An evolutionary model for 3D agents integrating continuous and plastic development

Artur Matos
Graduate School of Human Informatics
Nagoya University
Nagoya, 464-8601, Japan

Takaya Arita
Graduate School of Information Science
Nagoya University
Nagoya, 464-8601, Japan

Abstract

Most of the current research in generative encodings for artificial creatures has only focused on the final matured genotypes, and doesn't take into account the developmental process itself. In this paper, we introduce a grammar based approach for generating agents with both 3D morphology and neural networks that leads to more natural developmental processes than previous approaches. This new model is based on Lindenmayer systems, but with added extensions for dealing with continuous development of limbs, and for simulating chemical interactions between development and the surrounding environment. This paper includes a detailed description of the model, as well as some results of our preliminary experiments for evolving agents that exhibit walking or other similar behaviors.

1 Introduction

Since Karl Sims' seminal work [5] on virtual creatures, significant research has been done for evolving agents with both integrated 3D morphology and neural systems. Most of the research currently being done in the field has been focusing on generative encodings, where the genotype is interpreted as instructions for building the phenotype, instead of directly mapping traits to it. Regarding these generative encodings, two distinct approaches can be observed in the current literature: the first one uses formal grammars or high-level structures for representing developmental processes (*grammar based approaches*), for example, Lindenmayer systems (LSystems). On the other hand, *cell chemistry approaches* attempt to simulate the low level interactions between cells during development, for instance, gene regulation and targeting. Although at a first glance these two approaches appear to be distinct and even opposite to each other, this division is largely artificial, and it should be possible to incorporate concepts from one approach to the other. So in this paper, we present a developmental system that although being grounded on developmental grammars, attempts to bridge between these two approaches, by introducing concepts that can easily be modeled on cell chemistry systems but that are usually not found in the grammar based ones. This new model, based on a previous one by Hornby and Pollack [1], integrates two different LSystems: Differential LSystems (Prusinkiewicz, Hammel and Mjolsness [4]) for modeling continuous processes, for instance, elongation of limbs; And Open LSystems (Mech and Prusinkiewicz [2]), for simulating chemical interactions between the developmental process and the environment.

2 The 3D agents and the environment

The agents in our model have both a 3D morphology and a simulated nervous system to control it. The 3D morphology for each agent is made of rectangular parallelepipeds connected by hinge joints, that restrict the movement of the connecting parts in one degree of freedom. The nervous system actuates on the joints by changing the speed of actuating motors present on each joint. The nervous system is a simple free form neural network, and includes sensors for the environment and joint positions, processing nodes and actuators for the joints. The processing nodes include sigmoids, linear transfer functions and sinusoid oscillators. The nodes in the neural network use standard propagation functions on the incoming connections, that is, the weighted sum of all the connections values, and are constantly active (no activation function is considered). For the sinusoid oscillator, the computed weighted sum of the incoming connections works as frequency modulator for the sinusoid, mapping the frequency in the interval $[0, 2\pi]$.

The environment simulates all physical interactions, including gravity and collision with objects. A flat ground is also simulated, extending indefinitely in the XZ plane. The simulation may include other objects depending on the behavior that is being evolved.

3 The developmental process

Each step for the developmental process is represented by a sequence of commands indicating the morphology and neural system for the agent. The commands themselves are changed by the LSystem associated with each agent through developmental time. The commands used are similar to the ones by Hornby and Pollack [1], but extended for allowing continuous values, in order to use differential Lindenmayer Systems. The developmental commands for the morphology work as a turtle constructor, and they include: *forward(n)* for moving the turtle forward n units in the currently defined direction, creating a new stick, and connecting it to the previous existing one by a joint; *backward()* for moving the turtle cursor back to the parent stick; *right(n)*, *up(n)*, *clockwise(n)* for allowing to control the direction of the turtle, rotating in the specified direction n units, with 1 corresponding to $\frac{\pi}{2}$. Negative values can also be used for moving in the opposite direction. Finally, *revolute-1()*, *revolute-2()*, *twist-90()* and *twist-180()* change the kind of joint to be created between two successive forward commands. The revolute commands create joints over the current turtle's Z axis, with hinge stops between [0, $\frac{\pi}{2}$] for *revolute-1()*, and [$-\frac{\pi}{2}$, $\frac{\pi}{2}$] for *revolute-1()*, and *revolute-2()*, respectively. The twist commands work in a similar way, but for the X axis. The developmental commands for the nervous system work as edge encoding commands, and they are described in table 1. There are also stack commands (*push()* and *pop()*), for pushing a retrieving the current state of the turtle and neural constructor, allowing to create branched morphologies.

3.1 Representing continuous development

DLSystems are based on parametric Lindenmayer systems, but they add a differential component for modeling continuous changes. Due to space constraints, parametric LSystems themselves will not be explained here and we will focus on dLSystems instead. For a good introduction to LSystems, the reader is referred to to Prusinkiewicz and Lindenmayer [3].

The first major difference concerning dLSystems is that development doesn't occur in discrete steps, instead there is a continuous developmental time frame t. As in PLSystems, the development is iterative, but it progresses by an user specified time step, Δt. This time step is completely detached from the underlying model, so it is possible to use a different Δt in order to observe the developmental process with more or less detail, as desired.

For specifying the changes occurring as t progresses through the developmental space, two different kinds of successors are used. Continuous changes are specified by using differential equations, that relate the changes in parameters' values to the step Δt. Structural changes, or adding new limbs or nodes to the agent are specified by sequence of tokens, as in PLSystems. Discrete transitions are applied on the same way as in PLSystems, the token being replaced by the matched sequence. Depending on the time step ΔT, the LSystem interpreter may need to subdivide the interval for taking into account both continuous and discrete transitions. An example dLSystem can be seen in figure 1. In this example system, for $\Delta T = 1$, the following sequence, starting from ω would be derived: $B(1), A(0,0)$; $B(2), A(2,1)$; $B(3), B(4), C(2)$. A smaller Δt (for instance 0.5), would yield $B(1), A(0,0)$; $B(1.5), A(1,0.5)$, $B(2), A(2,1)$.

$\omega:$	$B(1)A(0,0)$			
$B(x):$	$x < 4$	$\rightarrow$	**solve** $\frac{dx}{dt} = 1$	(1)
	$x >= 4$	$\rightarrow$	$A(2,2), B(1)$	(2)
$A(x,y):$	$y < 2$	$\rightarrow$	**solve** $\frac{dx}{dt} = 2$	(3)
			solve $\frac{dy}{dt} = 1$	
	$y >= 2$	$\rightarrow$	$B(x), C(2)$	(4)

Figure 1: An example dLSystem

3.2 Evolving dLSystems

Currently, our model uses a standard Genetic Algorithm for evolving dLSystems that exhibit specific behaviors, for instance, walking or jumping. Each agent is created from a random Lindenmayer System as described before, undergoes the developmental process, and then it is evaluated for the task at hand. Each genotype contains a starting axiom and an LSystem, both constrained in order to exhibit only viable growth processes. This is assured both by the random generation process and by the GA operators.

In order to model continuous development there are two different sets of commands present in the system. The first set, corresponding to the commands explained before, are used for modeling already stable processes. Another set of commands mirrors exactly the same functionality, but are used for modeling maturing processes that are still changing in developmental time. This maturing set of commands has the same name as their matured counterpart, prefixed by an "m". This separation is needed most of the times for modeling growing phenomena by using Lindenmayer Systems as pointed out by Prusinkiewicz, Hammel and Mjolsness [4].

For assuring continuity, the generated rules are always instances of the same template, working as follows: if matched, a maturing command grows linearly, until it reaches a certain threshold. After this threshold has been reached, the token is replaced by its matured counterpart, possibly with some more additional tokens added. Only linear differential equations are used. Additional constraints may be present depending on the command. An example rule for the *mforward(a)* command can be seen in figure 2.

$$\begin{array}{llll} mforward(x): & x < 2 & \rightarrow & \textbf{solve } \frac{dx}{dt} = 1 \\ & x >= 2 & \rightarrow & forward(a), push(), \\ & & & right(-1), \\ & & & mforward(0) \end{array}$$

Figure 2: An example branching pattern

In addition to this, each genotype also maintains a set of control tokens, that can be used for controlling development at a higher level. These set of control tokens (starting alphabetically from A) are not used by the morphology or neural system constructor, but they can be used for generating higher level rules of development. The rules, if generated, follow the same template as described before.

Random genotypes are first created by generating random rules and successors. Mutations occur by replacing tokens in discrete successors by random ones, by changing conditions values, tokens in the starting axiom, or by adding or deleting new rules. Crossover occurs by choosing a token from all the available commands at random, and by exchanging rules alphabetically on that point.

3.3 Interactions with the environment

We also added plasticity to this developmental model by using another LSystem extension, Open LSystems (Mech and Prusinkiewicz [2]). This extension simulates chemical inflows and outflows with the surrounding environment, allowing the developmental process to both affect or be influenced by the surrounding environment. For this, we extended our current model for the environment by adding an unary array for representing chemical concentrations, that can be read and changed both by the developmental process and by the neural controller of the matured phenotype.

Summarily, OLSystems exchange information with the environment by using special tokens called *environmental symbols*, that are processed after each developmental step takes place. In our current model, these are of the form $?E(x, y, z)$, where x stands for the kind of command used (0 for setting a chemical concentration in the environment, 1 for setting a value), y for a index in the chemical concentration array, and z for the concentration value, if in set mode. After each LSystem iteration, any generated environmental symbols are processed in sequence, and allowed to change and retrieve the values in the environment. For any commands of the form $?E(1, y, z)$, the system replaces z with the value present in the environment at that time, in order to allow the productions to access that value on the next iteration. By using these commands as left or right context in rules, it is possible to trigger different developmental paths depending on the surrounding environment.

4 Preliminary experiments

Currently, the above described system has been used for evolving developmental processes for interacting tasks, like walking or jumping. One example of an evolved creature can be seen in figure 3. The fitness function used was the distance moved by the agents in the XZ plane. The developmental process, as can be seen from a) to c), works by generating two linear branches. The final evolved creature behaves like a snake, coiling itself in the ground. The system is implemented in Java, using Open Dynamics Engine for simulating physics and collisions. The renderings are done by using Art of Illusion (a free modeler and animation tool).

Other experiments are being planed and conducted for studying developmental plasticity in the implemented model, modeling of developmental interactions between predator and preys (cyclomorphosis) and canalization.

Command	Description
offset-weight(n)	Adds or subtracts n to the weight of the current link
duplicate(n)	Duplicates the current link
loop(n)	Creates a self connecting link from the current head neuron
merge(n)	Replaces the current link and connecting neurons with the head neuron. All connections from both neurons are copied into this neuron.
next(n)	Changes the head neuron to its nth sibling if it exists
parent(n)	Changes the head neuron to its nth parent, if it exists
output(n)	Creates a new output neuron, connected to the current joint. Creates a link from this new neuron to the previous head neuron.
reverse()	Reverses the current connection
split(n)	Creates a new sigmoid neuron, creates a connection from the current head to this new neuron, and from the new neuron to the current tail. Sets the current connection to the new one.
setNodeType(n)	Changes the node type of the current neuron. 0 → linear, 1 → sigmoid, 2 → oscillator
cut	Removes the current connection

Table 1: Commands for behavior

5 Conclusion

In this paper, we introduced a new developmental model that attempts to bridge between the two main approaches previously found in the literature. It is our hope that this model will demonstrate the advantages of both approaches, and will allow to model developmental processes more naturally in simulations. The added Open LSystems component should be important for studying developmental plasticity and other issues related with development, and hopefully this should be clear with future experiments.

Acknowledgments

This research is funded with a Japanese MEXT scholarship (400000357010989).

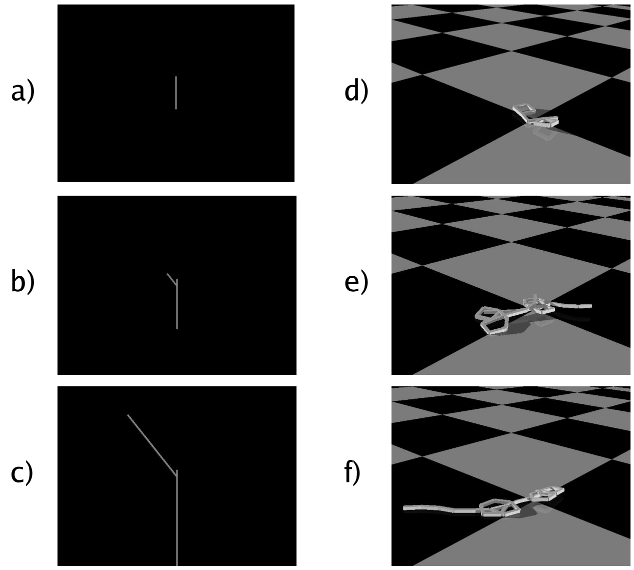

Figure 3: Development and interaction for a sample creature. a) to c) development seen from above for $t = 0.54$, $t = 0.99$ and $t = 2.33$, respectively. d) to f) The same creature, after development, interacting in the environment.

References

[1] Hornby G. and Pollack J. Creating high level components with a generative representation for body-brain evolution. *Artificial Life*, 8:223–246, 2002.

[2] Radomir Mech and Przemyslaw Prusinkiewicz. Visual models of plants interacting with their environment. In *Proceedings of SIGGRAPH 96*, pages 397–410. ACM SIGGRAPH, 1996.

[3] Prezemyslaw Prusinkiewicz and Aristid Lindenmayer. *The algorithmic beauty of plants*. Springer Verlag, 1996.

[4] Przemyslaw Prusinkiewicz, Mark Hammel, and Eric Mjolsness. Animation of plant development. In *Proceedings of SIGGRAPH 93*, pages 351–360. ACM SIGGRAPH, 1993.

[5] Karl Sims. Evolving virtual creatures. In *Siggraph '94 Proceedings*, pages 15–22. ACM SIGGRAPH, 1994.