Artificial intelligence in computer graphics: A constructionist approach

Article *in* ACM SIGGRAPH Computer Graphics · February 2004

DOI: 10.1145/1012272.1012275

4 authors, including:

Kristinn R. Thórisson
Reykjavik University
123 PUBLICATIONS **2,509** CITATIONS
SEE PROFILE

Thor List
18 PUBLICATIONS **335** CITATIONS
SEE PROFILE

Artificial Intelligence in Computer Graphics: A Constructionist Approach

Kristinn R. Thórisson, Christopher Pennock, Thor List, John DiPirro
Communicative Machines

Introduction

The ultimate embodiment of artificial intelligence (AI) has traditionally been seen as physical robots. However, hardware is expensive to construct and difficult to modify. The connection between graphics and AI is becoming increasingly strong, and, on the one hand, it is now clear that AI research can benefit tremendously from embodiment in virtual worlds [1, 4, 5]. On the other hand, computer games, a highly visible area of computer graphics development, could benefit from the use of more advanced AI [11]. But a synergistic marriage of the two fields has yet to occur.

A substantial impediment to introducing more intelligent characters into games is the lack of practitioners who understand both the realms of graphics and of AI, and who can drive their integration. Another is the diversity in programming languages and environments that people from both camps use.

We are working to address these problems using a multi-prong approach. Here we present two of these, which are interrelated. First, we present a new design methodology aimed at making the construction of AI systems easier for novices and experts alike. Second, we present network-based software designed to take advantage of this methodology, allowing general-purpose systems-level integration of AI programs with other systems, including graphics. The software is available free of charge to researchers.

An underlying assumption for this work is the theory that the mind can be modeled through the adequate combination of interacting, functional machines, or modules. In his seminal book, *Unified Theories of Cognition,* the late Allen Newell (1990) [16] called on researchers to start working on unified theories of mind, rather than continuing working with micro-theories about isolated components. We agree strongly with Newell in his quest for integration, and we do not interpret his call to mean a corollary to the search for the unified field theory in physics. Rather, we subscribe to the implications of Minsky's (1986) Society-of-Mind theory [8] that the mind is a multitude of diverse, interacting components. Our methodology and software foundation directly supports the construction of such systems.

The motivations for this work are numerous. Large systems such as virtual worlds with simulated inhabitants cannot be built from scratch without bringing together a large team of experts in each field that such a system naturally encompasses. Our methodology aims to help coordinate the effort. There is also a lack of incremental accumulation of knowledge in AI and related computer graphics. By supporting reuse of prior work we enable the building of increasingly powerful systems, as core system elements do not need to be built from scratch.

Background and Motivation

The creation of believable, interactive characters is a challenge familiar to many game developers. Systems that have to respond to asynchronous inputs in real time, where the inputs' timing is largely unpredictable, can pose a serious challenge to the system design. An even bigger challenge is presented by the complexity explosion in systems where the inputs include a high-bandwidth mixture of spatial and symbolic information, covering a wide range of meaning. This is the case for developers building communicative humanoids that interact with real people through state-of-the-art tracking technologies.

One of the main challenges of AI is, therefore, complexity management. This complexity is quite different from that encountered in computer graphics, where a small set of basic principles applied to a somewhat larger number of object types results in well-understood and predictable behavior, enabling the power of graphics systems to grow at roughly the same rate as the hardware. Not so in artificial intelligence.

Typically spanning from perception to action, AI systems necessarily touch on many capabilities of the mind that are not well understood. For a humanoid, those that are necessary to create a compelling simulation include, at a minimum, vision, hearing, planning and animation control. To be sure, much software has been written in the last two decades for handling cognitive, sensory and motor tasks, but these are typically implemented in isolation, in various programming languages and with a wide range of background assumptions about the operating environment. There is thus a pragmatic kind of complexity with which the AI practitioner must cope: The broad set of skills required to create and/or configure the necessary systems for processing input and output in these areas.

Another difficulty here involves the common research setting under which such AI projects are undertaken. Most research on humanoids today is done in small teams. Assuming the right team of researchers has been formed, it can still be a challenge to achieve effective collaboration between people with different backgrounds, such as computer graphics, hardware and artificial intelligence [2, 13].

Constructionist Artificial Intelligence

Constructionist AI is a practically driven approach to building AI systems. The Constructionist AI Methodology (CAIM) is an emerging set of principles for designing and implementing interactive intelligences, speeding up implementation of relatively complex, multi-functional systems with full, real-time perception-action loop capabilities. It helps novices as well as experts with the creation of embodied, virtual agents that can perceive and act in real time and interact with virtual as well as real environments. The embodiment of such humanoids can come in various levels of completeness, from just a face to a full body. Of special interest to us is human communication, including speech, gesture, facial expressions and gaze, in both the input and the output.

The Constructionist AI Methodology - so called because it advocates the use of modular building blocks and emphasizes incorporation of earlier work - addresses systems endowed with the full scope of human interactivity, from decisecond eye movement control to fully-planned, long-term strategic thinking over minutes, hours and days (the input is typically, purely for practical reasons, somewhat less comprehensive). CAIM is a highly modular approach; with a strong separation of functional elements and their interaction, the gross anatomy of architectures resulting from its use can in fact be faithfully replicated using physical LEGO blocks [Figure 1]. Here we give a short overview of the methodology for non-AI experts; those interested in the full details are referred to [12].

When beginning the construction of a mind for an interactive graphical character, the high-level goals of the system are first defined, along with the system's scope. This is done by writing scenarios with narratives that cover the various parts of the system. Then follows a period of modularization where division of labor is used to come up with functional modules and message types. The role of each module is determined in part by specifying the message types and content that needs to flow between the various functional parts of the system to support the system's operational goals. Thus the boundaries between modules are first delineated by specifying their inputs and outputs in the form of messages. Messages, and their content, are continuously refined as design progresses.

The principle of divisible modularity [12] prescribes iterative revision of modules through repeated division of their functionality into a set of ever-smaller interacting modules. CAIM provides several heuristics for how best to modularize. Among these are: Classifying modules into the three roles of perception, decision and action; avoiding duplication of information in various parts of the system;

and following the natural breaks along low-bandwidth information flow in the system. Modularization through explicit messages means that system designers can build out several parts of the system in parallel.

Constructionists take a breadth-first approach; every module starts out relatively simple, reading one type of message and posting another. This way, a full end-to-end chain of the whole system can be built for a single interaction case. Every element in the path should be tested on paper, along with the routing mechanisms, timing assumptions, etc., very early in the design process, and continuously over the course of development.

The relatively easy reconstruction of Ymir [13] as a physical LEGO model [Figure 1] demonstrates the Constructionist method's power to represent complex systems of multiple, interacting processes in a tangible way. The LEGO model of Ymir in turn exposes several features of the approach. The model represents the distribution of modules into perception, decision and action, as well as blackboards [10], motor scheduling and knowledge bases (decision support). Signal paths are shown via antennas of different colors: Modules with yellow antennas communicate with the blackboard that has the same-colored antenna. The star-shaped antenna allows knowledge bases to talk to each other (directly or via a separate blackboard). The model also demonstrates the concept of a three-layer priority scheme, indicated here by the three stories: Modules on the "ground floor" have the highest priority, as these service high-speed actions such as gaze, turn-taking and other communicative movements. Modules on the "top-floor" take lowest priority, as these are assumed to be the slowest processes in the system (it takes longer to formulate an answer to a question than it takes to glance in the direction of a loud noise). Several decision modules, as well as the knowledge bases, communicate with the animation scheduling and rendering system, which sits on top of a library of behavior modules, which inherit from each other and represent a hierarchy of increasingly complex motor plans [14]. With layers of modules in place, a small number of additional modules results in an exponential increase of the system's sophistication. Ymir was implemented in LISP and C/C++ [15], using an early version of CAIM, and used to control the massively interactive character Gandalf [Figure 2].

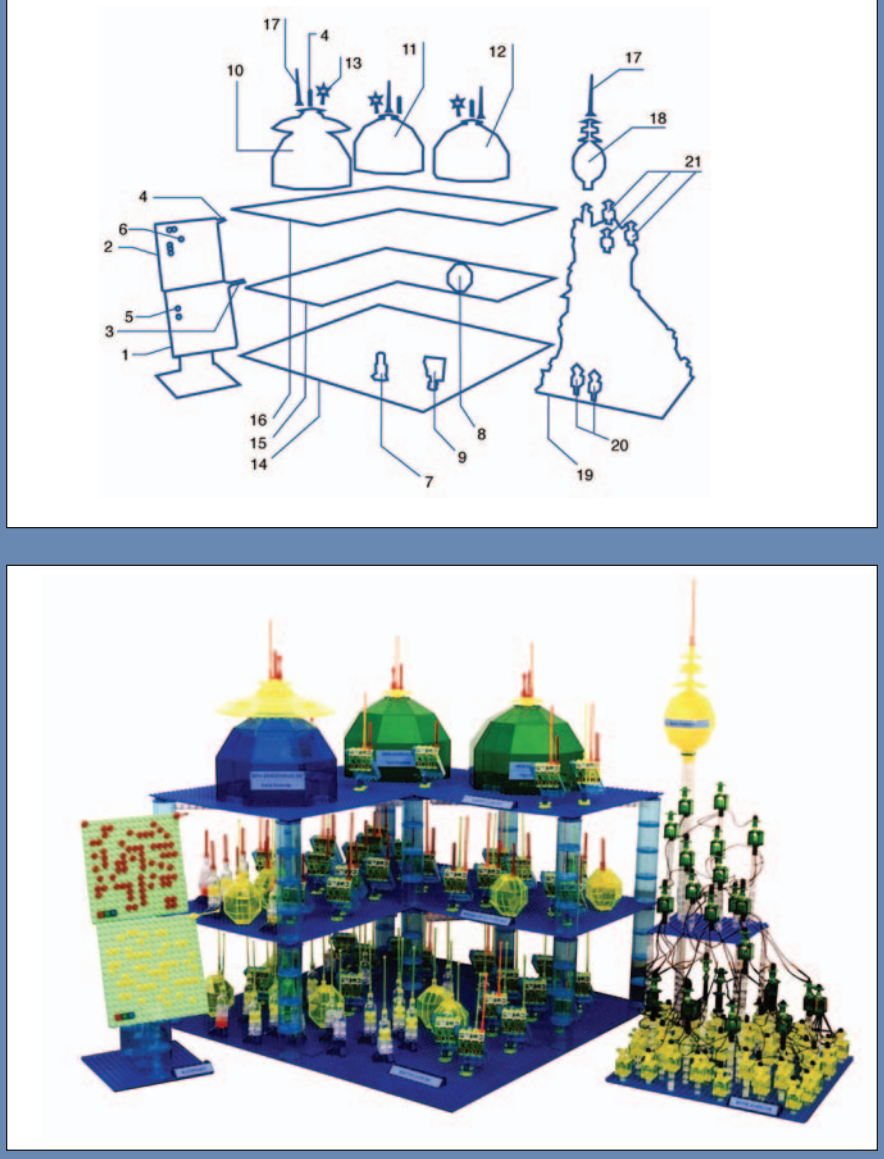

Figure 1: *The Ymir architecture underlying the Gandalf system has been reconstructed in physical LEGO blocks to demonstrate the power of functional modularity and modular divisibility.*

Figure 1 Legend

1. *Blackboard for receiving/dispatching messages containing simple data*
2. *Blackboard for receiving/dispatching messages containing complex, semantic data*
3. *Yellow antenna tuned to simple data*
4. *Red antenna tuned to complex data*
5. *A message containing simple data*
6. *A message containing complex data*
7. *A unimodal perceptual processor*
8. *A decider module*
9. *A multimodal perceptual processor*
10. *Meta-knowledge module*
11. *Dialog knowledge module*
12. *Domain knowledge module*
13. *Star-shaped antenna tuned to communication between knowledge modules (knowledge data blackboard, not shown)*
14. *Layer containing relatively high-speed processes*
15. *Layer containing medium-speed processes*
16. *Layer containing relatively slow processes*
17. *Orange antenna tuned to motor requests*
18. *Motor scheduler*
19. *Motor action library*
20. *Atomic motor instructions*
21. *Compound, complex motor instructions*

Psyclone

Unlike closed virtual worlds with predictable frame rates, interactive humanoids are best modeled as asynchronous, real-time, open loop systems, with unpredictable input streaming in real time. Psyclone™ is a message-based middleware that simplifies the design of such systems and their connection to input and output systems like vision, body tracking and graphics. Built around the concepts of modules and blackboards, it takes full advantage of the benefits of a message-based, publish-subscribe system. In publish-subscribe systems a module can register for a message type, and any time a message of that type is posted (by anyone in the system), the message will be delivered to the subscribed module. Among the most obvious benefits of this system is that the messages embody an

Figure 2a: *The Gandalf character interacting with a user, who is asking Gandalf to tell him about the planet Saturn. Gandalf's rich multimodal perception and multimodal output enables him to generate dialog behavior akin to that observed between two people. A special body-tracking system, including gloves, jacket and eye tracker, provide Gandalf with enough fine-grain perception to allow highly reactive behavior, in addition to deliberative actions.*

Figure 2b: *Gandalf is "massively interactive," reacting believably to anything from the user's subtle eye movements (e.g. a glance over to the solar system), to a full communicative act such as questions about the planets.*

explicit representation of each module's contextual behavior, and carry with it their state, so the blackboard provides a localized recording of all system events and system flow. Messages effectively implement APIs between the system's modules, specifying their interactions at the content level, through a unified protocol. Combined with blackboards, a publish-subscribe architecture is thus a powerful tool for incremental building and debugging of interactive systems. We take advantage of this fact with a web-based system inspector, which can display the state of modules while Psyclone is running. This facility has turned out to be invaluable for debugging, since the path of decisions, embodied in the message flow, can be inspected directly. Also, this tool works from a remote location, which can be very useful on systems where the machines are geographically disparate. The architecture of Psyclone is shown in Figure 3.

Comparing Psyclone to the well-known architecture CORBA, which shares some of its goals and features, Psyclone is created specifically for use in designing interactive AI systems, and thus has many features specific to systems requiring soft real time. In CORBA, an object makes a request for a service or for information, and this request is brokered by a central server, simulating an extended function call. By contrast, Psyclone uses blackboards: Modules post data to a central server, and that data is delivered to subscribed modules. Additionally, modules in Psyclone can retrieve old messages from the blackboards, through a simple query language. At a high level, then, CORBA is "pull" whereas Psyclone is both "pull" and "push." Psyclone also offers message time stamping and quality of service via prioritized scheduling, functionalities still missing in CORBA.

When a Psyclone module receives a message type to which it has subscribed, it may in turn post zero or more messages. Modules can also post messages at any time, independent of other message flow. To simplify development in Psyclone, the full set of modules and their attributes can be specified in an XML file called a psySpec. Modules that run on machines other than the Psyclone server can also be configured via this file.

At run-time, all data in the system travels in messages, via blackboards. A message is a convenient metadata wrapper around the message's content. The metadata includes the message's type, a globally unique ID (GUID), the language that the content is represented in, name of sender and time of posting, along with other timestamps. This metadata can be useful in making queries about the system's past behavior.

All messages in Psyclone have a type (most often assigned by the system designer and specified in the psySpec). For example, a face detector may post a piece of data of the type "Vision.Face," containing detailed facial data. There may be any number of different face detector modules, each of which posts that same data type, but with different content and emphasis. Message type names are represented as a tree with dot-delimitation, e.g. Vision.Face.Human and Vision.Face.Dog. One-to-one messaging between any two modules is done using unique message types.

In addition to standard XML and ASCII messages, Psyclone provides powerful facilities for publishing and subscribing to binary data streams, using Psyclone's version of blackboards, called "whiteboards", which natively support streaming media. In Psyclone, all modules and whiteboards have unique names; modules subscribe to message types from particular whiteboards as specified in the psySpec. They can also unregister and register dynamically for message types at run time. Through dynamic subscriptions, interaction between modules can thus be "rewired" on the fly, with any module able to alter its "connection" to other modules by registering or unregistering for the type(s) of data they produce.

It may be desirable to have the sensory I/O and cognition modules running in different languages and/or on different operating systems or hardware. Some components may only run on certain hardware architectures or configurations, as is often the case with open-source packages for speech recognition, speech synthesis and computer graphics. Another reason to run components separately is that one may want particular components to take advantage of specialized hardware. Yet another reason is cost: Since hardware is now cheaper than ever, access to multiple pieces of older yet perfectly usable hardware is becoming the norm. To allow communication between Psyclone and external modules we use network adapters called A.I.R. plugs. Each supported programming language, such as C++, Java and LISP, has its own plug. Plugs have a simple, open API with full messaging capabilities. The plugs free the developer from concerning themselves with sockets and other details of networking. Daemons facilitate starting and managing processes on remote machines: On startup, Psyclone tells the daemon running on each of the computers to start up all relevant executables on that machine, automatically sending over any configuration parameters for these executables specified in the psySpec.

The strategy of using separate executables or separate hardware for each component may seem foreign - even irrelevant - to some graphics professionals, many of for whom the goal is to get an entire system running on a single computer or game console. However, a strategy like CAIM, that supports modular separation of functionalities during development and allows for rearranging the components and their interaction in a highly dynamic way, is, to our knowledge, the most powerful methodology currently available for creating broad, interactive AI characters.

Once a modular design has been developed and tested using Psyclone, it can be more easily simplified and optimized to run correctly on a single processor. That said, Psyclone is also designed for long-term stability in deployed applications, and can be used for this purpose.

Mirage

We will now briefly describe how Psyclone was used with the Constructionist AI Methodology to produce an augmented-reality room inhabited by an embodied virtual agent, Mirage, developed by Thórisson and students at Columbia University [12].

To support interaction with the agent, the user puts on a pair of optical see-through glasses (Sony LDI-D100B) that reveal the location of the agent. Wearing the glasses, the user still sees the real world, but superimposed on it is a stereoscopic, ghost-like apparition of the Mirage agent, the interactive humanoid, whose behavior is generated by the system in real time. Using position and orientation tracking of the user's head and hand, Mirage [Figure 4] can remain stationary, or move around in the world, independently of the user, giving the impression that he occupies a physical location in space.

Mirage consists of eight main components: Two perception modules, an action/animation scheduling module, a speech recognition module, a speech synthesis module, a decision module and a news summary module [7]; a complete graphics system renders the embodied agent. All of these communicate through Psyclone.

A detailed 3D model of the entire room, including tables, chairs, etc., gives Mirage a certain amount of knowledge about his surrounding. All graphics are implemented using the Java3D API. The model of the room is scaled and oriented to match the real one; Mirage can thus "perceive" the real objects in the room, walk around obstacles and orient himself relative to the user. Mirage also has access to a database providing propositional information about the name, type, use and other attributes of individual objects.

To make the agent interactive and conversational, a multimodal input system was implemented that uses speech recognition and motion tracking of the user's right hand. The agent itself is capable of multimodal communication with the user via a speech synthesizer, body language and manual gesture. Users can point at any object in the room and ask Mirage what that object is. When asked about the news, Mirage will recite up-to-date news summaries. The agent is also aware of his own proximity and orientation to the user. When his name is spoken or the user comes within communicative distance, Mirage will turn to greet.

The role of perception modules is to create messages about changes in the environment, including recognition of user's speech, proximity of the user to the Mirage agent, etc. Those messages are then used by the Decision module to create sensible responses. Perception-1 is built into the augmented reality graphics system. It keeps track of agent's and user's position and orientation, noting changes of states such as "is the agent visible to the user," "is the agent looking at the user," etc. To track deictic gesture, a selection mechanism is triggered on any speech command, returning a time-stamped list of objects being pointed at [9]. Integrating speech with spatial perception included piping messages coming from the speech recognizer and Perception-1 into a third module, Perception-2. This step was achieved with relative ease, demonstrating the advantages of the modular Constructionist approach. The list of objects being pointed at is thus further processed by Perception-2, where they are combined with the speech to make a best guess at the meaning of the user's communicative act.

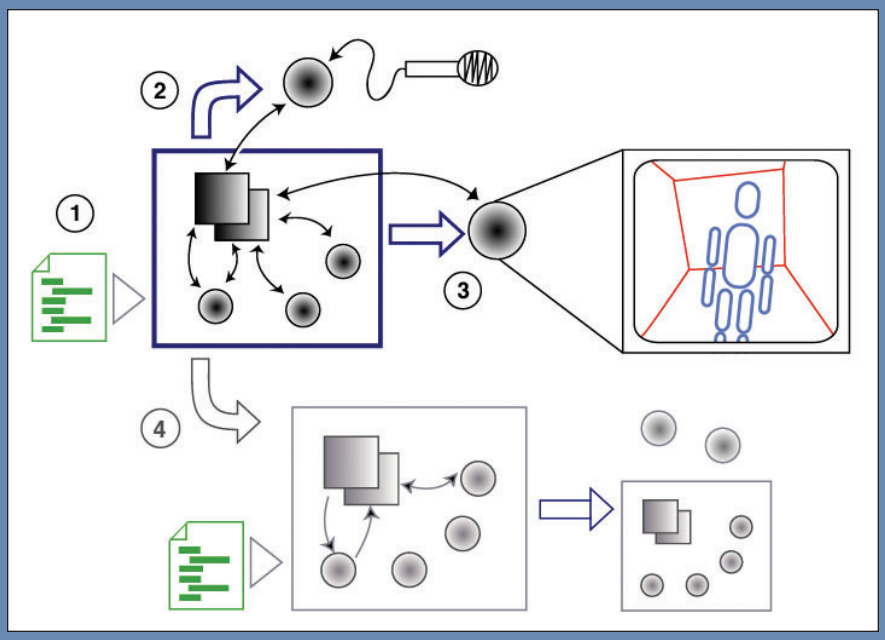

Figure 3: *When Psyclone starts up it reads in the psySpec (1), which specifies the setup of the system's modules. It then starts up any internal and external modules, e.g. speech recognition (2) and computer graphics (3), including those which run on external computers, and sets up all subscription mechanisms for these. Psyclone can further be set up to automatically start and configure other Psyclone servers (4), which in turn can start up yet more Psyclone servers, supporting a variety of configurations.*

The main reason for having two perception modules was to be able to use the built-in geometric computation methods of the graphics subsystem, and its preexisting object selection and intersection mechanisms, without having to put all perception into the graphics system. Theoretically cleaner solutions are undoubtedly possible, but they are likely to have been computationally less efficient.

The decision module interprets all messages from perception, and forms a response action for the agent. That action is then sent to the action scheduling module, inside the graphics system, which coordinates the character's animation and speech synthesis. When the decision module receives contradicting inputs or is unable to settle onto a single top-level decision, it can request that the avatar ask clarification questions, such as "Please repeat, I did not understand."

Although eight modules makes for a relatively coarse-grained system (compared to e.g. the Gandalf system), the development team, who had never used the Constructionist approach before, realized its benefits very early in the project. The advantages of defining the messages and their content at the same time as module functionality, and being able to test a skeleton of the system with "found" components in place, sped up the identification of potential problems and bottlenecks, and made it easier to modify the system's behavior during its development.

The total development time for Mirage was only an estimated two mind-months ("man"-months), over a period of nine weeks - well under anyone's prior expectations. This result is comparable or better than that of others using blackboard-like architectures, e.g. Maxwell et al. [6], who constructed a highly sophisticated robotic system with 10 full-time students, over a period of eight weeks.

Future Work

We will continue improving CAIM and use it to guide further development of the Psyclone system. Future enhancements of Psyclone include increased support of message content semantics, which will become increasingly

Figure 4: *Mirage, seen through the LCD screen of the "magic glasses" that reveals the agent to the user. (Image has been digitally enhanced for clarity.)*

important as more third-party modules become available and manual maintenance of semantics becomes impractical. Other planned enhancements include extending Psyclone to support a large number of modules (thousands or tens of thousands in mixed configurations, including single/multiple modules per executable), running on a large number of computers.

We will be looking specifically at the connection between Psyclone and graphical environments, refining it through its use for various projects, mostly focused on interactive graphical characters. We also plan on proposing specifications for virtual world vision representations and implementations, an area that has to be addressed for this work to progress at a faster pace.

Finally, we are in the process of setting up a research forum, in collaboration with other researchers, to develop this work further and build a repository of AI and graphics software that can be reused more easily through application of Constructionist principles. We invite anyone with interest to join in this effort. See **www.MINDMAKERS.org**.

Acknowledgments

The authors would like to thank the Mirage project members, Steve Feiner and Patricia Galvis-Assmus.

References

1. Badler, N., M. S. Palmer, R. Bindiganavale. "Animation Control for Real-Time Virtual Humans," *Communications of the ACM*, August, Vol. 42(8), pp. 65-73, 1999.
2. Bryson, J. and K. R. Thórisson. "Bats, Dragons & Evil Knights: A Three-Layer Design Approach to Character-Based Creative Play," *Virtual Reality, Special Issue on Intelligent Virtual Agents*, 5, Heidelberg: Springer-Verlag, pp 57-71, 2000.
3. CMLabs. *The Psyclone Manual,* **http://www.cmlabs.com/psyclone/,** 2004.
4. Laird, J. "Research in Human-Level A.I. Using Computer Games," *Communications of the ACM*, January, Vol. 45(1), pp. 32-35, 2002.
5. Laird, J. E. and M. van Lent. "Human-Level AI's Killer Application: Interactive Computer Games," *A.I. Magazine*, summer, pp. 15-25, 2001.
6. Maxwell, B. A., L. A. Meeden, N. S. Addo, P. Dickson, N. Fairfield, N. Johnson, E. G. Jones, S. Kim, P. Malla, M. Murphy, B. Rutter and E. Silk. "REAPER: A Reflexive Architecture for Perceptive Agents," *A.I. Magazine*, spring, pp. 53-66, 2001.
7. McKeown, K.R., R. Barzilay, D. Evans, V. Hatzivassiloglou, J. L. Klavans, A. Nenkova, C. Sable, B. Schiffman and S. Sigelman. "Tracking and Summarizing News on a Daily Basis with Columbia's Newsblaster," Human Language Technology '02 (*HLT* '02), San Diego, CA, 2002.
8. Minsky, M. *The Society of Mind*, New York: Simon and Schuster, 1986.
9. Olwal, A., H. Benko, S. Feiner. "SenseShapes: Using Statistical Geometry for Object Selection in a Multimodal Augmented Reality System," to appear in *Proceedings of The Second International Symposium on Mixed and Augmented Reality 2003 (ISMAR 03),* October 7-10, 2003, Tokyo, Japan, 2003.
10. Selfridge, O. "Pandemonium: A Paradigm for Learning, Proceedings of Symposium on the Mechanization of Thought Processes," pp. 511-529, 1959.
11. Terzopoulos, D. "Artificial Life for Computer Graphics," *Communications of the ACM*, August, Vol. 42(8), pp. 33-42, 1999.
12. Thórisson, K. R., H. Benko, A. Arnold, D. Abramov, A. Vaseekaran. "A Constructionist Methodology for Interactive Intelligences," to be published in *A.I. Magazine*, 2004.
13. Thórisson, K. R. "A Mind Model for Multimodal Communicative Creatures and Humanoids," *International Journal of Applied Artificial Intelligence*, 13 (4-5), pp. 449-486, 1999.
14. Thórisson, K. R. "Layered Modular Action Control for Communicative Humanoids," *Computer Animation '97*, Geneva, Switzerland, June 5-6, pp. 134-143, 1997.
15. Thórisson, K. R. *Communicative Humanoids: A Computational Model of Psychosocial Dialogue Skills,* PhD Thesis, MIT Media Laboratory, 1996.
16. Newell, A. *Unified Theories of Cognition,* Cambridge, MA: Harvard University Press, 1990.

About the Authors

Communicative Machines Laboratories (CMLabs) specializes in communication between humans and computers, using solutions from artificial intelligence to create interactive experiences and enable machines to perceive and act in a variety of environments. Based on research spanning more than a decade, the Psyclone platform has been in development by CMLabs for over three years, with the express goal of creating a new foundation for simulating complex phenomena. Psyclone is being used in several advanced development projects in Europe and the U.S. in areas including robotics, computer vision, computer graphics and animation.

CMLabs Europe
67 Giles Street, Suite 28
Edinburgh EH6 6DD
Scotland, UK
CMLabs US:
172 Park St., Suite 6
New Haven, CT 06511
U.S.A.
Email: **info@cmlabs.com**