Math for Biologists Software: A Plant Growth Engine

Developed by Christopher Quinn (cjq3@cornell.edu) under
Professor Bruce Land and Professor Ronald Hoy

May 2006

Introduction

Undergraduate biology students rarely receive thorough mathematics instruction, partially because the material is not taught in a manner relevant to their work. We want to teach biology students the principles of mathematics that are relevant to evolution (population dynamics, constraints) and to genomics (string functions, languages and grammars). We also want to tech them in a fun and interactive manner, in a means such that it will be easy for them to glean the principles we want to teach. The most intuitive platform through which to teach these principles is through an artificial life simulator, simulating populations of creatures interacting with each other and trying to survive.

There are many artificial life programs available. All were made with specific goals in mind and have user interfaces catering to their purpose. Some quite popular ones include Framsticks, GenePool, and Biomorphs. Framsticks is a mature environment which allows for some user interaction, modeling structural biomechanics (joints) and neurons, but not much natural selection and the constraints are not easily accessible to (novice) users. GenePool is quite different in that it focuses primarily on population modeling. Thus, evolutionary changes in a population are quite easy to see, but there is little user manipulation and no control of individual creatures' code. Biomorphs, on the other hand, is a program that works mostly through user interaction – the user decides which members of a population are most fit. Although there are many other programs, most of the mature projects fall primarily into one of these categories. Since our goal is to have an intuitive environment which can teach the mathematics of both individual creature development and population dynamics, we wanted a program that merged the characteristics of the other programs. Consequently, we began development on our own software that would fit our needs.

There are several approaches to making artificial life simulations. On one end is mostly gaming environment (focusing on AI aspect), and on the other is a hands on string based environment, such as that pioneered by Aristid Lindenmayer. He was a biologist who developed simple rule and grammar approach to modeling plant growth. It is known as "L-Systems." By using the simple string and grammar approach as a very simplified analog of genetics, populations of plants (each described with different strings from the same alphabet) can be grown and the optimal plant (with fitness largely depending on string, partially depending on such factors as position in environment, distance to neighbors, etc) can be determined. This provides a powerful and easy to implement framework to then make the software and allow it to have the user's interactions (principally manipulating the strings of the plants and the allowed operations) to coincide with an illustration of the mathematical principles we hope to illustrate (genetics coinciding with alphabets, grammars, and rules, and optimization principles inherent in population dynamics with specific constraints).

A brief introduction to L-Systems:
The principles of the language are that you can have an alphabet and operations on strings made of that alphabet. The alphabet and allowed operations can take on many forms. Let's say you want to have a simple drawing scheme that mimics an insect's walk. 'F' will stand for a line segment (corresponding to a step) and '+' will stand for an angle change. You might start off describing it as 'FF+F'. You decide that this is not very exciting. Consequently, you add on a parsing rule, that will take the string, and whenever 'FF+' is encountered, it is replaced with 'F++FFF+FF'. With 3 iterations,

'FF+F'

'***F++FFF+FF***F'

'F++F***F++FFF+FF***FFF'

The underline marks outline where our rule will parse and italic-bold outlines its replacement. By looking at the resulting string, it becomes evident that depending on the alphabet and on the operations, any number of changes can take place in what the initial string was and the final string becomes. By implementing an L-System, it will be quite easy to keep track of the individual plants (most of the information is contained in a simple string).

Methods

In the current implementation, all plants have a single rule, and that rule is specific to the plant; initially a given population starts off with a single rule that is defined by the user, but through mutation develop varying rules. It is thus mutation rate and the starting rule that is controllable and allows for variation in population development. Users currently control the rate for mutation as a whole, while the rates of different mutation types (changing angle, deleting length) are modulated by the general rate. Each plant's situation (its size, shape, physical distance to neighbors) are used in defining each member's fitness. Since we are using plants as creature types, the main criteria for fitness is amount of sunlight each plant receives. To model this, a camera takes a picture from a user specified angle of the population, and for each pixel that is exposed, a plant received one energy unit (the weight of pixel/size energy unit is defined by the user).

The plant's current status is held in a string (from the alphabet) and its growing rule is specified. By extending the simple example as mentioned before, this software uses a small, simple alphabet:

F{n}	move n steps forward
+	increment plane angle (theta)
-	decrement plane angle (theta)
/+	increment cylindrical angle (phi)
/-	decrement spherical angle (phi)
[	push current position and drawing angles on a stack
]	pop position and drawing angles from a stack

The alphabet is used to graphically represent each plant. There are three degrees of freedom (x, y, z) for the plant, using spherical coordinates. Using this alphabet strings are generated that, following the interpretations/meanings as given by the definitions draw plants. When drawing the population, initially, there is a blank canvas with an axis. Each plant is drawn (its string stepped through until all of the lines are drawn) on the axis. Using the camera's position as the sun's position, the canvas's image data is taken and pixel color counts are determined, with higher pixel counts corresponding to higher energy values; this is meant to be the analog of how much sunlight a given group of plants receive. Then seeds are created, dead plants removed, and a new generation develops. Young plants can "grow," having their strings parsed according to their specific rule, and all plants can have their rule changed or mutated.

How the population changes is entirely dependent on how the constraints define the fitness function for a given population. The current constraints for fitness are that number of pixels exposed to the camera is directly proportional to the energy increase of each individual, it takes a universal cost to grow, and there is an energy decrement proportional to each length. Plants are only allowed to grow the first two time steps of their existence, and they need a minimal amount of energy to grow. Each time step an amount of energy proportional to an individual's age is subtracted from that individual's total energy.

Matlab is currently used for simplicity of development. Because of the speed and memory required, as this project matures it might be ported to another language and use better graphics (such as opengl), but development is easier in Matlab.

Results

Currently, the plants that grow are quite simple and do not have constraints that are identical to nature, and this allows for flexibility in how information is conveyed by the engine. The goal of the project is not to be the most accurate model for plant growth, but

instead to be a platform from which a system can be developed that can give biology students an intuitive sense at the mathematics of evolution and genetics. Consequently, the constraints need to be modified to tailor what happens to the population and what understanding of genetic and population dynamics can be gained by the software
A variety of movie clips and screen shots can be obtained from (__LINK__).

By implementing the constraints previously mentioned, some intuitive results surfaced for certain ranges of settings of the systems. The results described below primarily apply to populations whose starting rule and light camera position would allow the population to survive and grow well given high energy unit size for each light-exposed pixel (otherwise, the population would die out quicker, although would still display these properties, only scaled to how easy populations died out).

<u>High Energy/per pixel</u> ➔ Often result in large populations (since most plants receive enough energy to not only stay alive but produce many seeds).

<u>Low Energy/per pixel</u> ➔ Initially have small populations, and quickly dwindle to one plant and then die out, or die out almost immediately.

<u>Large Mutation Rate</u> ➔ Plants in later generations become snakelike, loopy, and sometimes quite large and complex.

<u>Small Mutation Rate</u> ➔ Plants are effectively the same sizes and shapes for numerous generations.

<u>There are no Very Large Plants</u> ➔ The sizes of the largest plants in populations decrease as the size penalty is increased.

Combinations of the variations yielded behavior in line with these results. The energy/pixel dependence on survival is expected, as the goal of the population is to maximize energy netting (whenever there is competition). The mutation rate dependence on structure is not surprising because of how operations are allowed to occur to strings comprised of alphabet elements. Since angles can be added, deleted, or changed (from an increment to decrement and vice versa), it is quite easy for populations with high mutation rates to have greatly varying structures from not only the start string but also each other. Also, the magnitude of any constraint seemed to cause an increase in the expression of the expected population behavior.

Some example footage of populations with various constraint settings, all with the same initial rule (so only constraints varying) are included below.

Conclusion

The population dynamics yield intuitive results based on what constraints are implemented and to what degree. Consequently, constraints can be added/deleted at will in order to make an informative environment. If a constraint illustrates a principle we want well, then they can be kept. This software provides a good engine from which enhancements can be made and constraints tweaked to provide an environment which is desirable for illustrating mathematical principles to biologists, particularly with interest in genomics and evolution. Some possible changes not only for constraints but for the whole system include:

UI

- Make a constraints panel that is attached (or maybe popup if select a menu option)
- Make a bank of "good" startrules to give users more than one example
- Make a "random rule" generator that user can select ---
- PLACE MORE EMPHASIS ON MAKING THE RULES, not just running constraints users need to learn how to manipulate strings
- Make a panel that allows editing of all user-defined variables.
- Make energy needed per seed to be a user-defined variable
- Have a "simple" display with only necessary parts (no printout, no excess) and a "full" display (with all variables available to edit with links to descriptions of the variables)
- "Test-A-Rule" – allow you to press a button to open a modified gui that lets you do a single plant at a time, modifying its start string & start rule before implement in a population
- Allow for a "random seeding" in which take randomly selected start rules from a list of approved start strings so not all plants have to wait for genomes to differ through mutation
- The angle size of the {'+','-'} and {'/+','/-'}

Constraints –

- Size of plants need to be better constrained
- Balance of plants (torque)needs to be constrained (one horizontal large branch versus a T).

Growth –

- Make the chance a growth will be replaced by rule a user defined option
- Make the penalty for length, height (separate from length), and torque user defined

Genome –

- maybe have multiple growth rules per plant, and rules not only operate solely on 'F' (stochastic)
- Make a list of good start rules (ones that make for well-balanced and "full" trees)

Mutation –

- Alter rates of mutation , particularly adding of angles and to lesser extent adding of lengths → make it much more common to change than add/delete, and equally likely will and as will delete

Seeding and Merging genomes –
- make it so that rules do not exponentially increase in length
- allow for more spreading?
- !!! currently, only pass through major branchings, decide you to merge rules with nested branches!!! Since will take a whole [….] as one segment does not look inside and dependent on position in string
- Maybe do a thing where look at all nonbranches up to the first branch (so segment), merge lengths and angles intuitively, then decide what to do if two identical plants except the second has an extra branch at beginning → in real life would not be all that different
- Prof Land pointed out, which is true, that just because two plants have very close phenotypes (physical appearance), their genotype could vary significantly, since there is a layer of abstraction.
- Not sure if it makes sense to limit seeding to just one way of merging two plants (walking from "beginning" to "end" of a string, where directionality is significant in drawing, and so maybe walking in same direction is important).

Camera –
- Work with graphics such that the camera positions and angles never change unless the user wants them to. (Matlab tries to fit everything within)
- The camera angles with "Set user view as Light view button" do not set correctly → there is almost always a bad change

Other –
- Increase performance by optimizing code – too many loops
- Speed bottleneck at growth (if watch time for printout of each part of runstep)
- Port to Java, use JOGL

This project provides a good framework from which to teach some biologically significant areas of mathematics. One is genomics, which is tied into alphabets and strings, allowing users to understand how the physical status of an organism can be tied into long chains of DNA. The other is population dynamics, which is essentially a maxima-minima search in a field with constraints; since the constraints define what "fitness" means, by letting the user manipulate the physical world, the user can see the effects on how the population evolves. Also, several processes have probabilities, such as mutation rate and growth rate, and so by changing the probability an event will happen, the user can see the effect on the whole population.

This engine should provide for a good basis from which a final, ideal interactive environment can be built, with the ideal being the one that presents the mathematical principles of genetics and population dynamics in the simplest and most intuitive fashion.

References

http://en.wikipedia.org/wiki/Artificial_life

Background papers by Przemyslaw Prusinkiewicz (worked with Aristid Lindenmayer on L-System plants and has continued to do research)

http://algorithmicbotany.org/papers

- http://algorithmicbotany.org/papers/fractals.tip2004.pdf
- http://algorithmicbotany.org/papers/mpg.copb2004.pdf
- http://algorithmicbotany.org/papers/asl.acta2004.pdf

Algorithmic Beauty of Plants, a great book describing L-Systems, by Lindenmayer and Prusinkiewicz:

- http://algorithmicbotany.org/papers/abop/abop.pdf (17MB)
- http://algorithmicbotany.org/papers/abop/abop.lowquality.pdf (4MB)

Artificial Life portal:

- http://www.alife.org/index.html

Framsticks Website:

- http://www.frams.alife.pl/

Genepool Website:

- http://www.ventrella.com/GenePool/gene_pool.html

Biomorphs Website (a good implementation)

- http://physics.syr.edu/courses/mirror/biomorph/

Marc Ebner's (a researcher who worked on population dynamics of virtual plants) website for co-evolution

http://www2.informatik.uni-wuerzburg.de/staff/ebner/research/evoPlant2/evoPlant2.html