J. McCormack: Interactive Evolution of L-System Grammars for Computer Graphics Modelling, in Complex Systems: From Biology to Computation, David Green and Terry Bossomaier (eds.), ISO Press, Amsterdam, pp. 118-130, 1993.

Interactive Evolution of L-System Grammars for Computer Graphics Modelling

Jon McCormack

Computer Science Dept.
Monash University, Clayton
AUSTRALIA
email: jonmc@cs.monash.edu.au

ABSTRACT

Evolution of Lindenmayer Systems (L-Systems) provides a powerful method for creating complex computer graphics and animations. This paper describes an interactive modelling system for computer graphics in which the user is able to "evolve" grammatical rules and surface equations. Starting from any initial L-System grammar the evolution proceeds via repeated random mutation and user selection. Sub-classes of the mutation process depend on the context of the current symbol or rule being mutated and include mutation of: parametric equations and expressions, growth functions, rules and productions. As the grammar allows importation of parametric surfaces, these surfaces can be mutated and selected as well. The mutated rules are then interpreted to create a three-dimensional, time-dependent model composed of parametric and polygonal geometry. L-System evolution allows with minimal knowledge of L-Systems to create complex, "life-like" images and animations that would be difficult and far more time-consuming to achieve by writing rules and equations explicitly.

1. INTRODUCTION

Computer Graphics and Computer Aided Design (CAD) systems allow for the creation of three-dimensional geometric models with a high degree of user interaction. Such systems provide an adequate paradigm for modelling the geometric and splined surfaces made by humans. Many organic and natural objects, however, have a great deal of complexity that proves difficult or even impossible to model with surface or CSG based modelling systems. Moreover, many natural objects are *statistically self-similar*, that is they appear approximately the same but no two of the same species are identical in their proportions.

L-Systems have demonstrated an ability to model natural objects, particularly botanical and cellular models [1]–[7]. L-Systems work on the principle of *data-base amplification* which allows complexity to be generated by the repeated application of rules. The resultant output can be many orders of magnitude larger than the input – in much the same way as a complex biological organism's "blueprints" are stored in the relatively small amount of information contained within its DNA.

However, despite the flexibility and potential of L-Systems, (and procedural models in general), they are difficult for the non-expert to use and control. To create specific models requires much experimentation and analysis of the object to be modelled. Even an experienced user can only create models that are understood and designed by the human creator.

1.1 L-Systems

Lindenmayer Systems (L-Systems) are a class of string-rewriting mechanisms, originally developed by Lindenmayer [3] as a mathematical theory of plant development. The original emphases was on plant topology – neighbourhood relations between plant cells or higher structures. They arose from an interest in string rewriting based on Chomsky's work on formal grammars in the late 1950's [8]. The main difference between Chomsky grammars and L-Systems is that in Chomsky grammars productions are applied sequentially as opposed to L-Systems where productions are applied in parallel.

Hogeweg and Hesper [1] studied *propagating*, *deterministic* bracketed 2L-Systems to produce a rich variety of tree structures, however their graphical results were limited to 2D black and white line drawings.

The use of L-Systems for computer graphics modelling was latter developed by Smith [9] who coined the term *graftals* and made reference to the "fractal" nature which can form a part of rewriting grammars. Prusinkiewicz created many highly realistic models from a wide class of L-Systems. He showed the technique particularly useful in modelling herbaceous (non-woody) species, by extending grammars and providing pre-defined surfaces and advanced turtle interpretations, rendering models using a ray tracing technique [4, 5, 6, 7].

1.1 Evolution

Natural evolution, first proposed by Charles Darwin [10] provides a theory for the development of species. Through a process of natural selection organisms as complex as humans have evolved. The *genetic algorithm*, proposed by Holland [11] follows this evolutionary paradigm and provides a method for searching very large spaces for an optimal solution. Genetic algorithms have

been used to solve a number of problems in design, optimisation and fitness [12, 13, 14, 15].

In his book, *The Blind Watchmaker* [4], Richard Dawkins demonstrated simulated evolution by evolving *"biomorphs"* – simple two-dimensional structures resembling organic creatures created from simple sets of genetic parameters. The survival of each generation of biomorphs is selected by the user who evolves features according to their personal selection.

Other applications of the genetic algorithm to image and object generation include Todd and Latham who have evolved computer sculptures using CSG techniques [17]. Sims has evolved procedural models to create branching structures, textures, parametric surfaces and dynamic systems [18, 19].

1.3 Genetic Terminology

Genetic terminology is also applied in the case of simulations. The *genotype* is the genetic information that contains the codes for the creation of the individual. In organic life this is usually the DNA of the organism. In the case of simulations the genotype can be a string of digits or parameters. In the case of L-Systems grammars, it is the rules and parameters.

The individual, object or system created from the genotype is known as the *phenotype*. The process of *expression* (genome to phoneme) usually generates the complexity from the relative simplicity of the genotype.

Fitness of phenotypes is determined by *selection*. In a real world environment fitness of an organism determines it's ability to pass on it's genes from generation to generation – survival. In the real world, there are many factors which influence an organisms fitness to the task of survival. In simulated genetic systems, the fitness can be determined either automatically, by a defined *fitness function,* or, as is the case with this work, selected implicitly by the user.

In a natural situation, genetic variation is achieved through the process of reproduction. In the system described, "child" genotypes are created by mutation of the parent genotype. Such mutations cause different phenotypes to result. By a repeated process of selection by the user and mutation by the computer aesthetic characteristics of the resultant forms can be optimised (the genetic algorithm can be thought of as an optimisation process).

The next section looks at the syntax and structure of L-Systems used. Section 3 examines the mutation process, section 4 the interactive evolution process and section 5 discusses the implementation. Finally section 6 presents a summary of results and possibilities for further work.

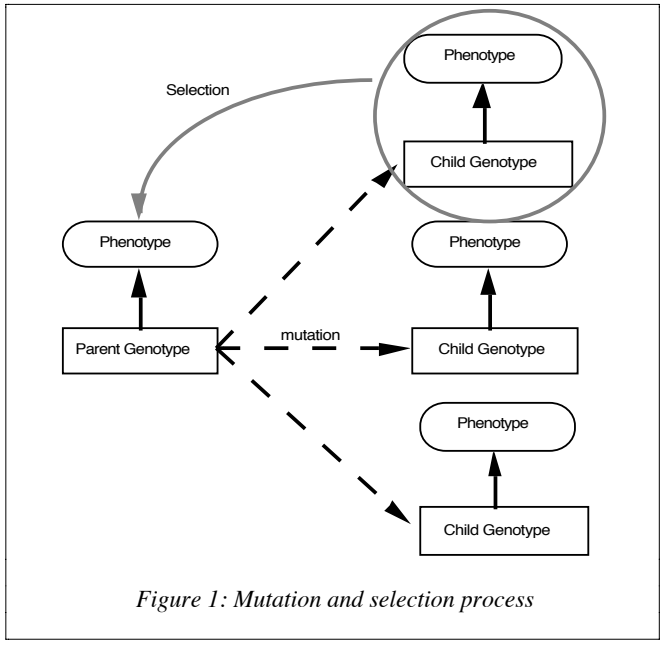

Figure 1: Mutation and selection process

2. L-SYSTEMS

A brief explanation and definition of L-Systems is presented here. For a fuller, more formal definition and explanation of L-Systems, the reader is referred to [4].

Basically, L-Systems consist of a finite set of *letters* (collectively called the *alphabet*). Letters are arranged in arbitrary length sequences to form *strings*. Each letter is associated with a rewriting rule. For example consider the letters a and b and their associated rules $a \rightarrow ab$ and $b \rightarrow a$. These rules mean (respectively) that each occurrence of the letter a in a string is to be replaced by the sequence ab and that letter b is to be replaced by a. Substitution of letters takes place in parallel across the entire string. The initial string of letters is called the *axiom*. Using the above rules as an example and applying them to the axiom a, we obtain the following sequence of strings each time the rules are applied:

a
ab
aba
abaab
abaababa
abaababaabaab

As can be seen the string length can grow quickly after only a few rewrites. In order to convert strings into geometric models a *turtle interpretation* [21] is applied. The concept is based on the idea of an imaginary turtle that walks, turns and draws according to instructions given. At any time the turtle has a current position in 3-space and a heading vector (the forward direction of movement). Individual letters in a string are treated as commands. Different letters change position or heading, record vertices in a polygon, apply pre-defined surfaces to the current position and orientation, change colour, etc. *Bracketed* L-Systems provide two special letters (usually square brackets: *[,]*) which push and pop (respectively) the current turtle position, heading and possibly other attributes. This feature allows for the generation of branching structures.

2.1 L-Systems implementation

This paper describes a modelling system developed for creating three-dimensional animated models using L-Systems. The foundation of the system is a parser which takes a set of rules and parameters and creates a time dependent geometric model as its output. The system permits timed, parametric deterministic (D0L-) and non-deterministic stochastic (0L-) Systems, both of which are context-free. Currently, context sensitive 1L and 2L-Systems are not implemented. Many of the results obtained with 1L and 2L Systems can be achieved with parametric 0L-Systems [6].

Parametric L-Systems allow an arbitrary number of parameters to be associated with each letter. These parameters can be used during interpretation of the produced string. For example the letter *F* is interpreted as "draw a line in the current direction". With a parameter, the distance of the move forward can be controlled. (i.e. *F(3.0)* means draw a line 3.0 units long).

Stochastic L-Systems allow for the simulation of random feature variation in a model. For example the rules:

$$a \rightarrow^{0.5} ab$$

$$a \rightarrow^{0.5} bb$$

mean that the letter *a* has will be replaced by the string *ab* with a probability of 0.5 or by *bb* also with a probability of 0.5. Stochastic rules for a single letter must total 1.0 exactly. A rule without a probability parameter implies a probability of 1.0.

Timed L-Systems give each letter a life time over which it exists. An example of timed rules is written:

$$(a,5) \rightarrow (a,0.5)(b,0)$$

This means that the letter *a* has a life of 5 seconds. When *a* reaches an age of 5 seconds the transition rule is applied. A new letter *a* is born with age 0.5, and a new letter *b* is born with age 0.0. Age values on the right side of a rule specify terminal age, age values on the left side of the rule specify birth age. Birth ages should never exceed death ages. Associated with each letter is a *growth function* which controls the behaviour of the letter over its lifetime. Growth functions are written: *g(a,t,T)* - *g* is the growth function for letter *a* at time *t*, with overall life time *T*. Growth functions are usually expressed as some mathematical function. Figure 2 shows an example image generated using these L-System techniques.

Figure 2: An L-System model

3. MUTATION OF L-SYSTEM RULES

In order for the structure and form of an L-System model to change its rules and parameters must be changed. Small changes in genotype are usually preferred so as not to completely alter the form on which it is based. However, for radical change larger amounts of mutation are required.

There are three basic areas to which mutation can be applied:

1. Mutation of rules and letters.

2. Mutation of parameters and parametric expressions.

3. Mutation of growth functions and letter ages.

In addition, pre-defined surfaces may themselves be mutated, for example using techniques described by Sims [18].

3.1 Rule Mutation:

For each production letters and rules are mutated. The probability of each mutation is specified separately. The types of mutations possible is listed below:

- A letter in a production may be removed
- A new letter may be added to a production
- A letter may change to another, different letter

The above three rules work either on previously defined letters or new letters may be created. In addition it is sometimes necessary to disallow mutation of some letters whose purpose is some kind of control (such as output resolution), or to limit a search space.

In addition to individual letter changes, rules may be created and deleted:

- A rule can split into a stochastic rule. For example the rule $a \rightarrow b\ b\ c$ may mutate to become $a \rightarrow^{0.22} b\ c\ c$ and $a \rightarrow^{0.78} b\ c$. The new rule is a mutated version of the old rule.
- A new rule can be created. This rule must contain a previously defined letter. If all previously defined letters already have a rule then this mutation can't take place.
- An existing rule may be deleted. If it is stochastic then the previous rules gain in probability in equal amounts equal to the probability of the deleted rule.
- The probability of a stochastic rule may change. The addition or difference redistributes probabilities over all the other stochastic rules involving that letter.

3.2 Parametric Mutation

For the sake of efficiency and ease of implementation, letters may not gain or loose parameters during mutation. New letters may be created, however with the default number of parameters, or if no default exists, a random number of parameters. Rules involving parametric letters on the left hand side may split as follows:

- Rules involving letters with parameters may spit on conditions. For example:

$$a(length) \rightarrow a(length \times 2.0)\ b(length^{2.0})$$

could become:

$$a(length) : length > 10.0 : \rightarrow a(length \times 2.0)\ b(length^{2.0})$$

and

$$a(length) : length < 10.0 : \rightarrow a(length \times 2.0)\ c(length / 2.0)$$

Again the new rule is a mutation of the old.

Parameters on the right side of rules are expressions. The are parsed into a tree structure and executed during application of productions. Each node on the expression tree can be recursively subject to mutation by the following rules:

- If the node is a variable is can mutate to another variable.
- If the node is a constant it is adjusted by the addition of some random amount.
- If the node is an operator it can mutate to another operator: i.e. $x + 5$ becomes $x / 5$.
- A node may mutate to a new expression.
- Nodes may become the arguments to a new expression: i.e. $x + 5$ becomes $y * (x + 5)$.
- An expression may reduce to one of its operands: i.e. $x + 5$ becomes x.

In the current implementation only simple arithmetic operators are supported (addition, subtraction, division, multiplication, negation and power). Other functions (such as trigonometric functions) could be added if required. Parameters on the left side of rules do not mutate as this serves no useful purpose.

3.3 Growth Function and letter age mutation

If a letter is timed it must have a birth and terminal age. Both these values must be constants. Both constants can be mutated by the addition of a random value. The range of the random value is usually proportional to the size of the constant.

The growth function for a timed letter controls the behaviour of that letter over its growth period. It is expressed in the form $g(L,a,t)$, where L is the letter, a is the current age and t is the terminal age. An simple example growth function could be: $g(leaf,a,t) = a/t$ which is a simple liner function which ranges from 0 to 1 as the letter ages. Since growth functions are expressions their internal and external construction is the same as for a letter's parameter expressions. Thus the mutations are identical to those described for expressions in the previous section.

3.4 Mutation Probabilities

Different sorts of mutations occur with different probabilities. An important part of evolving structures is correctly setting mutation probabilities. For example, it is better to set rule mutation probabilities to maintain or slightly shrink current rule size. If rule mutation is biased towards adding rules and/or letters then genotypes tend to become larger without necessarily creating greater fitness. Large rules take longer to parse and in general, take longer to generate. This does not stop rules *evolving* complexity by selection.

Mutation probabilities can be changed interactively by the user during the evolutionary process. This provides a useful aid when one appears to be approaching the desired result and wishes to limit mutations to specific areas.

4. THE INTERACTIVE PROCESS

To evolve forms interactively we begin with a parent genotype - namely a set of rules. The rule set may be empty or the result of some previous mutation process. A list of external surfaces to be used is also supplied. The parent rule set is then mutated according to probabilities specified. After mutation the rules are parsed and applied to a specified level. The user may interrupt the process at any time. The process will be automatically interrupted by the software if the computation time exceeds a specified limit. In traditional uses of the genetic algorithm, population sizes can be large as the selection process is automatic, however in the case of this type of system, selection is based on the vague human notion of aesthetics. This is one reason why the population size in this case is limited. A much more overpowering reason is the space on the screen to display phenotypes and the computation time involved in generating large populations.

Usually around 16 mutations per generation are performed. This provides a trade off between generation time and variety. With increased computation (or more patience) larger populations can be created. The screen size also limits the number of phenotypes that can be displayed an manipulated. The parent phenotype is displayed in the upper left-hand corner of the screen followed by its mutated children (figure 3).

At the conclusion of the mutation and generation process, the user may interactively manipulate and examine the phenotypes in space and over time. At some stage the user decides which phenotype is the most suitable and this becomes the new parent. Selecting the existing parent provides more mutations if none of the current generation are deemed suitable. At any time the genotype (rule set) for a particular phenotype may be saved to disk as a ASCII file. This file can be latter used as a parent for further mutations or generated with a higher degree of accuracy for final output.

The mutation/generation/selection process is repeated until a satisfactory form is achieved or the user runs out of time or patience.

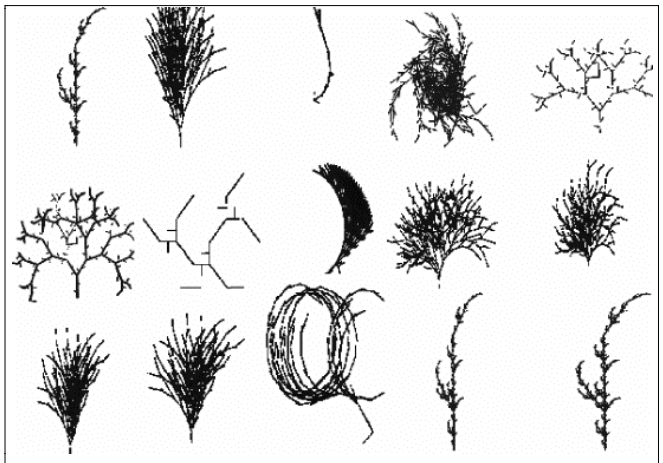

Figure 3: Parent (top left) and 14 mutations

5. IMPLEMENTATION

The system has been implemented in the C programming language, on Silicon Graphics workstations under the IRIX operating system. The workstations high speed graphics performance permits real time previewing interpreted strings. Since models evolve over time this is previewed as well. A display sub-system accepts graphics primitives on an object time basis. While the models are defined continuously over time they are usually sampled at regular discrete moments in time for playback. For the case of video animation this is either 25 times per second for frame rendered animation or 50 times per second for field rendered animations.

The generation of several seconds of animation for 16 or more phenotypes can be quite time consuming and thus for preview purposes the samples are taken at wider intervals. For complex geometries, simplified representations of complex surfaces can be substituted to increase display speed.

Once generated, the models are stored in the workstations graphics memory as display lists. This enables to user to examine the generated models from any position or angle, play the development either forward or backward. Once the most suitable phenotype is picked the display lists are cleared and a new set of phenotypes are created. At any time the user can save the L-system rules for a particular resultant phenotype to a human readable file. This can be used to regenerate the model at a latter time or as a genotype for further mutation and evolution work.

Even simple rules can evolve highly complex models which can not be displayed in real time by the workstation graphics hardware. An option exists to save geometry to disk for rendering by the *Advanced Visualizer* system. Advanced Visualizer is an animation and rendering package from Wavefront Technologies [21]. The software rendering of these models has a number of advantages over the simplistic real time display of the workstation hardware. It allows advanced shading and illumination models (such as ray-tracing), texture, bump and transparency mapping and as well as anti-aliasing and spectral sampled colour space definition. This extra quality comes at a price, however, as rendering of even a single frame can take 20-30 minutes or more (including model generation time). Integrating with the animation system was a key goal in development and the system takes advantage of the features of a powerful existing animation and rendering system.

To date the system has been used to produce a variety of successful works, both still and animated [22, 23, 24]. Figure 4 shows some of the results achieved with the system. The emphases on the system has been one of a sculptural tool for modelling and observing growth, form and behaviour.

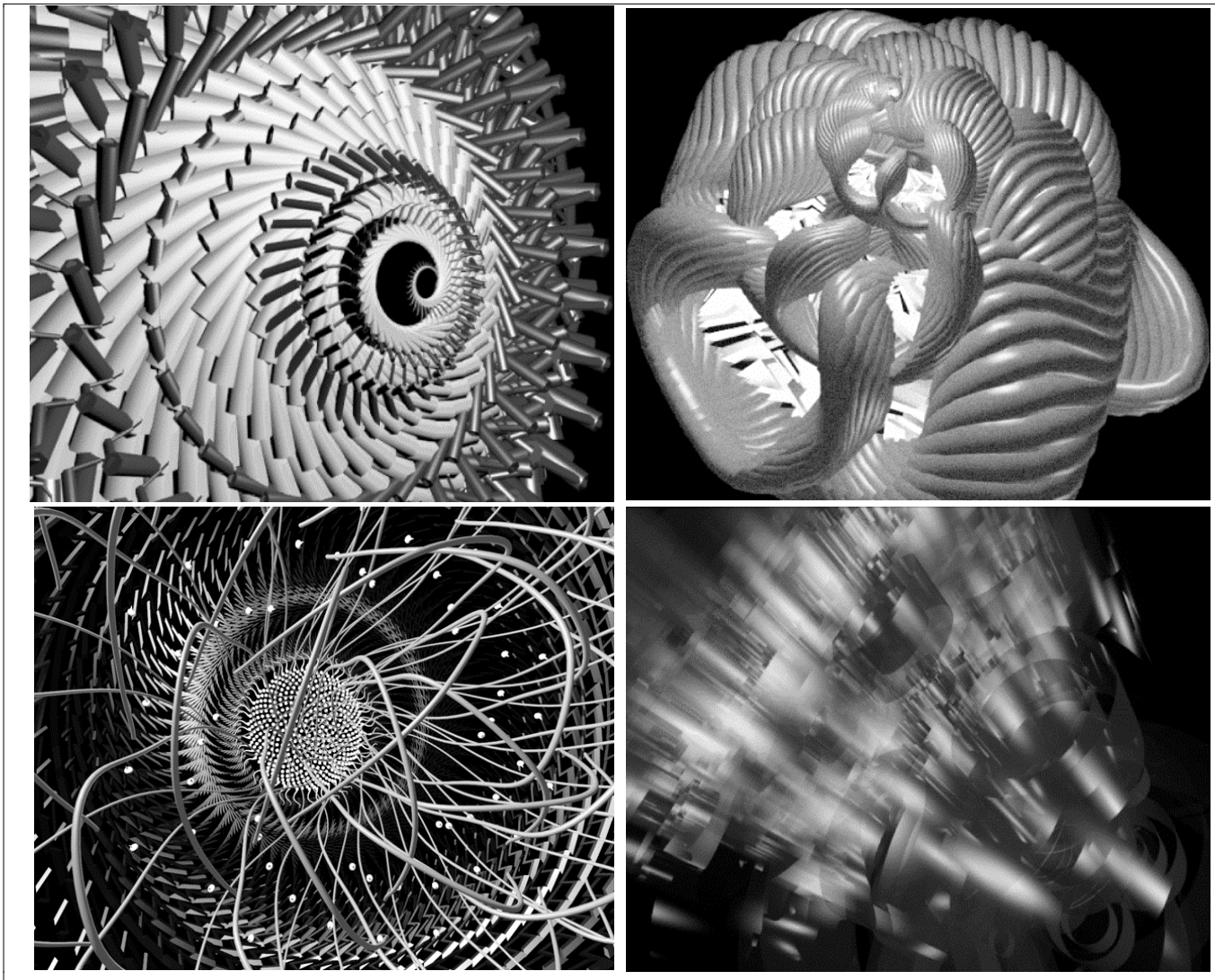

Figure 4: Example models generated with the system

6. CONCLUSION & FURTHER WORK

The interactive evolutionary technique provides advances in two areas: firstly, it enables a synergy between human and machine. Many of the models and results created by this technique would be extremely difficult, if not impossible to have created by explicit writing of rules.

Secondly, the technique allows novice users to create highly sophisticated models with little or no knowledge of the underlying processes involved. Users do not need to learn or understand how L-Systems work or write rules, a simple aesthetic selection is the all that is required.

One current limitation of the technique is the speed of generation of phenotypes. Sixteen or more animated models must be generated in a relatively short space of time in order to genuinely call the system "interactive". Simplified surfaces and wire-frame representation help to minimise display time. One would expect as workstation hardware performance increases larger and more complex populations could be created.

Currently context sensitive *n*L-Systems are being implemented which will allow even greater flexibility in modelling possibilities. Context sensitive rules can also be mutated, however in most cases this can significantly change the resultant phenotype. Proper mutation rules and probabilities is an area still under investigation.

As a modelling language, L-Systems have large scope in the type of models they can represent. However a fundamental limitation of the current system that the representation via L-System letters is highly topological. Several extensions to allow greater control over surfaces, geometry and texture are currently being tested. Greater control over constraints and physical effects such as light and weather is also needed.

7. ACKNOWLEDGMENTS

This work was produced with the assistance of the Australian Film Commission. Wavefront Technologies provided their *Advanced Visualizer* software. The Victorian Centre for Image

Processing and Graphics (C.I.P.A.G.) at Monash University provided animation and computing facilities.

8. REFERENCES

[1] Hogeweg, P. & Hesper, B. (1974) "A model study on biomorphological description", *Pattern Recognition*, **6**, pp 165–179.

[2] Honda, H., Tomlinson, P. B. & Fisher, J. B. (1982) "Two geometrical models of branching of botanical trees", *Annals of Botany,* **49,** pp 1–11.

[3] Lindenmayer, A. (1968), "Mathematical Models for Cellular Interaction in Development, Parts I and II", *Journal of Theoretical Biology,* **18**, pp 280–315.

[4] Prusinkiewicz, P. & A. Lindenmayer (1991), *The Algorithmic Beauty of Plants*, Springer Verlag.

[5] Prusinkiewicz, P. (1987) "Applications of L-Systems to Computer Imagery", In H. Ehrig, M. Nagl, A. Rosenfeld, and G. Rozenberg, eds., *Graph Grammars and their application to Computer Science; Third International Workshop,* pp 534–548. Springer-Verlag, Berlin.

[6] Prusinkiewicz, P & Hanan, J.(1989), *Lindenmayer Systems, Fractals and Plants*, Lecture Notes in Bio-mathematics, **79**. Springer-Verlag, Berlin.

[7] Prusinkiewicz, P, Lindenmayer, A. & Hanan, J. (1988) "Developmental Models of Herbaceous Plants for Computer Imagery Purposes", *Computer Graphics,* **22(4)**, August 1988, pp 141–150.

[8] Chomsky, N. (1956), "Three models for the description of Language", *IRE Transactions on Information Theory,* **2(3)** pp. 113–124.

[9] Smith, A. R. (1984) "Plants, Fractals and Formal Languages", *Computer Graphics*, **18(3)**, July 1984, pp 1–10.

[10] Darwin, C (1859), *The Origin of Species*, Penguin Paperbacks.

[11] Holland, J. H.(1975), *Adaptation in Natural and Artificial Systems,* Ann Arbor MI: University of Michigan Press.

[12] Grenfenstette, J. J. (1987), *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms,* Hillsdale, New Jersey, Lawrence Erlbaum Associates.

[13] Grenfenstette, J. J. (1985), *Proceedings of the First International Conference on Genetic Algorithms and Their Applications,* Hillsdale, New Jersey, Lawrence Erlbaum Associates.

[14] Rawlins, G. J. E. (ed) (1991), *Foundations of Genetic Algorithms*, Morgan Kaufmann.

[15] Goldberg, D.E. (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley.

[16] Dawkins, R. (1986), *The Blind Watchmaker*, Harlow Logman.

[17] Todd, S.J.P., & Latham, W. (1992), *Evolutionary Art and Computers*, Academic Press.

[18] Sims, K. (1991) "Artificial Evolution for Computer Graphics", *Computer Graphics*, **25(4)** July 1991, pp. 319–328.

[19] Sims, K. (1990) "Interactive Evolution of Dynamical Systems", *Proceedings of the First European Conference on Artificial Life*, Paris, Dec. 11–13, 1990.

[20] Abelson, H. & A. A. diSessa (1982), *Turtle Geometry,* MIT Press.

[21] Wavefront Technologies, Inc. (1992), *Advanced Visualizer Users Manuals,* Wavefront Technologies Inc. Santa Barbara, CA.

[22] McCormack, J. (1992), "Flux", *ACM Siggraph Video Review*, Issue 85, 1992.

[23] McCormack, J. (1992), "Bloom" & "Shell", *ACM Siggraph Stereoscopic Slide Set*, 1992.

[24] McCormack, J (1992), "Turbulence", Interactive video disc on Artificial Life (in progress).