CS585

Term - Project Report

Evolving Legged Locomotion in Virtual Creatures

Daniel Nikovski

Department of Computer Science
Southern Illinois University at Carbondale

1 Overview of the Project

The main objective of the project was to employ physically-based modeling and evolutionary computation techniques for the creation of autonomous virtual creatures that could be used in virtual reality systems. To this end an interactive animation system was developed that simulated the locomotion and evolution of particle-based virtual creatures. A mass-spring hexapod was designed and its motor program evolved using genetic algorithms (GA). Finally, a photorealistic animation clip was produced that demonstrated the locomotion of the hexapod under the control of an evolved motor program.

2 Completed Work

2.1 Interactive animation system

The interactive animation system provides a graphical user interface (GUI) to a simulated world inhabited by physically-modeled virtual creatures. The system is modular and parametrized as much as possible, so that it can be used to quickly model different creatures and environmental conditions. The main parts of the system are:

Dynamics module. This module contains a description of the structure of the virtual creature in the form of particles connected with springs. Different forces and environmental conditions are implemented to correctly reflect the dynamics of the simulated world. The equations of motion of the creature are integrated to produce a coordinate description of the creature for successive moments in time. The module uses extensively the Dynamics Toolkit from the BioBlock modeling system.

Control Module. The control module changes over the course of time the resting lengths of the springs that define the virtual creature in the Dynamics Module, so that the creature can exibit interesting behaviors. This module is necessary for the simulation of objects driven by active forces - for example walking, swimming or flying creatures. Currently the controller is implemented as a motor program - a sequence of instructions that change the resting lengths of the springs in the legs of the hexapod.

Evolutionary Module. In its turn, this module changes the Control Module over the course of the evolution of many generations of virtual creatures. The controller should be suitably parametrized and encoded so that the Evolutionary Module can perform optimization in the space of the parameters in search for the best motor program.

Visualization Module. The 3D coordinate description that results from the integration of the equations of motion is transformed through the viewing pipeline into a 2D image.

The module makes use of PHIGS in the X Window environment.

GUI. The GUI module implements the dialog with the user. The XView toolkit and DevGuide were used to create the GUI module.

It should be noted that the animation system is not precisely an end-user system. Providing a language for specifying the structure of the creature and the control system was clearly out of the scope of the project. Customizing the system is done through modifying the source code, but nevertheless is sufficiently flexible.

2.2 Design of a virtual creature

A hexapod consisting of 14 particles and 34 springs was designed within the animation system. Special care was taken to make a structurally stable physical system by reinforcing the design with 6 more springs placed appropriately. The spring and damping coefficients were established with interactive experiments. It turned out that by far not all choices of spring and damping coefficients result in a numerically stable system - the Euler integration scheme employed in the Dynamics Toolkit often produced divergent behavior.

2.3 Implementation of force functions

Since the concrete goal of the experiments was to simulate legged locomotion, the appropriate friction forces had to be supplemented. Due to the open design of the Dynamics Toolkit, this task was straightforward after the underlying physics laws were identified.

2.4 Experiments with different control systems

The implementation of a control system for a creature as complex as the designed hexapod is not easy, to say the least. The motion of the system is governed by 84 first order ordinary differential equations (ODE) and thus, at least in principle, it is possible to design a classical controller that will make the hexapod follow desired trajectories. However, besides being not very practical, this approach will fail unless trajectories for all particles are specified. If this was the case, we would not bother to use physical modeling in the first place - simple kinematic animation would suffice.

A very different approach was taken. A roug motor program was designed that specified approximately the resting lengths that would produce the backward wave walking pattern (i.e., only one leg moves at a time and all legs on one side move before any on the other,

starting with the hindmost leg). As expected, the creature was able to make a couple of walking cycles before falling down, but its gait was far from stable. It was reasonable to expect that relatively small variations of the resting lengths might produce stable gait, but finding the exact values of the corrections seemed intractable analytically.

It was realized then that the problem can be formulated as finding the value for the corrections that would maximize the stability of the gait, which is an optimization problem. Following the suggestions of McKenna and Zeltzer [1] and the work of Beer and Galagher [4], Sims [3] and Gruau [5], it was decided that evolutionary computation (EC) techniques might discover the desired corrections.

2.5 Evolving motor programs

The resting lengths of the spings in the legs of the hexapod can be encoded in a chromosome in many ways. There are a total of 20 such springs for the six legs of the creature. The motor program has 13 time slots that each specify a set of resting lengths. (The time interval between the slots is held constant at 5 time units.) This makes a total of 260 resting lengths, and if 4 bits are allocated for each value, the length of the chromosome would be 1040 bits.

To make the representation more compact, the symmetry of the animal was taken into consideration (the four front and rear legs are symmetric and the two middle legs are symmetric too). Six leg positions were defined - up-rear, up-middle, up-front, down-rear, down-middle, down-front and the motor program was encoded in terms of these six positions following the design principles used by Brooks [2]. The chromosome then encoded the resting lengths for these 6 positions of both types of legs. The resulting chromosome was 168 bits long - nearly 6 times more compact. The mapping of the encodings of the corrections to real values was chosen in such a way as to modify the original lengths by up to $\pm 50\%$ of their respective original lengths.

The fitness function of the GA was defined to be the distance traveled by the creature for a fixed time period (1000 units or about 15 gait cycles). Thus a gait that was not stable and led to falling down was penalized naturally. Only the distance traveled along the symmetry axis of the hexapod was taken into consideration. A sample of 30 random motor programs was used to estimate the statistical parameters of the fitness function - $\mu = 1064$, $\sigma = 1034$. The original motor program had a fitness of 1325, which was probably result of the attempts to "tweak" the resting lengths somehow before GA optimization was applied.

A number of simulations were run to optimize the fitness function. The best value obtained was 4070, which is definitely much better than the original motor program. However, it still lies at about three standard deviations from the average value, which gives it a probablity of about 0.5% to arise randomly if the distribution of the fitness function is normal.

Over 100 randomly generated individuals the probability that a value that high will arise is actually substantial - 39.5%. Over 1000 individuals this probability is already 99.4there is no statistical significance to the hypotheis that the value 4070 is due to the operation of the GA rather than to brute force random search. On the other hand, the probability distribution of the fitness function is likely to differ from the normal one exactly in the tail parts - very large values for the distance are physically not possible to be obtained in a fixed time. Thus it is difficult to estimate how good the result of the GA is in comparison to random search; whatever the answer, however, the best evolved motor program performs much better than the one without corrections.

2.6 Rendering the animation

The last stage of the project was the creation of a photorealistic animation of the locomotion of the hexapod. The original plan was to use the work of the REND group, but it was not completed and debugged on time to be used. Instead, version 3.1 of the *SIPP* public-domain library running under UNIX was employed to render the animation sequence. *SIPP* has very clean and efficient design and gives very good control over the rendering process. In addition, it has a number of shading algorithms (Gouraud, Phong, flat and wire frame) which permits quick previews of the animation in wire frame mode and photorealistic rendering later.

One gait cycle was animated (70 units of time). The scene was rendered for every second time unit, which resulted in smooth animation and image files of reasonable size. The frames were 256 by 256 pixels, true color. The images rendered with *SIPP* were transformed to TGA format and combined in an Autodesk Animator FLC file through a public-domain utility running under MS-DOS (*DTA* by David Mason). The file containing the animation has a size of approximately 1MB. The rendering process takes less than 3 hours on SunSparc Classic, while the production of the final animation clip takes about 3 minutes on a 486 PC.

The animation file is best viewed with the *xanim* public domain utility by Mark Podlipec.

3 Discussion and Future work

The plan set out in the Proposal for the project was realized almost completely. The stage that was not covered was concerned with the evolution of the structure of the creatures, i.e, full-blown evolution. However, there are strong reasons to believe that this will not be possible unless other intermediate tasks are completed.

A necessary improvement to the Dynamics Toolkit is to change the Euler ODE integrator with a Runge-Kutta one. The current design of the hexapod leads to a numerically stable

system of ODE under the Euler scheme. Variations from this design, however, are most likely not to be stable, which makes evolution impossible. Besides, as Karl Sims points out [3], the evolution will actually take advantage of numerical inaccuracies and produce best solutions that are not actually solutions.

Another major source of technical problems is PHIGS itself. No less than 30% of the programming time was spent in battling the quirks and glitches of PHIGS. It will be wise to consider other interactive 3D packages (in particular OpenGL) as an alternative.

Apart from these "technical" issues, the work on the project can continue in the following directions:

Improving the GA search. Techniques like Gray codes and Dynamic Parameter Encoding can be used to improve the effectiveness of the GA search [4].

Collision detection and response. Currently there is no collision detection and response in the dynamics system. The COLD group was no more ready with their work than the REND group when I contacted them, and besides, they have implemented only detection without response. This was not a severe problem, though, because the corrections to the resting lengths were kept within a safe range.

Enhancing the fitness function. The fitness function used was quite appropriate, with a single exception. The creatures discovered that if they made a plunge just before the end of the simulation time, they could gain some more distance (like sprint runners do). The parent that discovered this trick passed it to its generation and soon after that everybody was doing plunges. This is not what we want, definitely. The fitness function has to amended somehow.

Evolving the whole creature. This is the last stage from the original proposal and can be done first by changing the masses of the particles and later on changing structures. The use of a dependable ODE solver is critical to this end.

New control mechanisms. Control techniques other than motor programs can be tried - neural networks, classifier systems, finite automata, fuzzy systems. Most of the experiments reported in the literature use neural networks, though there doesn't seem to be a very convincing reason why neural nets should perform better than any of the other methods mentioned above. Evolutionary Programming techniques can be used since they can evolve solutions of any kind - not only ones encoded in chromosomes.

Employing parallel machines. All EC methods use considerable computing resources and more complex problems will become quickly intractable on serial machines. The EC methods can be naturally parallelized and parallel computers can be easily employed in simulated evolution.

4 Conclusion

The major result from the project is that realistic animation of nontrivial objects can be obtained with moderate effort on the part of the animator if physically-based modeling and evolutionary computation are used. The most difficult task is the design of a controller for the simulated object, and this is the place where EC methods or other optimization schemata can help a lot.

It seems that a very good way to reduce the work of the animator is to give enough "intelligence" to the animated creature. Behavior-based animation can benefit a lot from techniques employed in classical AI as well as current work in reactive systems, autonomous agents and simulation of adaptive behavior.

References

[1] McKenna, Michael and David Zeltzer (1990). Dynamic Simulation of Autonomous Legged Locomotion. *Computer Graphics*, 24(4):29-38.

[2] Brooks, R. A. (1989). A Robot that Walks: Emergent Behaviors from a Carefully Evolved Network. *Neural Computation* 1:253-262.

[3] Sims, Karl (1994). Evolving Virtual Creatures,*Computer Graphics* 28(4):15-22.

[4] Beer, R.D. and John C. Galagher (1992). Evolving Dynamical Neural Networks for Adaptive Behavior. *Adaptive Behavior* 1:91-122.

[5] Gruau, Frédéric (1994). Automatic Definition of Sub-neural Networks. *Preliminary copy released on the Internet.*