Further Adventures of the Sonomorphs

Gary Lee Nelson
TIMARA Department
Conservatory of Music
Oberlin, OH 44074
(440) 775-8223
email: gary.nelson@oberlin.edu

1. Introduction

In 1993, I reported the results of preliminary work on the application of genetic algorithms (GA's) to music.[1] This research was inspired by Oxford zoologist, Richard Dawkins[2] who describes his journeys through a genetic cyberspace where he models the evolution or organisms he calls "biomorphs." Each biomorph is a small graphic image drawn by a recursive subdivision algorithm that is driven by a numeric vector. Dawkins likens this vector to a genetic code. In each new generation, biomorphs are bred by causing small random mutations in the vector. A series of children are born from which he selects the parent of the next generation based on subjective visual criteria. His thesis is that complicated organisms are not the result of chance or conscious striving toward a long term goal. Rather, they arise from the accumulation of small changes that assure the survival of the fittest. The process immediately reminded me of the means that composers use to search the space of musical constructs to find just the right choice for a particular moment in a piece.

In paralleling Dawkins' research in the sound domain, I coined the word "sonomorph" to describe elements of musical compositions whose size is somewhere between a motive and a phrase. I have used a variety of strategies for constitution of the genetic code and selection criteria. The result is a body of pieces where musical phenotypes evolve through a combination of genetic mutation and reproduction. The genetic codes that are passed to each generation are interpreted with algorithms that generate musical structures. Numbers, symbols, and functions are mapped onto musical parameters and presented to the composer for subjective visual and aural evaluation. The sonomorphs that survive become the progenitors of future populations. Particularly fit individuals are saved and assembled with others into finished compositions.

A major goal of the work I reported in 1993 was to find optimum methods for structuring and interpreting genetic codes for musical organisms. I was looking for codes and interpretations that were rich enough to produce a broad range of musical expressions yet simple enough to be effective in interactive environments for composition and performance. An equally important goal was to discover something about the nature of subjective musical choice so that aspects of the process could be automated in programs for composition. Substantial progress has been made toward reaching these goals.

In this paper I will detail the current state of my research. This will include a summary of the links I am making between GA's and other feedback processes like iterated function systems

---

[1] Gary Lee Nelson, "Sonomorphs: An Application of Genetic Algorithms to the Growth and Development of Musical Organisms," Proceedings of the Fourth Biennial Art & Technology Symposium, Connecticut College, March 4-7, 1993, pp. 155-169. Copies of this paper are also available via email from the author.

[2] Richard Dawkins, *The Blind Watchmaker*, W. W. Norton, New York, 1987.

(IFS), Lindenmayer systems (L-systems), cellular automata (CA), and chaos. I am concerned with the use of GA's in generating procedural plans and transformation methods to achieve what Dawkins calls "the evolution of evolvability (EE).[3]" The concept of EE is an elegant example of the kind of thinking that grows from the examination of the feedback processes listed above. When we dramatically change the way we look at things we find new things to look at. In music, the application of EE to GA's results not only in offspring that develop into obvious variants of their parents but also offspring that mutate into surprisingly new structures. In my earlier study, I worked with genotypes of fixed size and syntax. In more recent research, I have been using mutations that change the structure as well as the content of the genotypes. I am varying the tools as well as the materials.

I will explain the methods I have used in making my pieces with the aid of sonic and visual illustrations.[4] Examples include the generation of monophonic and polyphonic pitch structures, rhythmic patterns, dynamics, articulation and timbre. I will show the implementation of some my techniques in the MAX programming language.[5]

2. Genetic Algorithms

A general grasp of genetic algorithms is important for understanding the present discussion. I repeat a section from my 1993 paper in abbreviated form.

Genetic algorithms are metaphors for natural evolution. Individuals in a population are represented by a string of numeric or symbolic parameters that hold the genetic code or genome of the organism. Each parameter symbolizes a locus on a chromosome. The value of each parameter encodes the allele[6] at that locus.

Genetic algorithms evolve a population by examining the genome of each individual and assigning a fitness score that is based on the suitability of the organism for performing a task or fulfilling a function. Successful organisms survive to become the parents of the next generation by a breeding process that includes crossover and mutation. Figure 1 shows the steps in a typical genetic algorithm.

1. Create random initial population
2. Evaluate each individual
3. Select parents
4. Breed
5. Add children to next generation
6. Replace parent generation with children
7. If end condition not satisfied repeat from step 2

Figure 1. Steps in a genetic algorithm.

---

[3]Richard Dawkins, "The Evolution of Evolvability," pp. 201-220 in Artificial Life (edited by C. G. Langton), Addison-Wesley, 1989.

[4]Sound examples and Max patches are available from the author.

[5]Miller Puckette and David Zicarelli, OpCode Systems, 3950 Fabian Way Suite 100, Palo Alto, CA 94303

[6]either of a pair of contrasting characteristics inherited according to the Mendelian law.

The following is only a brief description of the components of a genetic algorithm. I refer the reader to David Goldberg's excellent textbook on the subject for a more rigorous exposition.[7]

2.1. Genome Encoding

A common form of genome encoding is the bit string. Individuals in a population are represented by a string of some specified length. The bits, taken singly or in groups of fixed or varying size, convey the genetic information relative to each allele.

Other encoding schemes use strings of integers or real numbers. Each number holds the current state of a locus on the chromosome. These numbers may become parameters of a rendering algorithm that produces images or sounds.

In another category we find genomes represented symbolically. Each locus contains a token that may stand for a function to be executed or an action to be performed. The genome can be used to describe an individual's behavior in an environment. In biological models, food foraging actions and criteria for selection of a mate might be coded this way. In algorithms for game playing or machine learning the genome can be a computer program where the alleles represent small pieces of code. The success of the organism in solving a particular problem determines its fitness score.

2.2. Fitness Tests

Fitness tests vary with the goals of the genetic experiment. In the bit string model, fitness might be determined simply by viewing the bits collectively as an unsigned integer and using the value of that integer as the fitness score. Alternatively, we might add the 1's in the string and reward individuals with the highest or lowest sums or with sums that fall within a particular range.

One of the genetic models in my 1993 report was directed toward evolving rhythmic patterns and used a bit summing test. In that case the bit string was interpreted as a series of equally spaced pulses. If a bit was on, a note was articulated; if a bit was off, a rest was made. Selecting for high bit counts evolved rhythms that tended toward high density. After a few generations the individuals played notes on nearly all of the pulses. Selecting for low bit counts created an opposite tendency toward thinner rhythmic texture and finally the silence of extinction. In simple fitness tests like these there is a rapid and musically-unfortunate conversion toward the idealized state implied by the test.

2.3. Breeding

Once the most suitable parents survive the fitness test, breeding the next generation begins. There are variants of the breeding algorithm. Some algorithms depend on the number of parents chosen for a child of the next generation. The obvious choice from a human perspective is two parents. The usual operations in breeding are crossover and mutation.[8]

---

[7] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, New York, 1989.

[8] The current version of my MAX implementations of crossover, mutation and several other random processes are available via anonymous ftp at kahless.isca.uiowa.edu/pub/ftp/max/NelsonRandom.hqx.

In the two parent model the bits are combined using a crossover method. To begin the breeding, one of the two parents is chosen by a coin toss. The bits in each parent's genome are scanned in parallel. When the first pair of bits is considered, a coin is tossed again. If it is tails, no crossover occurs and the first bit for the child's genome is taken from the parent that was chosen in the very first coin toss. If the coin turns head up, a crossover occurs and a bit is taken from the opposite parent.

Figure 2 illustrates breeding of two parents with three crossover points.

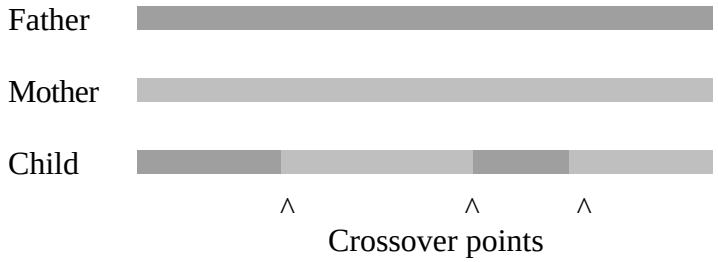


Figure 2. Breeding with three crossovers.

Figure 2 shows that selection began with the father and crossed over at the points indicated by the carets.

In computer simulations, the coin is replaced by a probability function with a variable weight. If the weight favors tails, few crossovers are made and longer strings of bits are taken from each parent. The result is inheritance of larger sequences of genetic code and greater dominance of features from one parent or the other. If the weight favors heads, crossover is more frequent and shorter substrings are passed to the next generation.

Like crossover, mutation is controlled by probability. In the bit string genome of a child bred by crossover, the probability of mutation determines the likelihood that a bit will change state from zero to one or vice versa. As the bit string is scanned, a weighted random function decides whether to alter the state of each bit. High probabilities reduce the effectiveness of the parent selection step and tend toward random evolution. If the mutation probability is low it can perturb the conversion effect that the fitness test encourages without altering the general flow of evolution.

3. Experiments

In these experiments, the output of various iterative models are mapped into musical patterns to produce phrases and sections of a musical composition and, sometimes, the an entire composition. The numbers given by the models become pitches, durations, dynamics, articulation and timbre choice through various methods of scaling and indexing. Since the subject is so vast, I will concentrate on pitch and time point mapping in the present discussion.

3.1. Rendering

The models described below produce patterns of real numbers in one or more dimensions. I usually map the first output (X) onto time and the second (Y) onto pitch. In a one dimensional model, successive outputs can be combined to create sets of related numbers. For example two consecutive outputs can be interpreted as X and Y to imitate a two dimensional system.

Sometimes the output range is unpredictable. To convert these numbers to a manageable range I use functions like SIN(X) to convert to a range of -1 to +1. I also use the modulus function X MOD N to convert to a range of 0 to N. In a two dimensional model, additional related numbers are created by taking the product of X and Y or using functions like SIN(X+Y). These operations sometimes change the form of the image that is associated with a particular model but the relative complexity of the pattern remains. Remember, my goal is not to find a particular pattern but to embrace patterns that are subjectively 'interesting' on grounds of complexity, beauty, asymmetry, and balance of form.

To map into pitch, a number is scaled to some desired interval range. An offset value is added to move the scaled numbers into a particular register. Scaling a set of numbers that lie between 0 to 1 by 24 and adding 48 maps the pattern as a two octave 'melody' centered at middle C.

Attack time presents a different problem. If the models are run in real time and X is to become attack time, I often encounter a situation where the second X in a pair has a lower value than the first. With strict interpretation, this means that a pitch has arrived after the time is must be played. Time cannot flow backward. My solution is to use 'time frames' of specified duration that corresponding to musical beats or measures. X values are scaled and mapped into these frames. An X value whose time has already passed is mapped into its corresponding position in the next time frame. Given a current point 12 seconds into a piece, a frame of 2 seconds and an ordered pair of X values .7 and .3 (in the range 0 to 1), the mapping steps would be:

	X_1	X_2
• scale X values to frame (x 2)	-> 1.4,	.6
• add current point offset (+ 12)	-> 13.4,	12.6
• add frame length to 'tardy' attacks (+ 2)	-> 13.4,	14.6

3.2. Visualization and Audition

At each breeding stage of the GA, time can be suspended and the current population tested for fitness. Evaluation is subjective. Only the eye and ear of the composer determine what parents will be used to produce the next generation.

Populations in scientific GA research are large and the process continues through many generations. In such research the GA is often left to run its course without intervention. The fitness test(s) are defined at the beginning. Human preferences and esthetics are usually excluded. Results are assessed statistically or by evaluating success in solving a particular problem, the fitness test is modified to improve performance and the GA is run again. Sometimes, the goal is to search for a particular state or solution. Sometimes we simply want to see what happens given certain breeding methods and fitness criteria.

In the present musical experiments the population size is limited to a very small number on a grid of 9, 16, or 25 individuals. Like Dawkins' Blind Watchmaker, my GA's pause for human assessment at each generation. The toroidal structure of these grids and the MAX implementation of the GA are described in detail in my 1993 paper.

In order to facilitate evaluation and selection within each generation of the population, I constructed a display that shows each individual in a graphic form similar to the piano roll notations seen in many music programs. With this notation, I can evaluate each musical pattern by eye with respect to pitch contour, rhythmic density, and general aspects of texture. Clicking the mouse button within a box on the grid plays the corresponding pattern with synthesized sounds via MIDI. Shift-clicking selects an individual to be one of the parents for the next generation and option-clicking anywhere is the display causes breeding of the next generation.

The visualization is generalized to show "notes" independent of the model used to generate them. Input to the display is a table of numbers where each row represents a note with starting time, duration, pitch, and an index that differentiates each individual in the population. MIDI channels correspond to the instruments in an ensemble and are represented by color. Dynamics are represented roughly by three degrees of line thickness. A display with a randomized initial population looks like Figure 3.

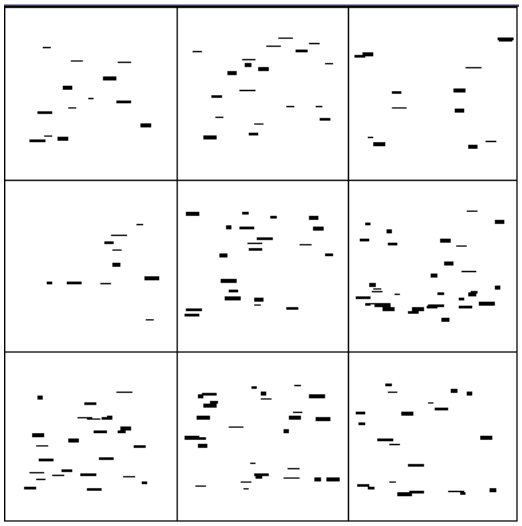


Figure 3. Display with randomized initial population.

4. Models

In the remainder of this paper I will describe a number of models that I have found particularly fruitful as objects for GA's. Each of the models is based on iterative processes that involve feedback from previous application of the same process. Although there are fine degrees of definition they are popularly grouped under the general name "fractals." Each model has initial conditions that include the starting value(s) for variables that will be iterated and values for constants or independent variables that control the general progress of the resulting patterns. Mapping of the contents of a genome onto initial values determines the evolution of the pattern. Each variation in the genome represents different initial conditions that will produce different results.

4.1. Dynamical Systems

Dynamical systems contain the rules that describe the way some entity (number or pattern) changes through time or iteration steps. In the three systems that follow, the rules are embodied in mathematical equations of one or more variables, some fixed and some changing with each iteration. My initial attraction to and interaction with these equations was in the visual domain. Since these have become popular objects for study there are many programs that implement them in computer graphics and facilitate empirical exploration. In my work with such systems I assume that equations that generate interesting images can be turned to make interesting sounds.

In these models the genomes are converted to real numbers that specify the initial conditions for the system and therefore determine how the system behaves through iteration. For example, the bit string, 00011100001110110101010111100001 has the decimal value 473650657 when

converted to a 32-bit integer. Dividing 473650657 by 2147483647 (the largest positive 32-bit integer) produces the real number, 0.22056077. Repeating this process with any bit string produces real numbers between -1.0 and 1.0 that can be conveniently scaled into any desired range.

4.1.1. Chaos

The logistic or "chaos" equation was popularized by James Gleick[9] although it represents only one of many functions that behave chaotically. It has become a favorite of computer musicians because of its simplicity of form and implementation and the wonderfully complex patterns it makes with a small number of variables. The equation takes the following form:

$$X = P*X*(1-X)$$

In this equation, X varies from 0.0 to 1.0 when P varies from 0.0 to 4.0. The graph of this equation (Figure 4) has become an icon of algorithmic composition.

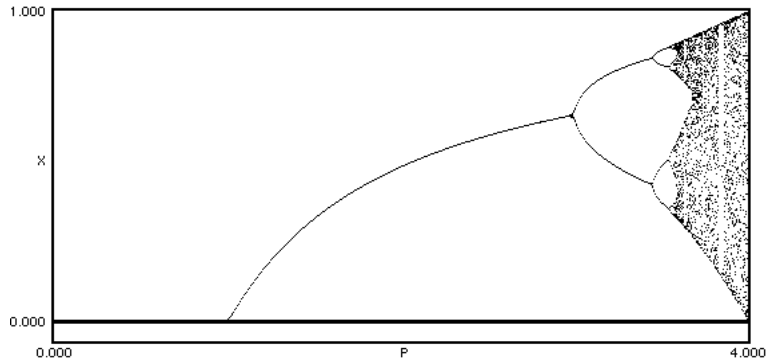


Figure 4. Graph of the logistic equation.

With each iteration of the equation, the previous output value for X is fed back to compute the next value. As P increases, the behavior of the equation generally increases in complexity. When P is between 0.0 and 1.0, the functions dies (all output gravitates toward zero). Between 1.0 and 3.0, the output values for X converge to values on a single curve that increases with the value of P. At 3.0 the function bifurcates resulting in a limit cycle of two X values. As P continues to increase, the interval between the two values increases until, at 3.5, a second bifurcation takes place to produce a four cycle. This behavior continues until P reaches approximately 3.6 where the cycles become so long and complex that they are difficult to follow. This is the first chaotic region. In spite of the complexity, every value of P (even to differences in the tenth decimal place) has a corresponding pattern that can be clearly recognized and repeated exactly. I have been using a program by Hall[10] to explore this equation.

Mapping the output of the logistic equation onto musical parameters creates patterns that range from simple alternation of two states to patterns that appear to be the result of sophisticated development and variation techniques. Real time control of the independent variable P allows a composer/performer to control the complexity of the music interactively.

---

[9] James Gleick. *Chaos: Making a New Science*, Viking Press, New York, 1987.

[10] Matthew Hall. "1-D Chaos Explorer," available from the Spanky Fractal Database on the World Wide Web (http://spanky.triumf.ca/pub/fractals/programs/MAC/CHAOS-EXPLORER-01B.HQX).

In my experiments with GA's and chaos, I convert the bit strings into real numbers that determine the initial values of X and P and the number of iterations of the logistic equation.

4.1.2. Wallpaper for the Mind

"Wallpaper for the Mind" is the name that Peterson[11] gave to a Macintosh program that implements Martin's two dimensional iterative equations:[12]

X = Y-SIGN(X)*SQRT(ABS(B*X-C))
Y = A-X

Figure 5 shows three images created by plotting X against Y on 1000 successive iterations of these equations.

Figure 5. Three examples from Martin's equations.

Like the logistic equation some variables (X and Y) undergo changes in value as the functions are iterated while others (A, B, and C) are given fixed values at the beginning of the process. Each set of initial values for these five variables produces an unique and repeatable output pattern. Once again, the GA determines these initial values and a run length for iteration. X and Y are mapped onto musical parameters to create sonomorphs for composer evaluation.

4.1.3. Pickover's Fancy

Clifford Pickover is a prolific writer on scientific curiosities who is particularly intrigued with mathematical pattern generation. One of his function sets has been implemented in a Macintosh program by Matthews.[13] Pickover calls them the "Latööcarfian equations[14]" and his books contain many images made with these equations along with the story of a fanciful alien society where the equation rests at the very core of the culture.

The equations are:

X = SIN(Y*B)+C*SIN(X*B)
Y = SIN(X*A)+D*SIN(Y*A)

---

[11]John W. Peterson. "Wallpaper for the Mind v 1.1," available from Info-Mac and its mirrors under the file name "app/wallpaper-for-the-mind-11.hqx."

[12]Barry Martin and Mike Mudge. "From chaos to beauty," Personal Computer World, November, 1987, pp. 118-122.

[13]John W. Matthews. "Cliff's World," available from the Spanky Fractal Database on the World Wide Web (http://spanky.triumf.ca/pub/fractals/programs/MAC/Cliff.hqx).

[14]Clifford Pickover. *Chaos in Wonderland,* St. Martin's Press, New York, 1994.

Once again, X and Y vary with each iteration while A, B, C and D set initial conditions. Like the logistic equation and Martin's equations, Latööcarfian equations exist in many variations. Images like those in Figure 6 are typical.

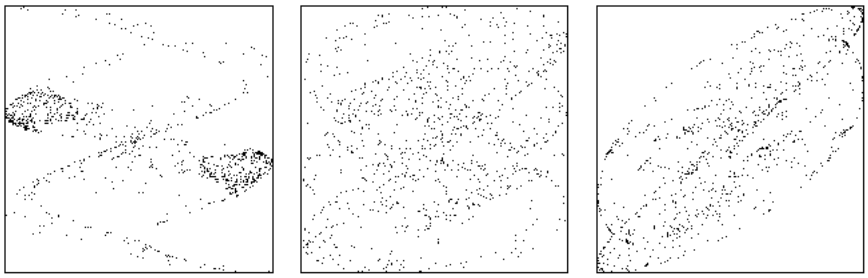


Figure 6. Three examples from Pickover's equations.

Like many of these models, dense clouds of points are required to bring the image to eye. Using a small number of points to generate only a few notes does not give the full impression of these images. However, the patterns, even with a small sampling, give interesting results. Perhaps the home for these images is in granular synthesis. The will be the subject of a future report on this work.

4.2. Iterated Function Systems

IFS can be used to compute complex images by randomized application of a small number of graphic transformations: scaling, translating, and rotation. The equations take the following form:

$$X = AX+BY+E$$
$$Y = CX+DY+F$$

The variables A-F are given by a vector that encodes each transformation. A-D are derived from the scaling factors for X and Y as well as the angle of rotation. E-F simply hold values for translation in X and Y. Usually, several transformation vectors make up an IFS. Each transformation has a weight that determines the frequency with which it will be used during the process of iteration. Peitgen[15] gives a particularly clear and detailed description of IFS and Barnsley[16] devotes most of a book to the subject.

[15]Hans-Otto Peitgen, Hartmut Jürgens, & Dietmar Saupe. *Chaos and Fractals: New Frontiers of Science*, Springer-Verlag, New York, 1992.

[16]Michael Barnsley. Fractals Everywhere, Academic Press, Boston, 1988.

The images in Figure 7 were produced with IFS. The image on the left is one of the classical realistic IFS examples, a fern.

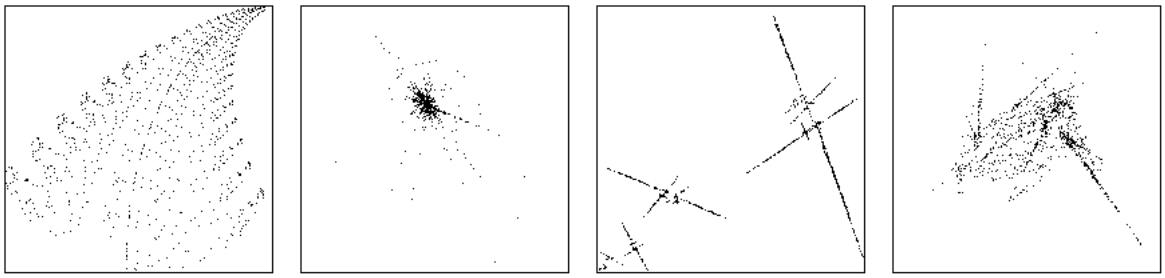


Figure 7. Four images from iterated function systems.

Figure 8 gives the values for the parameters in each of the four transformations that joined to form the fern. The final column, "P," gives the probability that the corresponding transformation will be chosen on a particular iteration.

	A	B	C	D	E	F	P
1	0.000000	0.000000	0.000000	0.172033	0.496139	-0.090510	0.010000
2	0.075906	0.312285	-0.257105	0.204233	0.494173	0.132616	0.075000
3	0.821130	-0.028405	0.029799	0.845280	0.087877	0.175709	0.840000
4	-0.023936	-0.356062	-0.323405	0.074403	0.470356	0.259738	0.075000

Figure 8. Iterated function system for computing a fern.

The remaining three images in Figure 7 were made by applying a GA to the selection of values for the variables A-F, and the probability, P.

I have been using a Macintosh program by Lee and Cohen[17] to investigate IFS.

[17]Kevin D. Lee and Yosef Cohen. "Fractal Attraction," Sandpiper Software, Box 8012, St. Paul, MN, 612-644-7395).

4.3. Lindenmayer Systems

A L-system is comprised of a set of functional symbols that can resemble a computer programming language. One implementation looks very much like LOGO in that its main "symbols" represent actions in turtle graphics. I will give only a hint here and refer the reader to Lindenmayer and Prusinkiewicz[18] for an elegant exposition of the exhaustive details.

In a simple L-system, the following symbols might appear:

F	move forward while drawing a line (pen down)
f	move forward without drawing a line (pen up)
+	turn right
-	turn left

The length of the line and the turning angles are specified in another part of the L-system, the "state." If we assume a state that includes a line length of 1 unit and an angle of 90 degrees, F+F+F+F will draw a 1 unit square.

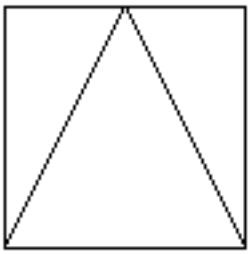
The third component of an L-system is a set of "production rules" that control string substitution in a series of iterations. The rule F -> F+F+F+F specifies that a single line is to be replaced by a square.

The final part of the L-system is the "axiom" or starting point. The following L-system can be used to produce the classic fractal, Von Koch's snowflake.

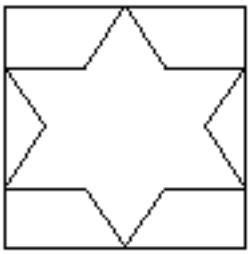
axiom	F++F++F
rule	F -> F-F++F-F
angle	60 degrees

[18]Aristid Lindenmayer & Przemyslaw Prusinkiewicz. *The Algorithmic Beauty of Plants*, Springer-Verlag, New York, 1990.

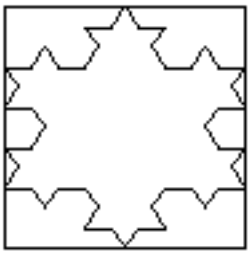
Figure 9 shows four generations of the image starting with the axiom, F++F++F (an equilateral triangle) and the character strings that specify how each image is to be drawn.



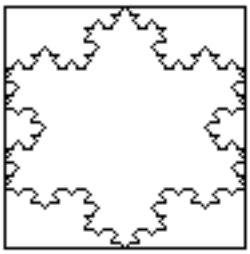
F++F++F



F-F++F-F++F-F++F-
F++F-F++F-F



F-F++F-F-F-F++F-
F++F-F++F-F-F-
F++F-F++F-F++F-F-
F-F++F-F++F-F++F-
F-F-F++F-F++F-
F++F-F-F-F++F-
F++F-F++F-F-F-
F++F



F-F++F-F-F-F++F-
F++F-F++F-F-F-
F++F-F-F-F++F-F-F-
F++F-F++F-F++F-F-
F-F++F-F++F-F++F-
F-F-F++F-F++F-
F++F-F-F-F++F-F-F-
F++F-F-F-F++F-
F++F-F++F-F-F-
F++F-F++F-F++F-F-
F-F++F-F++F-F++F-
F-F-F++F-F-F-F++F-
F-F-F++F-F++F-
F++F-F-F-F++F-
F++F-F++F-F-F-
F++F-F++F-F++F-F-
F-F++F-F-F-F++F-F-
F-F++F-F++F-F++F-
F-F-F++F-F++F-
F++F-F-F-F++F-
F++F-F++F-F-F-
F++F-F-F-F++F-F-F-
F++F-F++F-F++F-F-
F-F++F-F++F-F++F-
F-F-F++F-F++F-
F++F-F-F-F++F-F-F-
F++F-F-F-F++F-
F++F-F++F-F-F-
F++F-F

Figure 9. Four generations of Koch's snowflake.

Using less symmetric L-systems that arise from selection via GA's, images like the following emerge:

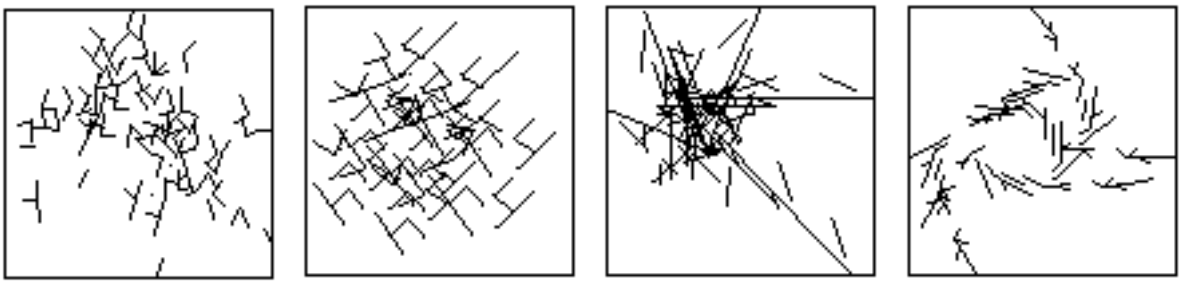


Figure 10. Images from algorithmically generated L-systems

Bourke's Macintosh program[19] has been useful in my investigations of L-systems in two and three dimensions.

4.4. Cellular Automata

CA is probably the most familiar of the models that interest me. It has been implemented many times on computers and its most famous two dimensional incarnation is as "the Game of Life."[20] CA's are also the closest to GA's in structure and iteration method. I will give only a brief description of a CA in one dimension.

The components of a one dimensional CA are the initial population, the neighborhood, and a set of rules that govern the character of successive generations. An initial population is usually a vector of random numbers chosen within a specified range. If the range is 0 to 1, the vector is binary and each element is a bit. The neighborhood is the number of individuals that will be used to determine the state (0 or 1) of an individual in the next generation. If the neighborhood is 3, a parent bit is summed with the bits on either side. There are four possible sums from 0 to 3. The rule is used to specify the transformation of each bit relative to the sum of its neighborhood. A rule like

0: 0
1: 1
2: 1
3: 0

specifies that for sums of 0 and 3, the child bit in the next generation should be 0. Likewise, sums of 1 and 2 produce 1 in the next generation. Here are the neighborhoods and the results of rule 0 1 1 0:

neighborhood:	000	010	100	001	110	101	011	111
result:	0	1	1	1	1	1	1	0

[19]Paul Bourke. "L-Systems," two dimensional version available from the Spanky Fractal Database on the World Wide Web (http://spanky.triumf.ca/pub/fractals/programs/MAC/LSYSTEM.SIT.HQX).

[20]Martin Gardner. "The Fantastic Combinations of John Conway's New Solitaire Game 'Life'," Scientific American, 223:4 (April, 1970), pp 120-123

Figure 11 shows four patterns in an initial population of 25 bit through 25 generations with a neighborhood of 3 and rules shown above each pattern.

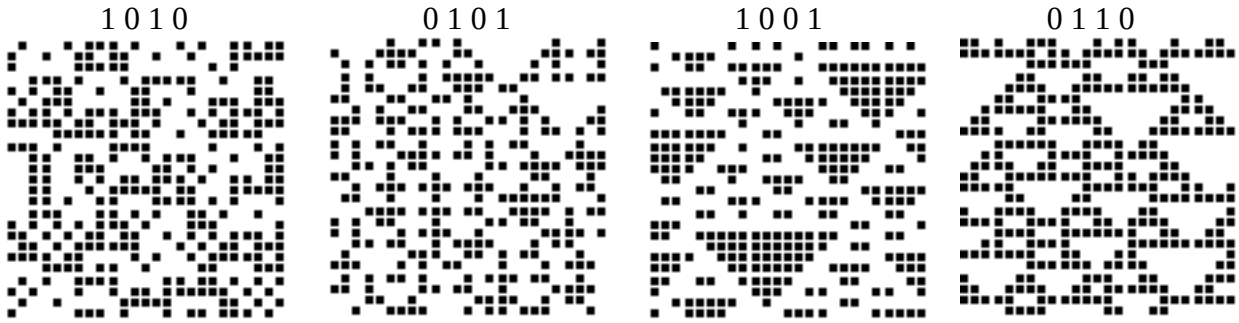


Figure 11. Four patterns from one dimensional CA.

Larger neighborhoods and a larger number of states produce greater complexity in the patterns.

I have found CA's particularly useful for generating rhythm and texture. By reading the array from left to right and top to bottom, the state of each point can be interpreted as an instrument number and pitches assigned in sequence to members of an ensemble. The patterns generated by the CA's create interesting rhythms and polyphonic interplay.

5. Conclusion

I have only been able to give a taste of the rich environment where I have been working. Still images without sound convey a very poor impression and much is yet to be done. You are encouraged to obtain the sound recordings and the software I have mentioned and experiment on your own.