Extending Grammatical Evolution to Evolve Digital Surfaces with Genr8

Martin Hemberg[1] and Una-May O'Reilly[2]

[1] Department of Bioengineering, Bagrit Centre, Imperial College London, South Kensington Campus, London SW7 2AZ, UK
martin.hemberg@imperial.ac.uk
[2] Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
unamay@ai.mit.edu

Abstract. Genr8 is a surface design tool for architects. It uses a grammar-based generative growth model that produces surfaces with an organic quality. Grammatical Evolution is used to help the designer search the universe of possible surfaces. We describe how we have extended Grammatical Evolution, in a general manner, in order to handle the grammar used by Genr8.

1 Genr8:The Evolution of Digital Surfaces

We have built a software tool for architects named Genr8. Genr8 'grows' digital surfaces within the modeling environment of the CAD tool Maya. One of Genr8's digital surfaces, is shown on the left of Figure 1. Beside it, is a real (i.e. physical) surface that was evolved first in Genr8 then subsequently fabricated and assembled. A surface in Genr8 starts as an axiomatic closed polygon. It grows generatively via simultaneous, multidimensional rewriting. During growth, a surface's definition is also influenced by the simulated physics of a user defined 'environment' that has attractors, repellors and boundaries as its features. This integration of simulated physics and tropic influences produces surfaces with an organic appearance.

In order to automate the specification, exploration and adaptation of Genr8's digital surfaces, an extended version of Grammatical Evolution (GE) [7] has been developed. The purpose of this contribution is to describe how and why we used and extended GE. We proceed as follows: In Section 2 we explain how Genr8 creates a surface via a generative process. A surface, because of this generative process, is difficult to design by hand and the search space of possible surfaces is vast. In Section 3 we give the reasons why evolutionary computation is ideally suited to address this problem. We also explain why GE, with several extensions, conveniently resolves the issues of representation and mapping we face when using EC. Following, we show some design results and summarize.

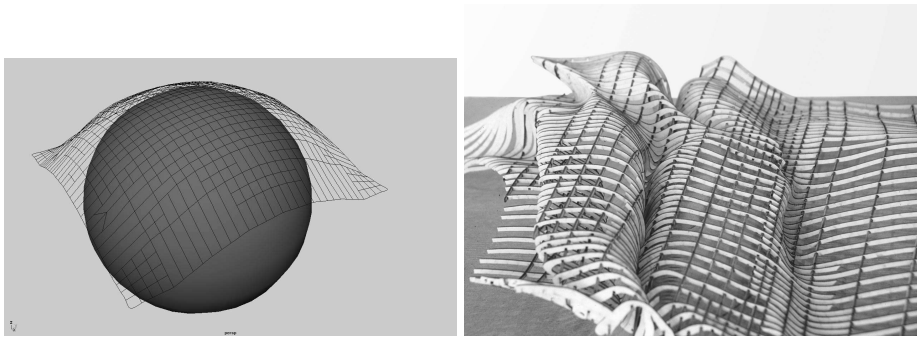

Fig. 1. Left hand side, a Genr8 surface grown in an environment where it has been pulled down by gravity on top of a spherical boundary. Right, a physical model made by Jordi Truco at the Architectural Association in London. The design process started with Genr8 and the evolved digital surface was subsequently exported so that a physical model could be manufactured.

2 Digital Surface Growth

Genr8 simulates organic growth and creates digital surfaces. Its growth process is reactive: it takes place in a simulated physical environment. Surface growth is generated by a HEMLS - a Hemberg Extended Map L-System. A HEMLS is a more complex version of the widely known Lindenmayer System (or L-system) [8]. An L-system is a *grammar*, consisting of a *seed* (or axiom) and a set of *production rules*, plus a *rewrite* process in which productions rules are applied to the seed and its successive states.

An L-system generates strings of symbols and is computer language oriented. It can be transformed into a visual representation by interpreting the seed and rewrite process graphically. The graphical interpretation is based on the metaphor of the seed being drawn in 3D space by a turtle subsequently moving through 3D space according to directions specified by the production rules and (re)drawing lines as it goes along. The turtle's directional movement is based on a set of instructions that are presented in Table 1. An example of a graphical L-system is given in Figure 2. Realistic images of plants, flowers and trees have been generated as a generated sequence line segments by specially designed L-systems [8].

A Map L-system models 2D cellular structures such as leaves via planar graphs, in contrast to the 3D linear segmented structures of an L-system. All rewrite rules are applied simultaneously to the planar graph (which has an internal representation as a graph) and then any unconnected new edges are either joined or eliminated. An example Map L-system is shown in Figure 3.

Since our goal was creative, responsive surface design, we extended a Map L-system to a HEMLS. The most important feature incorporated into a HEMLS is how the physics of our simulated environment is factored into the rewriting of an edge. The designation of the vertices of the new edge is altered in accordance with

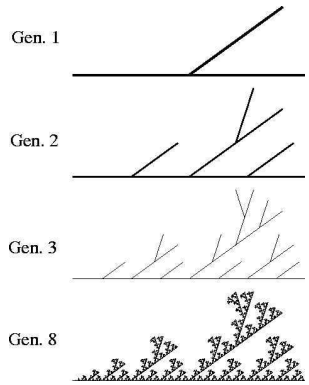

Fig. 2. An example of how a graphical L-system grows from [9]. The top image shows the seed, consisting of three segments. During each growth step, each segment is replaced by the original three segments of the seed. Growth steps 2, 3 and 8 are shown. The rewrite rule is symbolically expressed as `F -> F [ + F ] F`. The `F` to the left of the arrow is the *predecessor*, i.e., the symbol that will be replaced in the growth step. The sequence to the right is the *successor*; the symbols that will replace the the predecessor. The `F`s are segments and the `+` indicates that the turtle should turn at an angle of $\delta = 36°$ (the numerical value is given as a parameter to the system). The `[` pushes the current state (ie position) of the turtle on to a stack. When the pop symbol, `]` is encountered, a state is popped from the stack and the turtle attains that state. This push-pop mechanism enables the branching structure of L-Systems.

how the edge would be pulled by an attractor, pushed by a repellor or deflected by a boundary in the simulated physical environment. With this additional set of environmental factors, the surface growth mimics *tropism*, the response of an organism to external stimuli. Figure 4 shows how a HEMLS surface's growth steps in response to five repellors.

There are additional features in a HEMLS. These include the capacity for the grammar to be context sensitive (e.g. the left hand side of a production rule can express multiple symbols in a specific order), the option of randomly choosing among multiple production rules applying to the same symbol, and rewrite rules that change between time steps.

3 Evolutionary Computation in Genr8

Genr8's means of growing a digital surface implies using any HEMLS within a huge universe. Constructing interesting and useful grammars by hand is a formidable task and is unrealistic to expect of an architect. The EA component of Genr8 serves three purposes related to this need for automation:

- It automatically generates a HEMLS with correct syntax.
- It adaptively explores some of the universe of surfaces following preferred surfaces yet yielding creatively different ones.

Turtle Command	Meaning
$A_i, B_j, C_k, ...$	Move forward and draw a line.
+, -	Turn left/right δ.
&, ^	Pitch down/up δ.
/, \	Roll left/right δ.
$\sim$	Change direction of the segment.
[,]	Push/pop state on stack.

Table 1. Turtle commands and their meaning. A parameter δ specifies turn angle

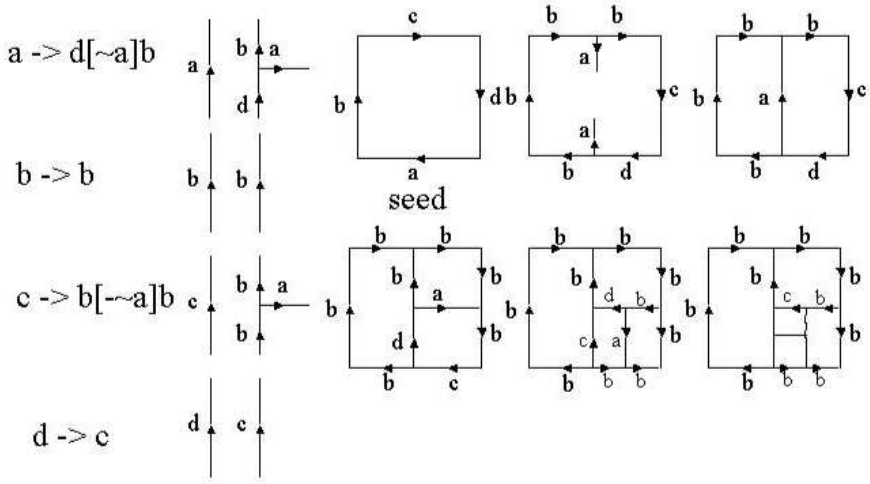

Fig. 3. An example of a Map L-System from [8]. The rewrite rules are shown to the left in symbolic and graphical form. Next, the seed (a square) is shown and the rewrite rules are subsequently applied to it.

– It can be used to explicitize a surface an architect has in mind by allowing its search process to be influenced by the architect.

These reasons are compelling to use EC but they leave open how a HEMLS should be represented within an evolutionary algorithm. Clearly the phenotype, i.e. the outcome, of the EC component is the digital surface. However, should a direct representation of a HEMLS function as the genotype, i.e., the genetic material subject to blind variation, as in [4] and [5]? This is an awkward choice because a HEMLS is a *grammar*. Blind variation on a direct grammar representation would have to be either contrived to adhere to syntax constraints or it would result in offspring genotypes that have to be syntactically repaired or culled afterwards. An alternative is to exploit a powerful approach used in various genetic programming examples such as cellular encoding [1] and embryonic circuit development [6]. In particular, we employ Grammatical Evolution [7] and provide simple effective extensions to it.

Fig. 4. A time series capturing 7 growth steps of a HEMLS surface in an environment with five repellors (here drawn as cylinders). The smallest surface is the axiom. The largest surface is the final growth step.

3.1 Genr8 extends Grammatical Evolution

GE provides a means of mapping a fixed length bit string or integer sequence (which is its genotype and thus blindly crossed over or mutated) into a program via the use of a programming language specification given in Backus-Naur Form (BNF). This implies that the genotypic search space is simpler in terms of representation. Also, it implies that GE is a general means of using genetic programming to generate an executable 'program' in an arbitrary language. In our case the 'programming language' is the language that describes any HEMLS. The 'program' is a HEMLS. In Genr8's BNF specification of the HEMLS language we express both syntactic and *semantic* constraints for a HEMLS. These are *implicitly* respected as the BNF is used to derive a HEMLS from the genotype. An example of a syntactic constraint is that a `<RewriteRule>` must have a `<Predecessor>`, followed by a arrow towards the right that is followed by a `<Successor>`. Semantic constraints are more powerful. One example, seen in Figure 5, is a heuristic for symmetry that is expressed in the `<Modifier>` nonterminal: a turn in one direction along one axis must be subsequently followed by a reciprocal turn on the same axis.

In typical GE systems, the program resulting from mapping the genotype with the BNF is executed and its outputs are compared to desired outputs. In Genr8 the HEMLS is rewritten via graphical interpretation as a planar graph (in a particular kind of execution) and a set of particular qualities of the resulting digital surface are assessed according to derive a fitness value. Thus, Genr8 has two levels of mappings. Interestingly, each mapping is essentially similar

in terms of being generative. A *generative* system has a simple starting point and the procedural means of becoming increasingly complex. The GE mapping from a gentoype to a HEMLS allows a standard genetic algorithm to search in a space of integer strings. This is simpler and more convenient with respect to withstanding the random effects of blind variation than operating directly in the grammar space. The use of GE preserves semantic and syntactic constraints in the more complex representation space of a HEMLS. The HEMLS to surface mapping supports 'growth' and produces a digital representation that is visual.

All Genr8 BNFs have `<L-System>` as their seed. The `<L-system>` symbol is rewritten as an `<Axiom>`, one or more `<RewriteRule>`'s and, then, some parameters for the turtle. An example HEMLS derived via this BNF is shown in Table 2

Edge0 + Edge1 + Edge2 + Edge3
Edge0 → Edge3 [[+ + Edge0] - - Edge0] Edge1
Edge1 → Edge1
Edge2 → Edge1 [[+ + Edge0] - - Edge0] Edge3
Edge3 → Edge2
Angle 45
Sync
BranchAngle 45

Table 2. An example HEMLS derived from the BNF of Figure 5

The first line expresses the `<Axiom>`, i.e., an initial surface of 4 edges each with a different label (e.g. zero through three). The next four lines are the `<RewriteRule>`s, one for each edge, and the three last rows contain parameters for the turtle. A more detailed description of the derivation and interpretation of the grammar can be found in [2].

The BNFs used in Genr8 use two interpretations of the genome that have not been used in previous implementations of GE. First, if there are many non-terminals in the BNF's production rules, an expanded (or derived) HEMLS is likely to be very large. For instance, in the BNF of Figure 5, six of the nine productions of the non-terminal `<Modifier>` themselves expand to `<Modifier>`. Thus, there is a $\frac{2}{3}$ probability of another expansion being required. In Genr8 such very complex rewrite rules in a HEMLS are generally undesirable because they tend to produce less interesting surfaces. The conventional way in GE to prevent this excessive expansion is to restrict the number of times the genome is allowed to wrap around during its decoding. Once the wrapping limit is reached, the individual is assigned the worst fitness fitness possible to make sure that it does not survive the selection process. In Genr8, the problem is addressed differently. A parameter called `max_depth` limits the maximum depth of the *expanded* syntax tree rather than limiting the mapping of the genotype. When the maximum depth is one short of being reached, only BNF production rules that use terminals are used to rewrite a non-terminal. This scheme still ensures

```
N = { L-System, Axiom, RewriteRule, Predecessor, Successor,
      Modifier, AngleValue, BranchAngleValue }
T = { +, -, &, ^, \, /, ~, [, ], <, >, ->, 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> ^ |
               ^ <Modifier> & |
               \ <Modifier> / |
               / <Modifier> \ |
               ~ <Modifier> |
               <Edge> ‘‘[‘‘ ‘‘[‘‘ + <EdgeX> ‘‘]’’ - <EdgeX>
               ‘‘]’’ <Edge>
               <Edge> ‘‘[‘‘ ‘‘[‘‘ + + <EdgeX> ‘‘]’’ - -
               <EdgeX> ‘‘]’’ <Edge>
<AngleValue> ::= 30 | 45
<BranchAngleValue> ::= 15 | 30 | 45 | 60 | 75 }
```

Fig. 5. A HEMLS language expressed in Backus Naur Form (BNF). Any derivation of this BNF produces a HEMLS (i.e. a grammar). One derivation is shown in Table 2.

that a HEMLS will not expand indefinitely while it also puts a less abrupt consequence to long expansion.

Second, in a BNF, the symbols {...} and [...] indicate that whatever symbols (terminals or non-terminals) appears between them is to be written zero or more times or one or more times, respectively. For example, in the BNF of Figure 5, <Successor> is rewritten as zero or more <Modifier>'s then an <Edge>. Because of the importance of genetic inheritance (and the propogation of genetic characteristics) in a GE system, it is important that this optional quantity stay consistent through all decodings of a genotype (i.e., from one generation to the next or within multiple copies of the genotype in the population). The ideal way to achieve this consistency is to use the genotype itself to determine how many times a symbol is written when it is optional. This is accomplished in Genr8 via a simple algorithm that, when a { is encountered, initializes a counter to zero. Next, the following genes on the genome are used, instead of a random number generator, to determine how many times the symbol between the {...} should be written. The algorithm loops and writes the symbol as long as the next gene has a value at least two less than the counter (which itself increments each time through the loop). This leads to a symbol being written again with decreasing probability as it is written more and more, yet there is no fixed upper limit on the quantity. This scheme is deterministic and ensures that the genome will produce the same HEMLS every time it is interpreted.

The <Edge> terminal in a HEMLS BNF is also special in the sense that it requires a label. For convenience a numerical labeling is used in Genr8. Whenever an <Edge> is derived for a HEMLS, the label must be determined. Either an existing label must be used or a new label must be generated. As in the case of optional quantities of symbols, label assignment requires consistency in terms of genotype decoding. The scheme used in Genr8 is similar to the one used for multiple nodes described above. There is a counter that keeps track of the current number of edge labels. The probability of introducing a new edge label decreases with the number of edge labels already existing.

Finally, the BNFs of Genr8 are extended in one way to ensure that branches of a HEMLS will connect. A semantic heuristic enforces the same edge being used in different places in a production rule's successor. For example:

```
<Modifier> ::= <Edge> [ [ + <EdgeX> ] - <EdgeX> ] <Edge>
```

Here, the <EdgeX> terminal is used instead of the <Edge> in order to imply (and enforce) that the edges designated by <EdgeX> need to both have the same label. As the <Modifier> non-terminal is expanded, the first <EdgeX> is assigned a label as described previously. All subsequent <EdgeX> terminals in the production will automatically get the same label as the first one. This informal 'binding' is reminiscent of free variable binding in logic programming.

4 Genr8's Digital and Physical Results

In the case of surface exploration a picture is sometimes worth a thousand words. Genr8 has now been used for two years in a design course within the Emergent

Design & Technologies graduate program at the Architectural Association (AA) in London. Figure 7 and 6 is the final example of Genr8 that space permits. Additional images are available on the world wide web at `http://www.ai.mit.edu/projects/emergentDesign/genr8`.

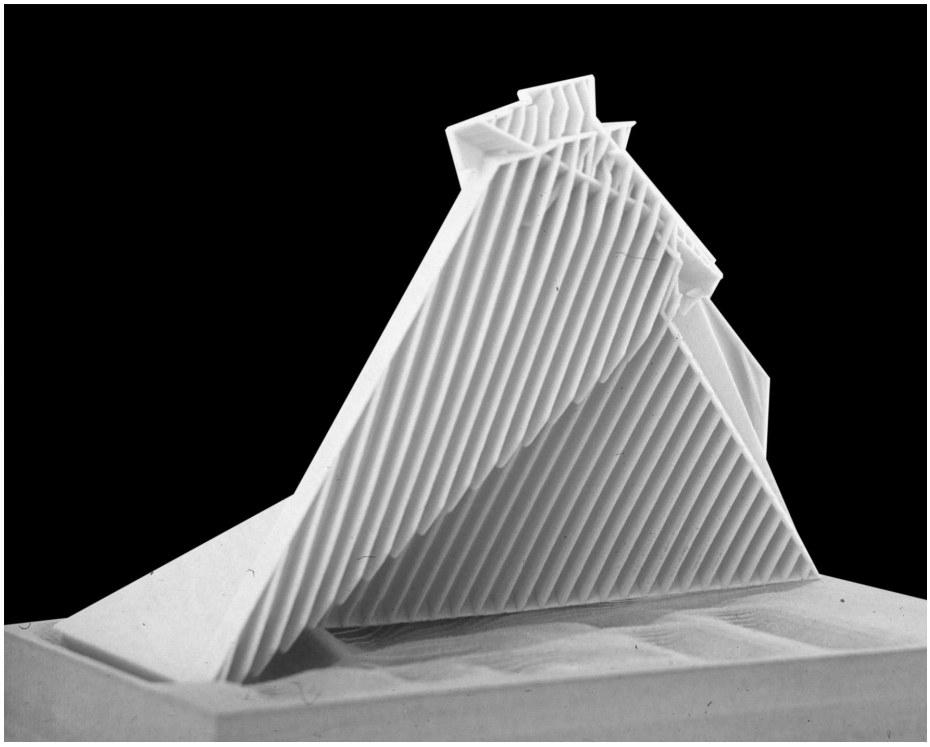

Fig. 6. A physical model, fabricated via CNC milling, based on a Genr8 surface. By Linus Saavedra, AA, London, UK, 2003.

5 Summary

We have described how evolutionary computation, and in particular, extensions of grammatical evolution, empower a digital surface exploration tool named Genr8. Because of its responsive growth and evolutionary algorithm, Genr8's results are attractive, spontaneous and organic in appearance.

Genr8 also demonstrates how a creative design tool based on evolutionary computation can be effective in three other major respects that are elaborated upon elsewhere (see [2, 3]: Genr8's Interrupt, Intervene and Resume (IIR) capability facilitates partner-based design between itself and its human user. Second, the parameterized fitness function allows the user to decide what features of the

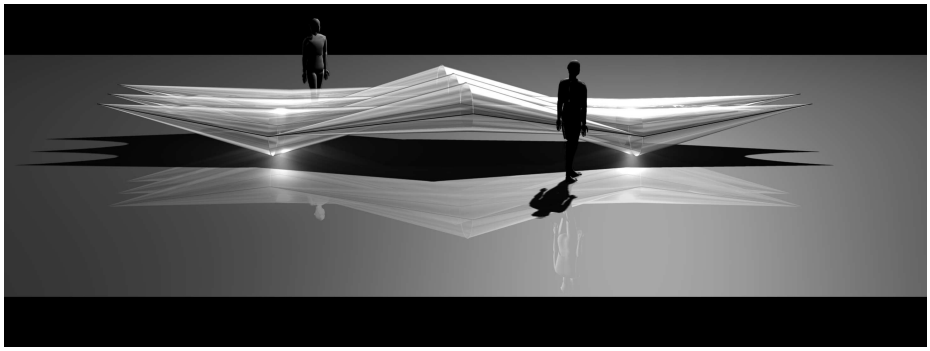

Fig. 7. A rendered image of a design for a pneumatic strawberry/champagne bar. This is Genr8's largest design that has been physically actualized. By Achim Menges, AA, London, UK, 2003.

surface should be selected for. Third, Genr8 is efficiently integrated into a larger design process so that the designer can use its results in subsequent explorations that include, for example, physical model construction.

References

1. Frédéric Gruau. Genetic micro programming of neural networks. In KE Kinnear, editor, *Advances in Genetic Programming*, pages 495–518. MIT Press, Cambridge, MA, 1994.
2. Martin Hemberg. Genr8 - a design tool for surface generation. Master's thesis, Chalmers University of Technology, 2001.
3. Martin Hemberg, Una-May O'Reilly, and Peter Nordin. Genr8 - a design tool for surface generation. In *Late breaking papers, GECCO 2001*, 2001.
4. Gregory S Hornby and Jordan B Pollack. The advantages of generative grammatical encodings for physical design. In *Congress on Evolutionary Computation*, 2001.
5. Gregory S. Hornby and Jordan B. Pollack. Body-brain co-evolution using L-systems as a generative encoding. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 868–875, San Francisco, California, USA, 7-11 2001. Morgan Kaufmann.
6. John R. Koza, Forrest H Bennett III, David Andre, and Martin A Keane. Automated design of both the topology and sizing of analog electrical circuits using genetic programming. In John S. Gero and Fay Sudweeks, editors, *Artificial Intelligence in Design '96*, pages 151–170, Dordrecht, 1996. Kluwer Academic.
7. Michael O'Neill and Conor Ryan. *Grammatical Evolution - Evolving programs in an arbitrary language*. Kluwer Academic Publishers, 2003.
8. Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The algorithmic beauty of plants*. Springer, 1996.
9. David J Wright. Dynamical systems and fractals lecture notes, 1996. http://www.math.okstate.edu/mathdept/dynamics/lecnotes/lecnotes.html.