Evolving a Logo Design using Lindenmayer Systems, Postscript & Grammatical Evolution

Michael O'Neill and Anthony Brabazon

Abstract— **We present an application of Grammatical Evolution to the exploration of Lindenmayer systems. The resulting L-systems are expressed in the Postscript language, and as such a Postscript grammar was provided as input to the Grammatical Evolution algorithm. The system takes the form of an interactive evolutionary algorithm, with a human-in-the-loop acting as the fitness function for the generated L-systems. The motivation for this research was to evolve a logo for the UCD Natural Computing Research & Applications group, and to this end the study was a success.**

I. Introduction

EVOLUTIONARY Computation has demonstrated much potential for Evolutionary Design (ED) producing solutions that are competitive, and even superior to those developed by human experts resulting in patentable inventions [1], [2], [3]. As such, the real world application domain of Design (in particular Analog Circuit Design [1]) has been a proving ground for the abilities of an artificial evolutionary process and has arguably led to the first routinely, human-competitive form of Machine Learning. ED is a challenging domain as it is often dynamic in nature due to the ever changing preferences of the human users that judge the aesthetic qualities of a design during evolution.

The combination of an EA coupled to a Grammatical, Developmental Representation (*Design Language*) is a particularly powerful and novel departure in recent years [4]. Research at this nexus of EC and a Grammatical Representation include GENRE and Genr8 amongst others [4], [6], [7], [8], [9]. As is the case in this study much of this research in grammar-based Genetic Programming and in more traditional approaches to Genetic Programming (e.g., [10]) has been undertaken using L-systems.

Aristid Lindenmayer developed what are now known as L-systems in 1968 to model the development of cells [11]. The L-systems are a form of grammar, similar to Chomsky grammars, with the difference that Lindenmayer grammars can apply production rules in parallel. There have been a number of applications of genetic programming approaches to the generation of various types of L-systems most notably the Hemberg-Extended Map L-systems for 3-D surface generation [6], and using Grammatical Evolution to design fractal curves with a specific dimension [12].

A convenient way to generate and display L-systems is to use the Postscript language, and in this study we use Grammatical Evolution [13] to evolve Postscript programs that represent aesthetically pleasing 2-D L-systems.

Michael O'Neill and Anthony Brabazon are with the Natural Computing Research & Applications Group, University College Dublin, Ireland (e mail: m.oneill@ucd.ie, anthony.brabazon@ucd.ie).

In the following Section II we provide a brief introduction to Grammatical Evolution. Section III details the grammar used to generate L-systems in PostScript, and results of the interactive evolution are presented in Section IV. Conclusions and a number of items for Future Research are outlined in Section V.

II. Grammatical Evolution

Grammatical Evolution (GE) is an evolutionary algorithm that can evolve computer programs in any language [13], [14], [15], [16], [17], [18], and can be considered a form of grammar-based genetic programming. Rather than representing the programs as parse trees, as in GP [19], [20], [21], [22], [1], a linear genome representation is used. A genotype-phenotype mapping is employed such that each individual's variable length binary string, contains in its codons (groups of 8 bits) the information to select production rules from a Backus Naur Form (BNF) grammar. The grammar allows the generation of programs in an arbitrary language that are guaranteed to be syntactically correct, and as such it is used as a generative grammar, as opposed to the classical use of grammars in compilers to check syntactic correctness of sentences. The user can tailor the grammar to produce solutions that are purely syntactically constrained, or they may incorporate domain knowledge by biasing the grammar to produce very specific forms of sentences.

BNF is a notation that represents a language in the form of production rules. It is comprised of a set of non-terminals that can be mapped to elements of the set of terminals (the primitive symbols that can be used to construct the output program or sentence(s)), according to the production rules. A simple example BNF grammar is given below, where `<expr>` is the start symbol from which all programs are generated. These productions state that `<expr>` can be replaced with either one of `<expr><op><expr>` or `<var>`. An `<op>` can become either `+`, `-`, or `*`, and a `<var>` can become either `x`, or `y`.

```
<expr> ::= <expr><op><expr> (0)
         | <var>            (1)
  <op> ::= +                (0)
         | -                (1)
         | *                (2)
 <var> ::= x                (0)
         | y                (1)
```

The grammar is used in a developmental process to construct a program by applying production rules, selected by the genome, beginning from the start symbol of the grammar.

In order to select a production rule in GE, the next codon value on the genome is read, interpreted, and placed in the following formula:

$$Rule = Codon\ Value\ \%\ Num.\ Rules$$

where % represents the modulus operator.

Given the example individuals' genome (where each 8-bit codon is represented as an integer for ease of reading) in Fig.2, the first codon integer value is 220, and given that we have 2 rules to select from for `<expr>` as in the above example, we get $220\ \%\ 2\ =\ 0$. `<expr>` will therefore be replaced with `<expr><op><expr>`.

Beginning from the left hand side of the genome, codon integer values are generated and used to select appropriate rules for the left-most non-terminal in the developing program from the BNF grammar, until one of the following situations arise: (a) A complete program is generated. This occurs when all the non-terminals in the expression being mapped are transformed into elements from the terminal set of the BNF grammar. (b) The end of the genome is reached, in which case the *wrapping* operator is invoked. This results in the return of the genome reading frame to the left hand side of the genome once again. The reading of codons will then continue unless an upper threshold representing the maximum number of wrapping events has occurred during this individuals mapping process. (c) In the event that a threshold on the number of wrapping events has occurred and the individual is still incompletely mapped, the mapping process is halted, and the individual assigned the lowest possible fitness value. Returning to the example individual, the left-most `<expr>` in `<expr><op><expr>` is mapped by reading the next codon integer value 240 and used in $240\ \%\ 2\ =\ 0$ to become another `<expr><op><expr>`. The developing program now looks like `<expr><op><expr><op><expr>`. Continuing to read subsequent codons and always mapping the left-most non-terminal the individual finally generates the expression `y*x-x-x+x`, leaving a number of unused codons at the end of the individual, which are deemed to be introns and simply ignored. Fig.1 draws an analogy between GE's mapping process and the molecular biological processes of transcription and translation. A full description of GE can be found in [13].

III. Postscript Logo Design Grammar

To evolve a logo design for the UCD Natural Computing Research & Applications group we wished to use a bio-inspired process that would complement the philosophy of the group. We also wished that the design itself would reflect the natural world to some extent. As L-systems were originally adopted to model cell growth and plant development we considered this developmental language appropriate. The input grammar adopted in this study is presented below, and contains the rules to generate an L-system grammar. The L-system must then be expanded to produce a design that can be evaluated by the human user.

```
<lsys> ::= <numrepeats> "{dup} repeat" <rules>"} \n
           if pop \n
           } def \n
           /rotangle"<rotangle> " def \n"

<rules> ::= <rules> <rules> <rules> <rules>
          | <Fcomplex>
          | <Xcomplex>
          | <fun>
          | <rotateop>

<rotateop> ::= + | -

<fun> ::= F | X

<Fcomplex> ::= <rotateop> F
             | <rotateop> F <rotateop> F

<Xcomplex> ::= <rotateop> X
             | <rotateop> X <rotateop> X

<rotangle> ::= 5 | 10 | 15 | 20 | 25 | 30
             | 35 | 40 | 45 | 50 | 55 | 60
             | 65 | 70 | 75 | 80 | 85 | 90

<numrepeats> ::= ?
```

The above grammar generates an L-system coded in the Postscript language [27], [28]. All the evolved grammars in this study take the form:

```
S -> X
X -> ?
```

where the angle of rotation of the turtle graphic (`rotangle`) is evolved according to the non-terminal `<rotangle>`, and the `order` (depth of expansion of the L-system) of the system is 3 by default, however, the interesting L-systems evolved in this study were allowed to divide further and are presented later on. To complete a valid postscript file the header and footers presented in Fig 3 are used to wrap the evolved L-System. A number of PostScript operators are provided including `dup` and `pop` which duplicate and pop the top item on the stack respectively, the conditional operators `if` and `ne` (*not equals*). The forward slash, `/`, is used to denote user-defined variable and procedure names along with the definition operator (`def`). `rlineto`, `newpath` and `moveto` are path construction operators that create a path, clear the current path and move to a specific cartesian coordinate respectively. The painting operator `stroke` is used to paint the current path with the current color and line width.

An unusual feature of this grammar is that the `<numrepeats>` non-terminals right-hand side is undefined at the start of the mapping process. Only after a first pass through the grammar, starting from `<rules>`, is completed can the the terminal symbol for `<numrepeats>` be determined by parsing over the sentence arising from `<rules>`. During the parse we count the number of occurences of procedure calls in the evolved L-System (i.e., the occurences of `X` and `F` in this case). `F` is a user-defined procedure that creates a path for a line, and the evolved procedure `X` determines how `F`'s are combined and rotated during the expansion of the L-system.

In this approach we therefore provide a grammar as input to GE, which represents another grammar (the L-system). The evolved L-system must then itself be generated in order to evaluate it. This meta-grammar approach (the

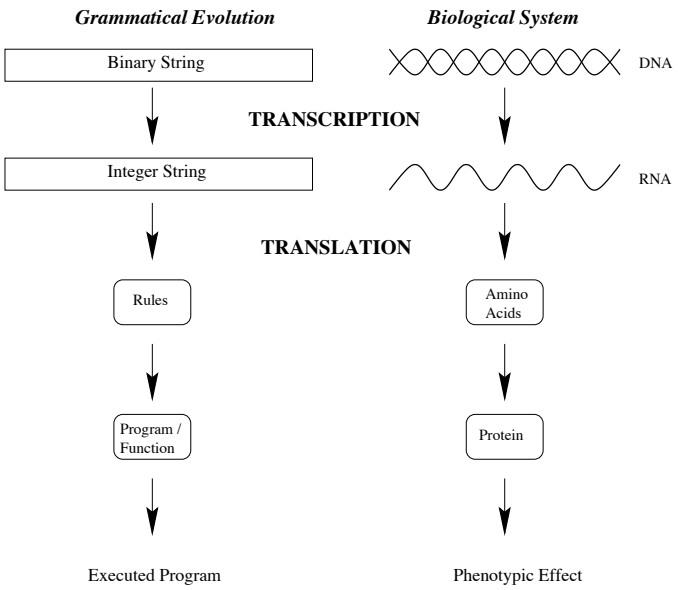

Fig. 1. A comparison between Grammatical Evolution and the molecular biological processes of transcription and translation. The binary string of GE is analogous to the double helix of DNA, each guiding the formation of the phenotype. In the case of GE, this occurs via the application of production rules to generate the terminals of the compilable program. In the biological case by directing the formation of the phenotypic protein by determining the order and type of protein subcomponents (amino acids) that are joined together.

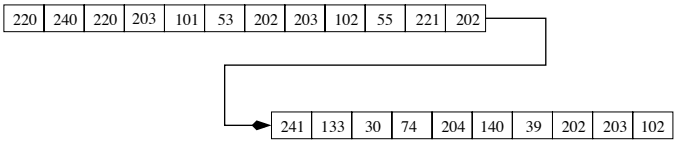

Fig. 2. An example GE individuals' genome represented as integers for ease of reading.

```
Postscript Header:
%!PS
/order 3 def                          %set the systems order
/START { X } def                      %start definition of X
/X {                                  %let evolution fill in
dup 0 ne                              %the rest
{1 sub

Postscript Footer:
/F {                                  %define F - draws a line
0 eq { 10 0 rlineto } if
} bind def
/- { rotangle neg rotate } bind def   %rotation angle specified
/+ { rotangle rotate } bind def       %by evolution
/paperx 8.5    72 mul def
/papery 11     72 mul def
/xx paperx 0.3 mul def
/yy 400 def
/thick 25 def
/factor { pop 2 } def
1 setlinejoin
1 setlinecap
newpath
xx yy moveto                          %centers in page, roughly
thick 1 1 order { factor div } for dup scale
90 rotate                             %initial angle
order START
stroke
showpage
```

Fig. 3. Postscript header and footers into which the evolved l-system is placed.

input grammar represents another grammar) is a powerful abstraction that is made possible through the representational flexibility of grammars themselves. Many examples of the use of meta-grammars with GE exist (e.g., [6], [23], [24], [25], [26]).

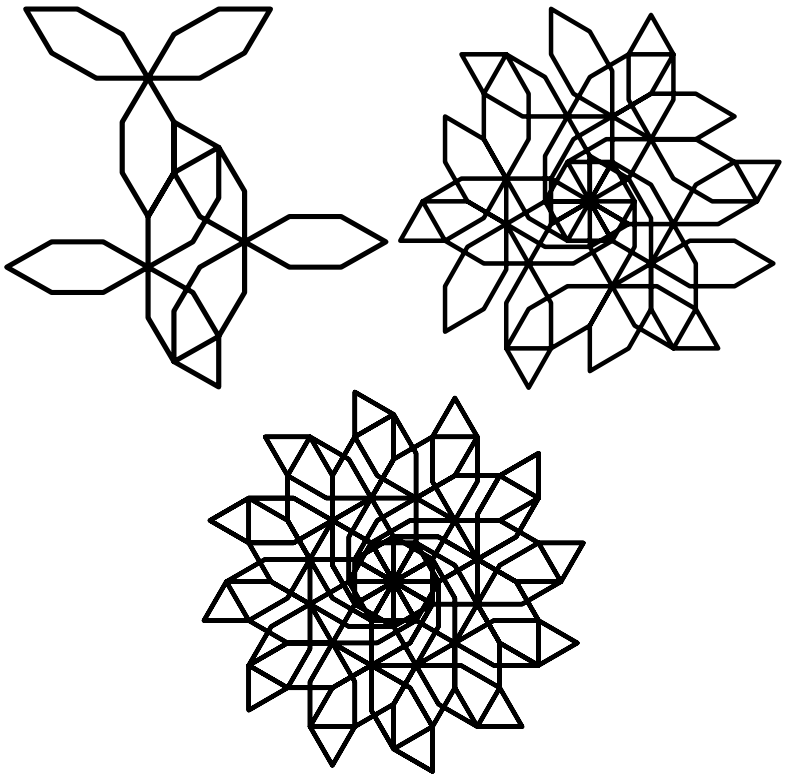

Fig. 4. F-X-X-X-F-F-FF-F, 30^o.

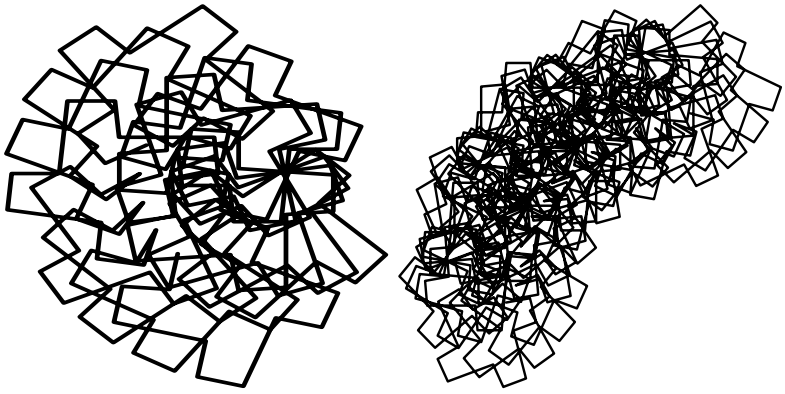

Fig. 5. F-F+F+FXF-F+F+FXF-F+F+FXF-F+F+FX, 85^o.

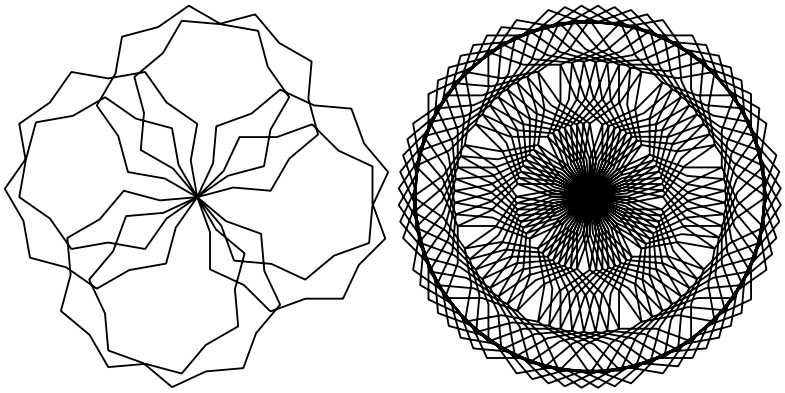

Fig. 6. -X-X-X-FFX-X-X-X-FFX, 85^o.

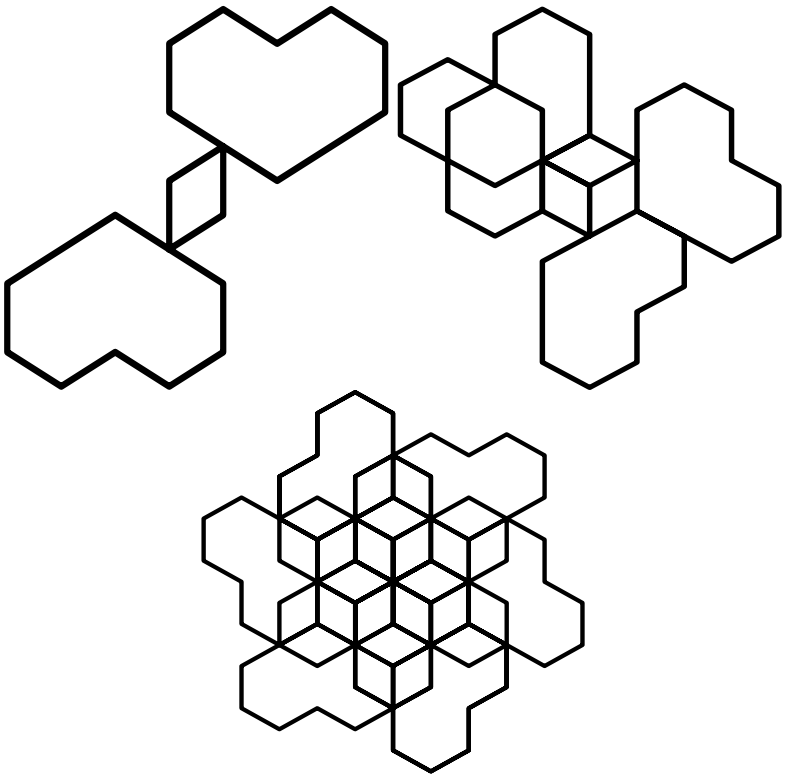

Fig. 7. F-F-F-F+X+X-F-F-F-FF-F+F+F, 60^o.

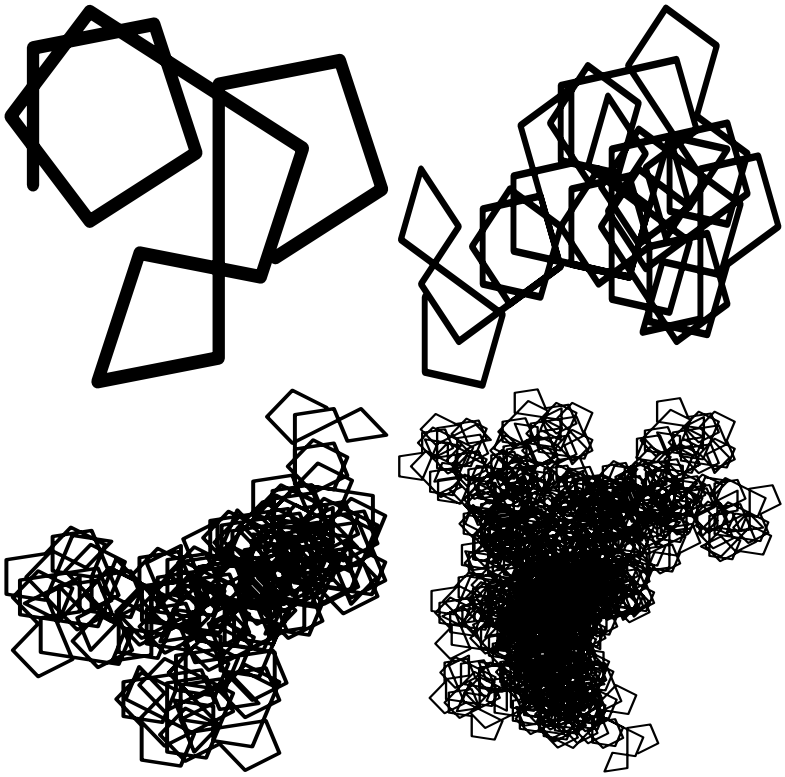

Fig. 8. F-F-F-F-X+XX-F-F-FF-F-F+FX-X-X-F+FF-F-F-F, 80^o.

IV. Results

The experiments were performed with a population size of 10 running for 20 generations, with a bit mutation probability of 0.01, and a variable-length one point crossover probability of 0.9, with roulette selection and Steady state replacement.

Initialisation was random with individuals' genotypes being generated in a size range of 15 to 25 codons. Wrapping was enabled up to a limit of 10 events. The fitness function was the author with fitness values assigned within the range 0.0 to 1.0 where 1.0 represented a perfect solution. The more aesthetically pleasing of the evolved L-systems are presented in Fig's. 4, 5, 6, 7, and 8 at the original expansion depth (order 3) and other smaller and larger depth variants to expose the underlying building block(s) that generate the larger design patterns.

Fig.9 was selected as the logo for the UCD Natural Computing Research & Applications group[1], which was founded at University College Dublin, Ireland in January 2006 by Michael O'Neill and Anthony Brabazon. The logo has a natural form that resembles an Ammonite.

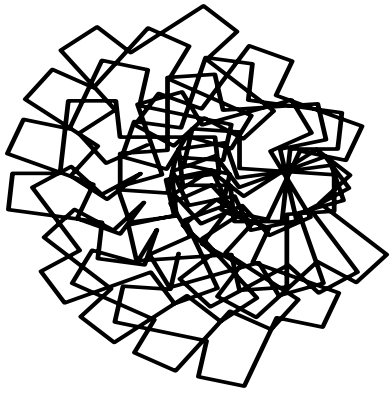

Fig. 9. F-F+F+FXF-F+F+FXF-F+F+FXF-F+F+FX, 85^o.

V. Conclusions & Future Work

We presented an application of Grammatical Evolution to the exploration of L-systems expressed in Postscript. The use of this approach resulted in the discovery of a logo for the UCD Natural Computing Research & Applications group.

Future work will include the continued development of Grammatical Evolution in the context of interactive Evolutionary Computation for the exploration of designs. In particular, Shape Grammars have proven a successful tool to capture the essence of many designs including Coffee Makers, Cars and Harley Davidson Motorbikes [29], [30], [31], [32], [33], [34], [35]. We are examining their potential for combination with evolutionary search using GE.

Research we have undertaken in addressing the fitness evaluation bottleneck for interactive evolutionary computation during sound synthesis [36] also has relevance to evolutionary design with GE. We are investigating the extension of the sweeping interface to GE to allow interpolation between individuals at different levels of granularity.

References

[1] Koza, J.R., Keane, M., Streeter, M.J., Mydlowec, W., Yu, J., Lanza, G. (2003). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

[2] Takagi, H. (2001). Interactive Evolutionary Computation: Fusion of the Capabilities of EC Optimization and Human Evaluation. Proceedings of the IEEE, Vol.89, No.9, pp.1275-1296.

[3] Bentley, P. (Ed.) (1999). Evolutionary Design by Computers. Morgan Kaufmann.

[4] Hornby, G., Pollack, J.B. (2001). The advantages of generative grammatical encodings for physical design. In *Proc. of the Congress on Evolutionary Computation*, pp.600-607. IEEE Press.

[5] Hornby, G., Pollack, J.B. (2001). Evolving L-systems to generate virtual creatures. Computers and Graphics, Vol.25 No.6., pp.1041-1048. Elsevier.

[6] Hemberg, M. (2001). GENR8 - A Design Tool for Surface Generation. MSc Thesis. MIT.

[7] Hemberg, M., O'Reilly, U-M. (2004). Extending Grammatical Evolution to Evolve Digital Surfaces with Genr8. In LNCS 3003 *Proc. of the European Conference on Genetic Programming*, pp.299-308. Springer.

[8] Hemberg, M., O'Reilly, U-M., Menges, A., Jonas, K., da Costa Goncalves, M., Fuchs, S. (2007). Genr8: Architect's experience using an emergent design tool. In *Art of Artificial Evolution*. Springer.

[9] Gero, J.S. (1994). Evolutionary Learning of Novel Grammars for Design Improvement. AIEDAM, Vol.8, No.2, pp.83-94.

[10] Langdon, W.B. (2004). Global Distributed Evolution of L-system Fractals. In LNCS 3003 Proceedings of the European Conference on Genetic Programming EuroGP 2004, pp. 349-358. Springer.

[11] Lindenmayer, A. (1968). Mathematical Models for Cellular Interaction in Development. *Journal of Theoretical Biology*, Vol. 18, pp. 280-315.

[12] Ortega, A., Dalhoum, A.A., Alfonseca, M. (2003). Grammatical evolution to design fractal curves with a given dimension. *IBM Journal of Research & Development*, Vol. 47, No. 4, July 2003.

[13] O'Neill, M., Ryan, C. (2003). *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers.

[14] O'Neill, M. (2001). *Automatic Programming in an Arbitrary Language: Evolving Programs in Grammatical Evolution*. PhD thesis, University of Limerick, 2001.

[15] O'Neill, M., Ryan, C. (2001). Grammatical Evolution, *IEEE Trans. Evolutionary Computation*. 2001.

[16] O'Neill, M., Ryan, C., Keijzer M., Cattolico M. (2003). Crossover in Grammatical Evolution. *Genetic Programming and Evolvable Machines*, Vol. 4 No. 1. Kluwer Academic Publishers, 2003.

[17] Ryan, C., Collins, J.J., O'Neill, M. (1998). Grammatical Evolution: Evolving Programs for an Arbitrary Language. *Proc. of the First European Workshop on GP*, 83-95, Springer-Verlag.

[18] Dempsey, I. (2007). Grammatical Evolution in Dynamic Environments. PhD Thesis. University College Dublin.

[19] Koza, J.R. (1992). *Genetic Programming*. MIT Press.

[20] Koza, J.R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press.

[21] Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D. (1998). *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann.

[22] Koza, J.R., Andre, D., Bennett III, F.H., Keane, M. (1999). *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufmann.

[23] O'Neill, M., Ryan, C. (2004). Grammatical Evolution by Grammatical Evolution. In LNCS 3003 *Proc. of EuroGP 2004*, pp.138-149. Springer.

[24] O'Neill, M., Brabazon, A. (2005). mGGA: The meta-Grammar Genetic Algorithm. In LNCS 3447 *Proc. of EuroGP 2005*, pp. 311-320. Springer.

[25] Hemberg, E., Gilligan, C., O'Neill, M., Brabazon, A. (2007). A Grammatical Genetic Programming Approach to Modularity in Genetic Algorithms. In LNCS 4445 *Proc. of EuroGP 2007*, pp.1-11. Springer.

[26] Hemberg, E., O'Neill, M., Brabazon, A. (2008). An investigation of meta grammars in Grammatical Evolution. *Proc. of EuroGP 2008*. Springer. To appear.

[27] Taft, E., Chernicoff, S., Rose, C. (1999). PostScript Language Reference. 3rd Edition. Addison-Wesley.

[28] Adobe Systems, Inc. (1985). Postscript Language Tutorial and Cookbook. Addison-Wesley.

[1] http://ncra.ucd.ie

[29] Stiny, G., Gips, J. (1972). Shape Grammars and the Generative Specification of Painting and Sculpture. In Proc. of IFIP Congress71, pp.1460-1465. North-Holland.
[30] Stiny, G. (1991). The Algebras of Design. Research in Engineering Design. Vol.2, No.3, pp.171-181.
[31] Brown, K. (1997). Grammatical Design. IEEE Expert, March-April, pp.27-33.
[32] Koning, H., Eizenberg, J. (1981). The language of the Praire: Frank Llyod Wright's Praire Houses. Environment and Planning B, Vol.8, pp.295-323.
[33] Stiny, G., Mitchell, W.J. (1978). The Palladian Grammar. Environment and Planning B, Vol.5, pp.5-18.
[34] Knight, T.W. (1980). The generation of Hepplewhite-style chair back designs. Environment and Planning B, Vol.7, pp.227-238.
[35] Li, A. I-Kang. (2002). Algorithmic Architecture in Twelfth-Century China: The Yingzao Fashi. In Nexus IV: Architecture and Mathematics, pp.141-150. Kim Williams Books.
[36] McDermott, J., Griffith, N., O'Neill, M. (2007). Evolutionary GUIs for Sound Synthesis. In LNCS 4448 Proc. of the Fifth European Workshop on Evolutionary Music and Art EvoMUSART 2007, pp.547-556. Springer.