Fast simulation of animal locomotion: lamprey swimming

Matthew Beauregard, Paul J. Kennedy, and John Debenham

Faculty of IT, University of Technology, Sydney, PO Box 123, Broadway, NSW 2007, AUSTRALIA,
paulk@it.uts.edu.au

Abstract. Biologically realistic computer simulation of vertebrate locomotion is an interesting and challenging problem with applications in computer graphics and robotics. One current approach simulates a relatively simple vertebrate, the lamprey, using recurrent neural networks for the spine and a physical model for the body. The model is realized as a system of differential equations. The drawback with this approach is the slow speed of simulation. This paper describes two approaches to speeding up simulation of lamprey locomotion without sacrificing too much biological realism: (i) use of superior numerical integration algorithms and (ii) simplifications to the neural architecture of the lamprey.

1 Introduction

Vertebrate locomotion – walking, swimming, crawling, hopping – is a complex process that is difficult to imitate in simulated environments. Arms, legs, and spinal columns have many degrees of freedom that must be controlled in a coordinated way for stable locomotion to occur. This complexity has limited the use of biologically realistic locomotion in computer graphics and robotics, despite its considerable advantages.

Some characteristics common to locomotion in all vertebrates suggest a fruitful approach. The key to vertebrate locomotion is not the brain but the spinal cord, which contains all the structures necessary for coordinated movement. All types of motion, whether by legs or slithering muscles, on land or through water, are driven by oscillations between the left and right sides of small segments in the spinal cord. This suggests that an approach to the problem is to study a simple vertebrate and design a simulation that successfully imitates it. The findings will illuminate locomotion in higher vertebrates.

Let us introduce that simplest vertebrate. The lamprey is a jawless eel-shaped fish. Primitive in an evolutionary sense, its major distinguishing feature is a large rounded sucker surrounding the mouth [1]. Its spinal cord is a continuous column of neurons made up of around 100 clusters. Each cluster projects motoneurons to the surrounding muscles [2]. Lampreys swim by propagating a wave along the body from head to tail by phased muscular contraction. Normally the wavelength of this traveling wave is constant and approximately corresponds to body length; its frequency determines the speed of swimming.

The lamprey has been studied thoroughly over several decades. See [3] for a clear introduction to some of the modeling and other papers in the same volume (e.g. [4]) for further details. A variety of simulations has been implemented (see Section 2) but in the pursuit of absolute faithfulness to biology they run very slowly, which is a hindrance to their practical use in computer graphics.

In this paper we develop an understanding of the neural structure, demonstrate an implementation of swimming, and introduce a simplification resulting in reduced execution time while retaining biological realism. Sections 2 and 3 describe our model and the implementation and numerical strategies for increasing simulation speed. In section 4 we describe typical behavior of the model and, in section 5, give results for increasing simulation speed by simplifying the neural structure. Finally, section 6 gives a conclusion.

2 Model

Ijspeert [2] groups neural models of the lamprey into three classes: biophysical, connectionist and mathematical. Biophysical models most closely replicate the biological systems of the lamprey. Their main intent is to investigate whether enough is understood of lamprey neurobiology to produce models whose results agree with physiological observations. Connectionist models are less realistic and seek to capture only the main feature of biological neurons: a changing frequency of action potential spikes according to input. The main interest here is in connections between neurons. Mathematical models are more abstracted again and view the neural controller as a chain of oscillators, the focus being on examining the couplings between them. Connectionist models are very similar to dynamical recurrent neural networks. They compute the mean firing rate of biological neurons depending on input and time constants. They can be discrete in time or continuous using *leaky integrators*. Ekeberg presents a sophisticated connectionist model that formed the basis for our work [5] as well as that of [2].

The model presented here simulates both the lamprey's neural activity and the results of that activity when applied to a physical body in water. In contrast to previous work, the neural and physical aspects of simulation are combined into a single model, rather than two separate but interacting models.

2.1 Neural model

While biologically the spinal cord is a continuous column of neurons without clear boundaries, it can be considered as roughly 100 discrete but interconnected oscillators (or segmental networks). The combined assembly is known as a central pattern generator (CPG). The main types of neuron involved in the process are: motoneurons (MN) projecting to muscles, excitatory interneurons (EIN) projecting to ipsilateral neurons (ie. those on the same side of the segment), lateral inhibitory interneurons (LIN) projecting to ipsilateral neurons, contralateral inhibitory interneurons (CIN) projecting to contralateral

neurons (ie. the other side of the segment) and excitatory brain stem (BS) that project from the brain. The controller consists of 100 segmental networks. Each model neuron represents a population of functionally similar neurons. Actual connections between segments are not well known. Ekeberg chose a simplified, symmetric coupling (except for connections from the CINs which are longer tailward). Parameters for both inter– and intrasegmental connections and extents are given in Table 1. In order to limit output from neurons and to compensate neurons in segments near ends of the body (and have fewer intersegmental inputs), synaptic weights are scaled by dividing by the number of input segments.

Ekeberg [5] advocated supplying extra excitation to the first few segments of the spinal column to help generate a phase–lagged oscillation down the spine. However, we found in our simulations that this is not necessary to generate phase–lagged oscillation and actually impaired the speed of the lamprey. Arrangement of segmental coupling is sufficient to cause a traveling wave towards the tail. Accordingly, extra excitation was not applied.

Table 1. Neural connection configuration. From [2] with additions from [5] and separately–controllable left– and right–side excitation. Negative weights indicate inhibitory connections. Extents of connections to neighbor segments are given in brackets (headward and tailward, respectively).

To ↓	From: EIN_L	CIN_L	LIN_L	EIN_R	CIN_R	LIN_R	BS_L	BS_R
EIN_L	0.4 [2, 2]	-	-	-	-2 [1, 10]	-	2	0
CIN_L	3 [2, 2]	-	-1 [5, 5]	-	-2 [1, 10]	-	7	0
LIN_L	13 [5, 5]	-	-	-	-1 [1, 10]	-	5	0
MN_L	1 [5, 5]	-	-	-	-2 [5, 5]	-	5	0
EIN_R	-	-2 [1, 10]	-	0.4 [2, 2]	-	-	0	2
CIN_R	-	-2 [1, 10]	-	3 [2, 2]	-	-1 [5, 5]	0	7
LIN_R	-	-1 [1, 10]	-	13 [5, 5]	-	-	0	5
MN_R	-	-2 [5, 5]	-	1 [5, 5]	-	-	0	5

Each neuron is modeled as a leaky integrator with a saturating transfer function. u is the mean firing frequency of the population of neurons:

$$\begin{aligned}
\dot{\xi}_+ &= \frac{1}{\tau_D}\left(\sum_{i\in\psi_+} u_i w_i - \xi_+\right) \\
\dot{\xi}_- &= \frac{1}{\tau_D}\left(\sum_{i\in\psi_-} u_i w_i - \xi_-\right) \\
\dot{\vartheta} &= \frac{1}{\tau_A}\left(u - \vartheta\right) \\
u &= 1 - e^{(\Theta-\xi_+)\Gamma} - \xi_- - \mu\vartheta \text{ if positive} \\
&\quad 0, \text{ otherwise,}
\end{aligned} \tag{1}$$

where ξ_+ and ξ_- are the delayed 'reactions' to excitatory and inhibitory input and ϑ represents the frequency adaptation (decrease in firing rate over time given a constant input) observed in some real neurons. w are the synaptic weights of excitatory and inhibitory presynaptic neuron groups ψ_+ and ψ_-, τ_D is the time constant of the dendritic sums, τ_A the time constant of frequency adaptation, μ a frequency adaptation constant, Θ the threshold and Γ the gain. The parameters for the constants given in Table 2 are hand-tuned to produce output that matches physiological observations.

Table 2. Neuron parameters. From [5]. See text for an explanation of symbols.

Neuron type	Θ	Γ	τ_D	μ	τ_A
EIN	-0.2	1.8	30 ms	0.3	400 ms
CIN	0.5	1.0	20 ms	0.3	200 ms
LIN	8.0	0.5	50 ms	0.0	-
MN	0.1	0.3	20 ms	0.0	-

2.2 Physical model

We model the body similarly to [5] and [2]. It is represented by ten links (so ten neural segments act on one body segment) and nine joints between them with one degree of freedom. Each link is modeled as a right elliptic cylinder with the major axes of the ellipses aligned vertically. All links have length l of 30 mm and are 30 mm high. Their width is a maximum of 20 mm, decreasing towards the tail. Muscles appear on both sides of the body, attached to the centers of each segment. They are modeled with a spring–and–damper arrangement, where the force exerted by the muscle is set using the spring constant. Thus the local body curvature varies linearly with muscle length. Body and neural network are linked by having motoneuron excitation drive the muscular spring constants. For the purposes of modeling, the body is represented in two dimensions as rectangles with joints at the midpoints of their sides. The position of a link i can be described by (x_i, y_i, φ_i), where x_i and y_i are the coordinates of the rectangle centre and φ_i is the angle of a line through the centre and the joint with respect to the x–axis (see Fig. 1). This simulation strategy gives a maximal representation of the body position, so constraint forces are used to keep links together. Physical parameter values of the links are the same as those of [2].

Body movement is a result of three forces: torques T generated by the muscles, forces W_i from the water and constraint forces F_i and F_{i-1}. These forces determine the acceleration of the links according to Newton's law of motion. Change in position for links $i \in \{1, \ldots, N\}$ is determined by integration.

$$m_i \ddot{x}_i = W_{i,x} + F_{i,x} - F_{i-1,x}$$
$$m_i \ddot{y}_i = W_{i,y} + F_{i,y} - F_{i-1,y}$$

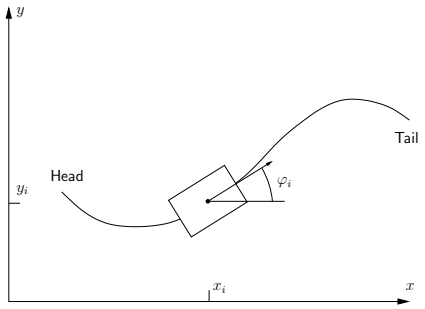

Fig. 1. Co-ordinates describing the position of a link. From [5].

$$I_i\ddot{\varphi}_i = T_i - T_{i-1} - (F_{i-1,x} + F_{i,x})\,\frac{l_i}{2}\sin\varphi_i + (F_{i-1,y} + F_{i,y})\,\frac{l_i}{2}\cos\varphi_i \tag{2}$$

Muscles are modeled as springs directly connected to the sides of the links. In an adaptation of Hooke's law, the force exerted by each spring on its associated joint is determined not only by the local curvature of the body but also linearly by the output of the motoneurons in the associated segments. As in [5], torque is defined as

$$T_i = \alpha\,(M_L - M_R) + \beta\,(M_L + M_R + \gamma)\,(\varphi_{i+1} - \varphi_i) + \delta\,(\dot{\varphi}_{i+1} - \dot{\varphi}_i)$$

where M_L and M_R are left and right motoneuron activity and the parameters α (=3 N mm), β (=0.3 N mm), γ (=10) and δ (=30 N mm ms) are set as in [5].

Speed of motion through water in our case is sufficiently high that we only account for inertial water force which is proportional to the square of the speed:

$$W = \rho v^2 \frac{A}{2} C$$

where ρ is the fluid density, v object speed, A area parallel to movement and C drag coefficient. The abbreviation $\lambda = \rho\frac{A}{2}C$ is made in [5], together with the simplification $W = W_\perp + W_\parallel = v_\perp^2\lambda_\perp + v_\parallel^2\lambda_\parallel$ and values of $\lambda_\perp$ and $\lambda_\parallel$ for links.

Body segments are constrained such that for adjacent segments, joints for the facing sides must be in the same position (ie., the links stay joined together). Joint position is expressed in terms of x_i, y_i and φ_i for $i \in \{1, \ldots, n-1\}$, so

$$\begin{aligned} x_i + \frac{l_i}{2}\cos\varphi_i &= x_{i+1} - \frac{l_{i+1}}{2}\cos\varphi_{i+1} \\ y_i + \frac{l_i}{2}\sin\varphi_i &= y_{i+1} - \frac{l_{i+1}}{2}\sin\varphi_{i+1} \end{aligned} \tag{3}$$

Equations (2) and (3) form a differential–algebraic equation (DE) system [5] typical of non–minimal coordinate systems that can be numerically integrated.

3 Implementation

The neural and physical models described above were implemented in C++. The numerical solver collection from the GNU Scientific Library [6] integrated the DEs. A rendering program, to represent output of the simulation graphically, was implemented in Python using PyOpenGL [7]. The program accepted an output logfile generated by the simulation containing system state at 5 ms intervals, and rendered this state both to a monitor window and to file. The files were later combined into a video using mencoder [8]. While the simulator and renderer can be run without the intermediate step of a logfile, keeping a permanent record of simulations is advantageous, as are the abilities to run the simulation without rendering overhead and to re–render simulations.

Embedded 8th order Runge–Kutta Prince–Dormand method with 9th order error estimate (rk8pd) was selected as the numerical solver for the DE system. Also, rather than selecting a fixed step size an adaptive solving controller was used, that takes a variable number of smaller substeps, with backtracking, to keep solution error under a set maximum error limit. We set the error limit to 10^{-3}, which is the largest error rate resulting in acceptable precision for simulations of 100000 ms. This error limit should be decreased for longer simulations.

The physical and neural models were combined into one system of DEs for solution, a departure from Ijspeert [2] where the systems were separate. Ijspeert's physical solver used a step length of 0.5 ms, 1/10th that of the step length of his neural solver, presumably because the neural system behaves in a more regular way and is thus less subject to accumulated error, allowing larger solving steps to be taken and thus reducing runtime. But when an adaptive step–size controller is used this arrangement is detrimental to performance. Input from the neural system is constant for 10 steps of the physical system, then abruptly changes, resulting in "sawtooth" input. At every "tooth", the adaptive controller responds to the sudden change by backtracking and taking extremely small solving steps. For this reason a combined system was constructed, so the neural and physical models are solved at the same rate. This is faster, although more steps are taken through the neural system than are perhaps required.

4 Typical model behavior

We describe a typical simulation in which left and right brainstem inputs are set to 0.67, a suitable level for forward movement. Quantitative results vary with excitation level but general system behavior is the same. A 10 s simulation takes 620 s to run on an AMD 1800 CPU.

As simulation begins, both ends of the lamprey curl inward to the left (Fig. 2). As they start to uncurl at 100 ms, a wave forms along the body and is evident at 150 ms. This wave mirrors in place, then at 300 ms starts to propagate down the body. For the first 750 ms the wave propagates without causing forward motion, but then the lamprey begins to move and at 1000 ms has travelled

70 mm. At 2000 ms, the lamprey travels at 385 mm s^{-1}, 80% of its steady–state speed. It reaches 466 mm s^{-1} at 6000 ms and maintains this indefinitely (Fig. 3).

The body is about 1.5 wavelengths long as the initial propagation forms, but at steady state this becomes two wavelengths as segment co–ordination improves. Amplitude of the tailward wave peak is about three times that of the headward, partly because of the lesser mass in the tailward segments, and partly because they are at the end of the fish. Steady–state undulation frequency resulting from the given brainstem excitation is just over 6 Hz. Swimming speed and characteristics observed are similar to those observed by [2].

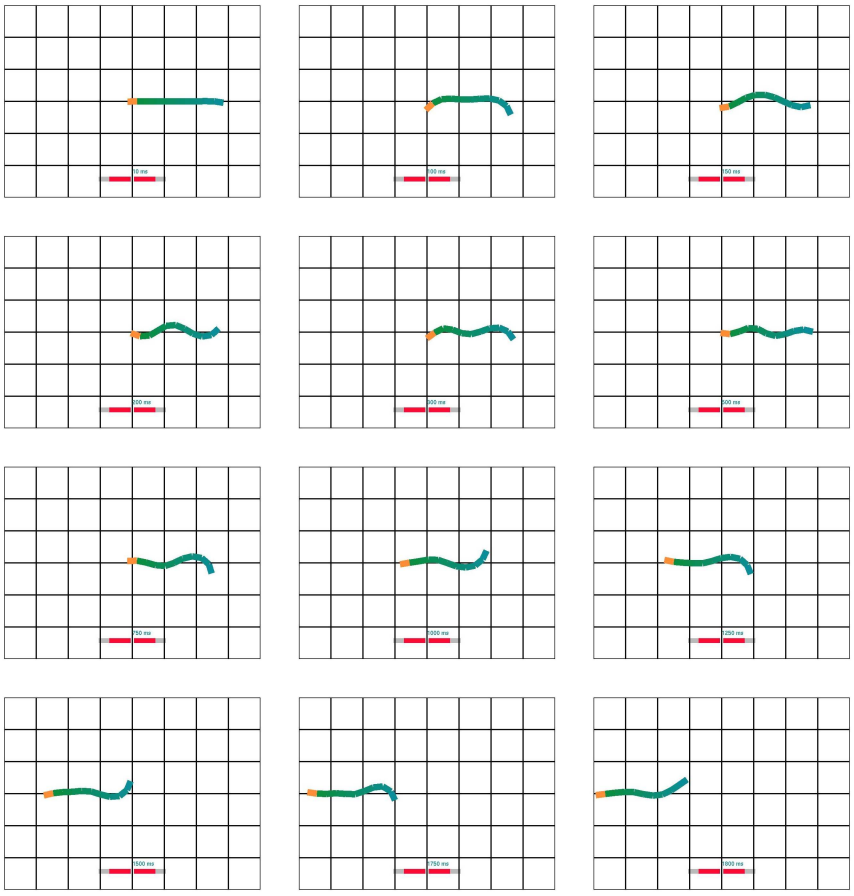

Fig. 2. Straight-line swimming behavior. Gridlines are 100 mm apart. Lamprey is in middle of each image with (fixed) input from left and right brainstem below. Images taken at 10, 100, 150, 200, 300, 500 ms, then every 250 ms.

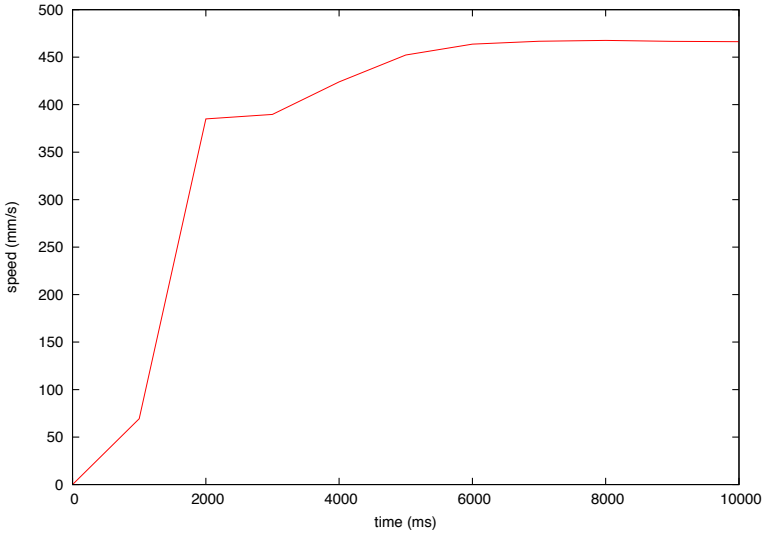

Fig. 3. Straight-line swimming speed

5 Experiments in varying neural segment count

Ekeberg [5] chose 100 segments in his neural model, staying true to biology. Selection and tuning of parameters followed from this choice. Simulation of a 100 segment neural model is slow and, given the demonstrated stability of the model, we decided to examine whether fewer segments could produce swimming. So, simulations were conducted to observe and measure behavior of lampreys with spinal cords of between 90 and 10 segments, in steps of 10 to preserve the relation between neural and physical segments. No other parameters were modified and extents of intersegmental connection were the same in all cases.

We found that any length of spinal cord, starting with 10 segments, can sustain oscillations and phase delay. Furthermore, all lengths can cause wave propagation in the physical simulation, resulting in forward swimming. There are, of course, differences in the nature and speed of the swimming produced.

As expected, the wavelength of the body does not change; however there is a noticeable decrease in the extent of perpendicular tail motion. In the 100–segment lamprey the amplitude of the tailward segments is around three times that of the headward, compared with two times in the 50–segment lamprey. The tail amplitude of the 10–segment lamprey barely exceeds that of the headward segments. At this length propagation of a traveling wave is only just perceptible, with most of the activity being the standing–wave inversion characteristic of the startup behavior of the 100–segment lamprey.

Shortening the lamprey spine causes a proportional decrease in swimming speed. Because the simulation exhibits different behaviors at startup before reaching steady state, swimming speed was measured by recording the time

taken for the lamprey to swim 15, 30, 50, 200 and 400 mm. The 50 mm milestone represents significant forward motion and can be regarded as the boundary between initial conditions and steady state. The 400 mm milestone gives a reasonable measure of steady swimming speed independent of starting conditions.

Fig. 4 shows the time for lampreys of different segments to reach the distance milestones. With one exception, the swimming speed of each lamprey is reasonably proportional to the reduction in segments. Regarding the startup behavior of the lampreys, it can be seen that while the 30–segment lamprey reaches the 15 mm milestone earlier than expected, and the 40– and 50–segment lampreys reach it later, this early lead has almost vanished by 50 mm.

The graph clearly shows the unexpectedly small length of time for the 20–segment lamprey to reach steady state. It passes the 15 mm and 30 mm marks ahead of the 100–segment lamprey, and reaches 50 mm in equal time. And while it takes 2.5 times as long to pass 400 mm, this is twice as fast as we would estimate from the behavior of the other lampreys. A 20–segment spine seems very compatible with the 10–segment body, for reasons not yet fully understood. Investigation of this high affinity has the potential to yield significant insight into the relationship between the neural system and the body, and bears exploration.

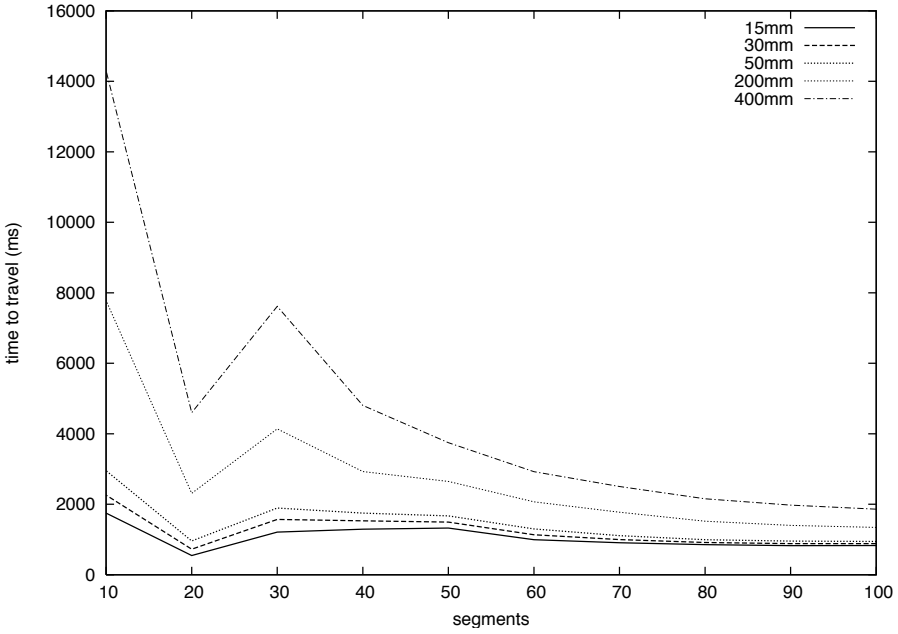

Fig. 4. Time for 10- to 100- segment lampreys to reach distance milestones

Most of the execution time is spent in the neural simulation and is essentially linear with the number of segments. This is because there are 24 DEs per neural

segment compared with 3 per body link giving a total of $24s + 3b$ DEs for a lamprey with s neural segments and b body links. Thus, reducing the number of neural segments results in a corresponding reduction in execution time. A 1000 ms simulation takes 78 CPU seconds for the 100–segment lamprey and 16 s for the 20–segment. Execution time for the 10–segment lamprey is 13 s, allowing us to estimate the time spent in the physical simulation as around 9 s.

The number of neural segments, and the intersegmental connectivity scheme, in Ekeberg's original simulation are well–chosen and result in a model that is both similar in layout to the biological lamprey and in behavior. By reducing the number of segments while keeping the same connection scheme and the same physical parameters, we are simulating a shorter lamprey stretched into a longer, more flexible body. That this arrangement can propagate a traveling wave and generate forward motion is additional confirmation of the model stability.

6 Conclusion

We describe the design and implementation of a simulation of the neural pathways and physical body of a lamprey realized in a system of DEs. Two approaches for increasing simulation speed with graceful degradation of biological realism are identified: (i) better use of numerical integration algorithms and (ii) reduction of the number of neural segments. The latter highlighted a particularly high affinity between a 20–segment spine and the 10–link body. The fact that a simpler neural architecture can produce acceptable behavior may lead to more widespread use of such methods in computer graphics and robotics.

References

1. Janvier, P.: Tree of Life: Hyperoartia. URL: `http://tolweb.org/tree?group=Hyperoartia` (1997)
2. Ijspeert, A.J.: Design of artificial neural oscillatory circuits for the control of lamprey- and salamander-like locomotion using evolutionary algorithms. PhD thesis, Department of Artificial Intelligence, University of Edinburgh (1998)
3. Lansner, A., Ekeberg, Ö., Grillner, S.: Realistic modeling of burst generation and swimming in lamprey. In Stein, P.S.G., Grillner, S., Selverston, A.I., Stuart, D.G., eds.: Neurons, Networks and Motor Behaviour. The MIT Press (1997) 165–172
4. Wallén, P.: Spinal networks and sensory feedback in the control of undulatory swimming in lamprey. In Stein, P.S.G., Grillner, S., Selverston, A.I., Stuart, D.G., eds.: Neurons, Networks and Motor Behaviour. The MIT Press (1997) 75–81
5. Ekeberg, Ö.: A combined neuronal and mechanical model of fish swimming. Biological Cybernetics **69** (1993) 363–374
6. Galassi, M., Davies, J., Theiler, J., Gough, B., Jungman, G., Booth, M., Rossi, F.: GNU Scientific Library Reference Manual. 2nd edn. Network Theory Ltd. (2003)
7. Fletcher, M.C., Liebscher, R.: PyOpenGL – the Python OpenGL binding. URL: `http://pyopengl.sourceforge.net/` (2005)
8. Bérczi, G.: Encoding with MEncoder. URL: `http://www.mplayerhq.hu/DOCS/HTML/en/mencoder.html` (2005)