestures.

Second, even if the gestures are natural to the environment, it is still necessary to have a human "guide" present to give the user hints about what s/he could do (i.e. "try petting the Hamster", "the Puppet will go away if you point", etc.). The current ALIVE system includes an artificial guide which is implemented as yet another autonomous agent that fulfills a special role: it observes the user and memorizes what kinds of interactions the user has had with the world and occasionally offers suggestions by speaking as well as by using gestures. The guide is visualized as a Parrot in the virtual world because people expect a Parrot to talk, but they do not expect that the Parrot will be able to understand speech.

Third, users are more tolerant of the imperfections in an agent's behavior (as opposed to that of objects), such as lags and occasional incorrect or missed recognition. Having agents causes people to have appropriate expectations about the performance of the sensor system. We learned that people expect virtual inanimate objects to "work" reliably, i.e. the reaction of the object has to be immediate, predictable and consistent. On the other hand, people assume that animal or human-like agents have perception and state, and thus are able to accept that the agent may not have sensed something. As a result, gestures which are hard to recognize, such as waving, can be used successfully in the context of agents (an agent might not have "seen" the user waving), but the same gesture would cause the user frustration if used in the context of some inanimate object, e.g. a switch.

Fourth, it is important to visualize the motivational and emotional state of the agents in the external features of the agent. For example, a more sophisticated creature such as Silas the dog will "lead" with its eyes, that is, he turns to look at an object or a person before he actually walks over to the object or the person (e.g. to pick the object up or to invite the person to play). If a character does not lead with its eyes, its behavior looks very mechanical and as such not very life-like. Another reason why it is necessary to visualize motivations and internal state is that the user may get confused or frustrated if s/he cannot perceive internal variables which determine the behavior of the agent. For example, if Silas is hungry, he may not be very obedient. It is important that the user can observe that Silas is hungry, so that s/he realizes that this may be why he behaves very differently from a few minutes ago.

Finally, the most important lesson learned is that for an immersive environment to be captivating, it may not be so important how fancy the graphics are, but rather how meaningful the interactions that the user engages in can be. ALIVE users reported having a lot of fun using the system and interacting with the creatures. In particular, they seemed to enjoy worlds inhabited by "emotional" agents, characters the user can have an emotional relationship with the most. For example, users were very much intrigued by the facial expressions of the Puppet and would feel bad when their actions caused the Puppet to pout or

would feel happy when they caused it to smile.

The ALIVE system demonstrates that entertainment can be a challenging and interesting application area for autonomous agents research. ALIVE provides a novel environment for studying architectures for intelligent autonomous agents. As a testbed for agent architectures, it avoids the problems associated with real hardware agents or robots, but at the same time forces us to face non-trivial problems, such as dealing with noisy sensors and an unpredictable, fast changing environment. It makes it possible to study agents with higher levels of cognition, without oversimplifying the world in which these agents live.

ALIVE represents only the beginning of a whole range of novel applications that could be explored with this kind of system. We are currently investigating ALIVE for interactive story telling applications in which the user plays one of the characters in the story and all other characters are artificial agents which collaborate to make the story move forwards (cfr. 3 short papers on ALIVE in [2]). Another obvious entertainment application of ALIVE is video games. We have hooked up the ALIVE vision-based interface to existing video game software, so as to let the user control a game with his/her full body. In addition, we are investigating how autonomous video game characters can learn and improve their competence over time, so as to keep challenging a video game player. Finally, we are modeling animated characters that teach a user a physical skill in a personalized way. The agent is modeled as a personal trainer that demonstrates to the user how to perform an action and provides personalized and timely feedback to the user, on the basis of the sensory information about the user's gestures and body positions.

## Conclusion

Recently developed systems such as the Woggles [1], Neurobaby [13], Terzopoulos' Fish [12], Julia [10] and ALIVE [8]. demonstrate that entertainment can be a fun and challenging application area for Autonomous Agents research. However, at the same time, these early experiments demonstrate that this application area will require a more interdisciplinary approach which combines the know-how of the human sciences with the computational models developed in Artificial Life and Artificial Intelligence.

## References

- [1] Joseph Bates, "The Role of Emotion in Believable Characters", Communications of the ACM, Volume 37, Number 7, July 1994.
- [2] Joseph Bates, Barbara Hayes-Roth and Pattie Maes, Workshop notes of the AAAI Spring Symposium on Interactive Story Systems: Plot and Character, AAAI, March 1995.
- [3] Bruce Blumberg, "Action Selection in Hamsterdam: Lessons from Ethology", Proceedings of the 3rd International Conference on the Simulation of Adaptive Behavior, Brighton, MIT-Press, August 1994.
- [4] Scott Fisher et.al., "Menagerie", SIGGRAPH-93 Visual Proceedings, Tomorrow's Realities, ACM SIGGRAPH 1993, pp. 212-213, 1993.
- [5] Krueger M.W., Artificial Reality II, Addison Wesley, 1990.
- [6] Chris Langton (editor), Journal of Artificial Life, Volume 1, Number 1/2, MIT Press, Fall

1993/Winter 1994.

- [7] Pattie Maes, "Modeling Adaptive Autonomous Agents", Journal of Artificial Life, Volume 1, Number 1/2, MIT Press, Fall 1993/Winter 1994.
  - [8] Pattie Maes, Trevor Darrell, Bruce Blumberg and Alex Pentland, "The ALIVE system: Full-body Interaction with Autonomous Agents", Proceedings of the Computer Animation '95 Conference, Geneva, Switzerland, IEEE-Press, April 1995.
  - [9] Pattie Maes, "Agents that Reduce Work and Information Overload", Communications of the ACM, Volume 37, Number 7, July 1994.
  - [10] Michael Mauldin, "ChatterBots, TinyMuds, and the Turing Test: Entering the Loebner Prize Competition", Proceedings of the AAAI 1994 Conference, MIT Press, pp. 16-21.
  - [11] Craig Reynolds, "Flocks, Herds and Schools: A Distributed Behavioral Model", Computer Graphics: Proceedings of SIGGRAPH '87, 21(4), ACM Press, July 1987.
  - [12] Demetri Terzopoulos et.al., "Artificial Fishes with Autonomous Locomotion, Perception, Behavior and Learning, in a Physical World", Proceedings of the Artificial Life IV Workshop, Pattie Maes and Rod Brooks (editors), MIT Press, 1994.
  - [13] Naoko Tosa, "Neurobaby", SIGGRAPH-93 Visual Proceedings, Tomorrow's Realities, ACM SIGGRAPH 1993, pp. 212-213, 1993.
- 
- 



[Back to the Autonomous Agents Group Home Page](#)



[To Pattie Maes' Home Page](#)