GENR8 - Using Grammatical Evolution In A Surface Design Tool

Martin Hemberg
Department of Physical Resource Theory
Chalmers University Of Technology
412 96 Gothenburg
f96mahe@dd.chalmers.se

Una-May O'Reilly
AI-Lab
Massachusetts Institute of Technology
Cambridge, MA 02 136
unamay@ai.mit.edu

Abstract

In this paper we discuss how Grammatical Evolution (GE) was used in GENR8, an architect's design tool. Compared to standard GE, GENR8 provides an additional mapping that fits nicely into the relationship between GE and biology. GENR8 has a novel scheme to limit the size of the grammars that are produced when we interpret the genotype. Additionally, GENR8 has a limited form of context-sensitivity and we present a way to handle production rules that produce an optional number of nodes. GENR8 also features a function that inverts the mapping from grammar to genotype.

1 INTRODUCTION

GENR8 (Hemberg et al, 2001) is a design tool for surface generation, based on Evolutionary Algorithms (EAs) and Map L-Systems. It was developed as a plug-in for Alias|Wavefront's Maya by the Emergent Design Group (EDG) at MIT, more details can be found in (Hemberg, 2001). The tool searches the universe of surfaces with an evolutionary algorithm. It is meant to be an aid for designers trying to find new surfaces and it demonstrates the usefulness of Artificial Life (ALife) in the field of architecture.

In this paper we go into further details discussing the EA that is used in GENR8. Before we discuss those details, we give a brief description of the other components of GENR8. In the section 2 we discuss the concept of Emergent Design (ED). Next we briefly describe the growth model that GENR8 is based on. Then we describe the EA that GENR8 uses in more detail and in particular we investigate how GE has been used.

2 EMERGENT DESIGN

Artificial Life is to a large extent the study of complexity. Emergence is a central theme and it is essential to understanding ALife. We try to apply the concepts of ALife to architecture. In ALife, structures are usually built bottom-up, a notion that architects also are very fond of. Emergence allows us to create complex solutions through the interaction of primitive elements.

We combine this notion of emergence with EAs to create a system that can generate novel forms and explore a large universe of potential designs. EAs bring creativity to the design process and enables the computer to create truly novel designs.

The focus of ED (O'Reilly and Testa, 2000) is on the process that creates an artifact rather than the artifact itself. In ED, we try to formulate principles that are the foundations of these processes. The objective is to create a level of dynamics and interactions that will create a level of organization in space and time.

3 GROWTH MODEL

GENR8 creates surfaces using an organic growth model. An organic growth model tries to mimic the way plants and animals grow; the model has three important components.

Seed The seed is the starting point of the growth.

Rules In addition to the seed there are a number of rules that describe how the seed will grow.

Environment There is an environment that affects the growth. In the general case, it is impossible to determine the outcome of a growth process without knowledge of the environment. The synergy of environment and growth is an important aspect of complexity and emergence in GENR8.

3.1 L-SYSTEMS

Lindenmayer Systems (Prusinkiewicz and Hanan, 1989) or L-Systems for short, is a class of formal grammars. L-System grammars are applied in parallel to a string and a simple L-System is given in table 1. If the strings are interpreted using turtle-graphics, we can construct grammars that resemble plants and trees.

Table 1: A simple L-system. The first line is the seed for the L-System. The second and third lines are the rewrite rules for the L-System.

$$\omega{:}F_1$$
$$p_1{:}F_1 \rightarrow F_1[-F_2]F_1[+F_2][F_1]$$
$$p_2{:}F_2 \rightarrow F_2F_1$$

3.2 MAP L-SYSTEMS

Map L-systems are an extension of ordinary L-Systems that was originally used as model of cellular development in organisms. Formally, it can be seen as a method for rewriting planar graphs. A simple example of a Map L-system is shown in Figure 1.

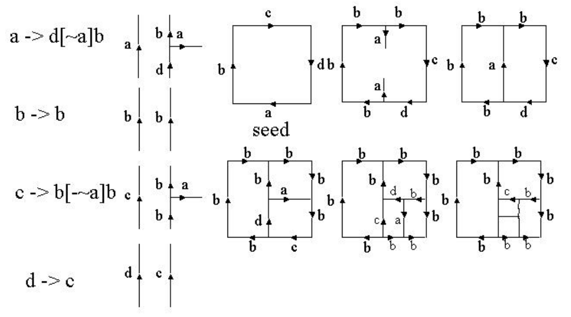

Figure 1: A simple example of a Map L-System. On the top row, the rewrite rules are shown, both symbolically and graphically. To the left on the middle row, the seed is shown and then the results of successive applications of the rewrite rules.

3.3 HEMBERG EXTENDED MAP L-SYSTEMS

In order to grow surfaces in three dimensions, we had to extend the Map L-Systems model. We call this model Hemberg Extended Map L-Systems (HEMLS). Like ordinary Map L-Systems, we have a graph and a set of production rules that are successively applied to the graph. The major additions to the ordinary Map L-systems are:

3D We have added a third dimension so that the surfaces can exist in space rather than being confined to a plane.

Growth The growth is achieved by simple scaling. In each step, the nodes of the graph are simply moved away from the center of the graph.

Probabilistic rewrite rules This is a concept that is present in ordinary L-Systems and it has simply been modified to work with HEMLS.

Context sensitive rewrite rules This is another concept present in ordinary L-Systems. By making the rewrite rules context sensitive, we can have much more complex surfaces where the growth depends on the number of steps grown.

4 EVOLUTIONARY ALGORITHM

The growth model presented in section 3 is very powerful and it can create a wide range of surfaces. Unfortunately, there is one major drawback, constructing the grammars is a very difficult task. If the tool is going to be useful to architects, we can not expect them to sit down and construct their own grammars by hand, the tool has to aid them.

Apart from helping the user constructing a grammar, the EA serves two purposes.

Exploration The designer is able to search the universe of surfaces for interesting shapes. GENR8 helps the designer by presenting various surfaces.

Discovery If the designer has some particular shape in mind, the EA can be used to find a grammar that generates the desired shape.

4.1 MAPPINGS

Compared to standard GP, GE introduces an additional mapping; from the genome to the grammar. Ryan and O'Neill (1998) argue that this makes GE more similar to 'real' biology. In the cell, DNA is transcribed to RNA that carries the encoded instructions to proteins in the ribosomes. Here proteins are assembled from amino acids. In GE, transcription maps a binary string to an integer string. The production rules use the integers to create a grammar.

In GENR8 the terminals of the Backus-Naur Form (BNF) production rules are of course the words that constitute the grammar. Thus we can argue that GENR8 goes one step further and extends the analogy; the mapping of a grammar to a surface, could be compared to proteins building more complex structures in the cell. At this point, the environment affects the process, both in the cell and in GENR8. This is illustrated in Figure 2.

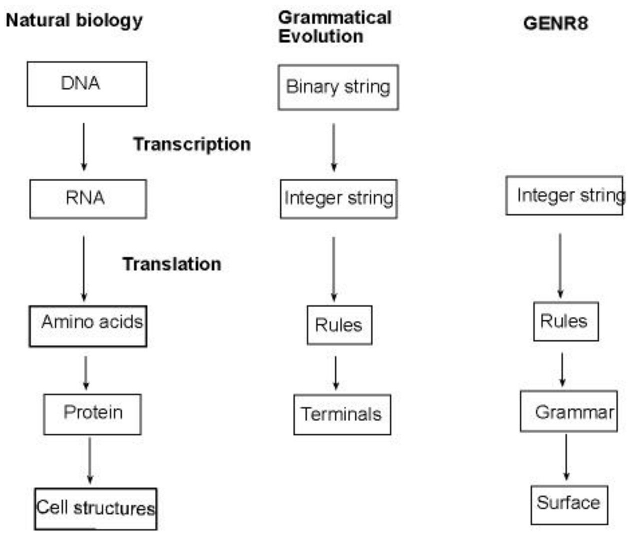

Figure 2: A comparison between biology, GE and GENR8.

4.2 EXPANSION

In standard GE, the grammar is generated by applying the production rules in the BNF to the seed; GENR8 contains five different BNFs:

Parser This is the most general BNF in the system. It is the grammars that the built-in parser can handle.

Default The default BNF used by the EA.

Symmetric This BNF generates grammar that produce symmetric surfaces.

Probabilistic This BNF produces grammars containing probabilistic rewrite rules.

Reversible The grammars generated by this BNF can be inverted so that we can obtain the genotype that was used to generate it. This is further described in 4.3.

The default BNF is given below:

```
N = { L-System, Axiom, RewriteRule,
 Predecessor, Successor, Modifier,
 AngleValue, BranchAngleValue }

T = { +, -, &, ˆ, \, /, ˜, [, ], <, >, ->,
 Edge, Angle, Sync, EdgeX, BranchAngle }

S = { <L-System> }

P = {<L-System> ::= <Axiom> <RewriteRule>
 { <RewriteRule> } Angle AngleValue [ Sync ]
 BranchAngle BranchAngleValue

<Axiom> ::= <Edge> [ ˜ ] + <Edge> [ ˜ ] +
 <Edge> { [ ˜ ] + <Edge> }

<RewriteRule> ::= <Predecessor> -> <Successor>

<Successor> ::= { <Modifier> } <Edge>

<Predecessor> ::= <Edge> { <Edge> } |
<Edge> ‘‘<’’ <Edge> |
<Edge> ‘‘>’’ <Edge> |
<Edge> ‘‘<’’ <Edge> ‘‘>’’ <Edge>

<Modifier> ::= { <Edge> } |
+ <Modifier> - |
- <Modifier> + |
& <Modifier> ˆ |
ˆ <Modifier> & |
\ <Modifier> / |
/ <Modifier> \ |
˜ <Modifier> |
<Edge> ‘‘[‘‘ ‘‘[‘‘ + <EdgeX> ‘‘]’’ -
 <EdgeX> ‘‘]’’ <Edge>
<Edge> ‘‘[‘‘ ‘‘[‘‘ + + <EdgeX> ‘‘]’’
 - - <EdgeX> ‘‘]’’ <Edge>

<AngleValue> ::= 30 | 45

<BranchAngleValue> ::= 15 | 30 | 45
 | 60 | 75 }
```

To have a complete grammar, we must also specify the type of the <Edge>. The type is represented as a number and we use a gene to determine the type.

We note that there are production rules that contain the { } symbols. They indicate that we are going to have an optional number of the (non)terminals that are between them. To determine how many of the (non)terminals we are going to have, we read genes from the genome and interpret them as random numbers. The procedure is best illustrated with a short example.

We start with a genome: `(617, 666, 8008)` and we are going to expand `<Edge>` . The brackets surrounding the non-terminal indicate that we are going to have an optional number of `<Edge>`. We use the genes to determine how many; by testing `617 Mod 2 = 1`. This is greater than the current number of expansions from this node, so we add an `Edge` node. Next we test 666 Mod 3 = 0, which is less than the number of expansions, so we stop expanding this node. We continue by determining the type of the `<Edge>` terminal by taking `8008 Mod 4 = 0`, where the modulo is the current number of edge types (in this example arbitrarily set to 3) plus one. Thus we end up with `Edge0`, when the expansion is finished

When using this scheme, there is no upper limit to the number of nodes that we can add. But for each node that we add, the probability that we will add another decreases.

In order to get a more regular structure and facilitate the connection of the branches we wish to force the system to choose the same index for the branches. The `EdgeX` symbols indicate that we should have the same type for all `EdgeX` symbols in the production rule. This can be regarded as a very limited form of context-sensitivity, the type of the second `Edge` depends on the type of the first.

4.2.1 Max depth

When expanding a seed into a grammar, there is a possibility that it will grow indefinetly. To limit the size of the grammar, we have devised a scheme that differs from the one used in standard GE. In standard GE, the number of wrap-arounds of the genome is counted. When the number of wrap-arounds exceeds a certain number, the expansion is interrupted.

We use a parameter called maximum depth that is initiated to a positive value. When a node is expanded, the maximum depth is decreased by one. If the value reaches zero, we choose production rules in a different way than we do ordinary. If possible, we choose a production rule that only contains terminals. By doing this, the expansion will automatically stop as soon as possible when the max depth has been decreased below zero. Another way to state it is that at a certain point in the expansion we start choosing production rules from a subset of the previous rules to halt the expansion.

4.3 REGN8

We have developed a feature that we call regn8. This function takes a grammar as argument and it returns a genome that could have been used to create this grammar. When using this feature, we must add some restrictions to the BNF. This feature allows the user to construct her own grammar by hand, either from scratch or by modifying one that has been provided by the system. The user-constructed grammar can be fed into the system and a corresponding genotype is constructed. The genotype can be used as a seed for the evolution and the user can easily experiment with variations on a certain grammar. This gives the user more control over the tool, the user can now modify a grammar by hand if she believes that a certain change would improve the surface; she does not have to wait for the EA to make this modification. From a user perspective, this allows for greater control and flexibility.

5 CONCLUSIONS

We have successfully applied GE to the problem of surface generation. We have extended the standard GE model by adding an additional mapping step from the grammar to the phenotype. We have also come up with some new ideas for expanding the grammars as well as a method for mapping the grammars back to genotypes.

References

Martin Hemberg, Una-May O'Reilly and Peter Nordin, *GENR8 - A Design Tool for Surface Generation*, late-breaking paper GECCO 2001.

Martin Hemberg, *GENR8 - A Design Tool for Surface Generation*, Master's Thesis, Chalmers University Of Technology, Gothenburg, 2001, report, plug-in and source code available at www.ai.mit.edu/projects/emergentDesign/genr8.

Una-May O'Reilly and Peter Testa, *Representation in Architectural Design Tools*, Invited paper at Fourth International Conference on Adaptive Computing in Design and Manufacture (ACDM 2000) ,Plymouth, 2000.

Przemyslaw Prusinkiewicz and James Hanan, *Lindenmayer systems, fractals and plants*, Springer-Verlag, 1989.

Conor Ryan and Michael O'Neill, *Grammatical Evolution: A Steady State approach.*, Second International Workshop on Frontiers in Evolutionary Algorithms, pages 419-423, 1998.