Cognitive Modeling for Computer Games and Animation

John Funge
john.funge@intel.com
Microcomputer Research Lab,
Intel Corporation

Introduction

Modeling for computer animation addresses the challenge of automating a variety of difficult animation tasks. An early milestone was the combination of geometric models and inverse kinematics to simplify keyframing. Physical models for animating particles, rigid bodies, deformable solids, and fluids offer copious quantities of realistic motion through dynamic simulation. Biomechanical modeling employs simulated physics to automate the realistic animation of living things motivated by internal muscle actuators. Research in behavioral modeling is making progress towards self-animating characters that react appropriately to perceived environmental stimuli.

In our research, we explore *cognitive modeling* for computer animation and computer games. Cognitive models go beyond behavioral models in that they govern what a character knows, how that knowledge is acquired, and how it can be used to plan actions. Cognitive models are applicable to directing the new breed of highly autonomous, quasi-intelligent characters that are beginning to find use in animation and game production. Moreover, cognitive models can play subsidiary roles in controlling cinematography and lighting.

We decompose cognitive modeling into two related subtasks: *domain specification* and *behavior specification*. Domain specification involves giving a character knowledge about its world and how it can change. Behavior specification involves directing the character to behave in a desired way within its world. Like other advanced modeling tasks, both of these steps can be fraught with difficulty unless animators are given the right tools for the job. To this end, we develop a cognitive modeling language, CML.

CML rests on solid theoretical foundations laid by artificial intelligence (AI) researchers. This high-level language provides an intuitive way to give characters, and also cameras and lights, knowledge about their domain in terms of actions, their preconditions and their effects. We can also endow characters with a certain amount of "commonsense" within their domain and we can even leave out tiresome details from the specification of their behavior. The missing details are automatically filled in at run-time by a reasoning engine integral to the character which decides what must be done to achieve the specified behavior.

Traditional AI style planning certainly falls under the broad umbrella of this description, but the distinguishing features of CML are the intuitive way domain knowledge can be specified and how it affords an animator familiar control structures to focus the power of the reasoning engine. This forms an important middle ground between regular logic programming (as represented by Prolog) and traditional imperative programming (as typified by C). Moreover, this middle ground turns out to be crucial for cognitive modeling in animation and computer games. In one-off animation production, reducing development time is, within reason, more important than fast execution. The animator may therefore choose to rely more heavily on the reasoning engine. When run-time efficiency is also important our approach lends itself to an incremental style of development. We can quickly create a working prototype. If this prototype is too slow, it may be refined by including more and more detailed knowledge to narrow the focus of the reasoning engine.

Specifying behavior in CML capitalizes on our way of representing knowledge to include a novel approach to high-level control. It is based on the theory of *complex actions* from the situation calculus [4]. Any primitive action is also a complex action, and other complex actions can be built up using various control structures. As a familiar artifice to aid memorization, the control structure syntax of CML is deliberately chosen to resembles that of C.

Although the syntax may be similar to a conventional programming language, in terms of functionally CML is a strict superset. In particular, a behavior outline can be nondeterministic. By this, we do not mean that the behavior is random, but that we can cover multiple possibilities in one instruction. As we shall explain, this added freedom allows many behaviors to be specified more naturally, more simply, more succinctly and at a much higher-level than would otherwise be possible. The user can design characters based on behavior outlines, or "sketch plans". Using its background knowledge, the character can decide for itself how to fill in the necessary missing details.

The complete list of operators for defining complex actions is defined recursively and the mathematical definitions are given in [4]. The corresponding CML syntax is given in the CML documentation. The documentation and the Java executable can be found at [2].

We have used CML in a number of research projects related to computer animation and games. Some of the images from the corresponding animations can be found at

[3]. We will give a brief description of these case studies below, some further details can be found in [1].

Cinematography

We have positioned our work as dealing with cognitive modeling. At first, it might seem strange to be advocating building a cognitive model for a camera. We soon realize, however, that it is the knowledge of the cameraperson, and the director, who control the camera that we want to capture with our cognitive model. In effect, we want to treat all the components of a scene, be they lights, cameras, or characters as "actors". Moreover, CML is ideally suited to realizing this approach and can reduce long and complicated Finite State Machine (FSM) controllers to a few lines of code (see [1]).

Prehistoric world †

In our prehistoric world we have a Tyrannosauras Rex (T-Rex) and some Velociprators (Raptors). The motion is generated by some simplified physics and a lot of inverse kinematics. The main non-aesthetic advantage the system has is that it is real-time on a Pentium II with an Evans and Sutherland RealImage 3D Graphics card. In only a few lines of CML code we were able to specify a territorial behavior for the T-Rex that caused it to "herd" the Raptors out of its territory.

Undersea world †

In our undersea world we bring to life some mythical creatures, namely "merpeople". The undersea world is physics-based. The high-level intentions of a merperson get filtered down into detailed muscle actions which cause reaction forces on the virtual water. A low-level reactive behavior system described in [5], provides a buffer between the reasoning engine and the environment. Thus at the higher level we need only consider actions such as "go left", "go to a specific position", etc. and the reactive system will take care of translating these commands down into the required detailed muscle actions. We used CML to create some engaging pursuit and evasion behaviors of a larger and faster shark chasing the more intelligent merpeople. In open water a shark can easily catch the merpeople. When we add some rocks for the merpeople to hide behind they can use their intelligence to often out smart the sharks and escape.

Conclusion

In summary, CML always gives us an intuitive way to give a character knowledge about its world in terms of actions, their preconditions and their effects. When we have a high-level description of the ultimate effect of the behavior we want from a character, then CML gives us a way to automatically search for suitable action sequences. When we have a specific action sequence in mind, there may be no point to have CML search for one. In this case, we can use CML more like a regular programming language, to express precisely how we want the character to behave. We can even use a combination of these two extremes, and the whole gamut inbetween, to build different parts of one cognitive model. It is this combination of convenience and automation that makes CML such a potentially important tool in the arsenal of tomorrow's animators and game developers.

Acknowledgements

Note that the sections marked with a † refer to work done in collaboration with Xiaoyuan Tu. We would also like to thank Eugene Fiume for originally suggesting the application of CML to cinematography, Demetri Terzopoulos for originally suggesting creating a merperson, and Angel Studios for developing the low-level dinosaur API.

References

[1] J. Funge. *Making Them Behave: Cognitive Models for Computer Animation*. PhD thesis, Department of Computer Science, University of Toronto, Toronto, Canada, 1998.

[2] John Funge, `www.cs.toronto.edu/~funge/cml`. *CML Compiler*, 1998.

[3] John Funge, `www.cs.toronto.edu/~funge/images.html`. *Selected Frames from various Animations*, 1998.

[4] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–84, 1997. Special issue on Reasoning about Action and Change.

[5] X. Tu. *Artificial Animals for Computer Animation: Biomechanics, Locomotion, Perception, and Behavior*. PhD thesis, Department of Computer Science, University of Toronto, Toronto, Canada, January 1996.