Genetic Programming and Evolvable Machines manuscript No.
(will be inserted by the editor)

Integrating Generative Growth and Evolutionary Computation for Form Exploration

Una-May O'Reilly[1], **Martin Hemberg**[2,3]

[1] Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA
[2] Department of Bioengineering, Imperial College London, Exhibition Road, London SW7 2AZ, UK
[3] Emergent Design and Technologies, Architectural Association, 36 Bedford Square, London WC1B 3ES, UK

Received: 060331 / Revised version: 061204

Abstract We present an algorithmic growth process that is an extension of Lindenmayer's Map L-systems. This growth process relies upon a set of rewrite rules, a map axiom and a geometric interpreter which is integrated with a 3D simulated environment. The outcome of the growth process is a digital surface in 3D space which has "grown" within and in response to its environment. We have developed a complementary evolutionary algorithm that is able to take over the task of generating the rewrite rules set for a growth process. Using a quantitative multi-objective fitness function that evaluates a variety of surface properties, the integrated system (evolutionary algorithm and growth process) can explore and generate diverse and interesting surfaces with a resemblance of organic form. The algorithms have been implemented to create a design tool for architects called Genr8.

1 Introduction

In this submission we describe how we have exploited an evolutionary algorithm (EA) to explore and generate computational definitions of growth processes. Once evolved, each growth process is executed interactively with a modeled physical environment and instantiated as a three dimensional (3D) digital surface. Our EA functions as a key exploration component within an open ended design assistance tool named Genr8 [16] [15]. Genr8 is a design tool for architects for surface generation that was developed by the members of the Emergent Design Group at MIT.

Natural form has always been important to architects and designers. To many people, natural form has a strong aesthetic appeal. Moreover, most natural structures are very efficient in terms of structural capacity and economy of materials. They are robust to a wide range of failures and they can fulfill multiple functions [14] [54]. Natural structures are the result of millions of years of evolution and for designers there is a compelling argument to take inspiration from Nature's strategies.

The vast majority of organisms develop by growing from a single cell. Even though this phenomenon has been studied extensively by biologists, we are still far from a complete understanding of this complex process [11]. The process takes place over multiple time scales making quantitative descriptions notoriously difficult. Many of the mathematical descriptions can be traced to the seminal work by D'Arcy Thompson [48]. In his inspirational book he uses physical principles to shed light on biological form. Another influential paradigm is that of Cellular Automata (CA) which was proposed by von Neumann as a way of describing self-replicating systems [51]. A third approach is Lindenmayer systems (L-systems) which were first introduced by biologist Aristid Lindenmayer as a phenomenological description of the growth of multi-cellular organisms. L-systems have since been successfully used to describe plant growth [37]. Unlike the work of Thompson, CAs and L-systems are algorithmic descriptions of a growth process.

Hornby and Pollack [18] have demonstrated that evolutionary algorithm results are more scalable when a generative representation of the design is employed rather than a direct representation. This suggests that one should evolve rules for creating an object rather than creating the object itself directly. This is in accordance with biology, where the genome should be thought of as instructions for how to build and maintain an organism rather than the exact blueprint of an organism. For this reason, L-systems are of obvious interest since they express growth using a set of rewrite rules which can be represented by a context-free grammar.

We proceed in the following manner: In Section 2 we discuss algorithms that have been used to generate computational versions of growth and EAs in the context of architecture. In Section 3, we provide an overview of the Genr8 system architecture. In Section 4 we describe the rewrite systems and geometrical interpreter we have designed to "grow" surfaces in 3D. In Section 5 we describe how we extend Grammatical Evolution [35] so that we can exploit an EA to explore and discover rewrite systems. In Section 6, we show Genr8's surfaces that are the products of our EA and generative growth system integration.

2 Related work

The use of growth-like algorithms has been very successful in a wide range of areas in science and engineering, including but not limited to, computer graphics [43] [37] [20], neural networks [13] and analog circuit design [25]. In this brief review, we will focus on applications of growth algorithms and EAs in architecture. The use of EAs to produce artistic works has been reviewed by Johnson and Romero [22]. A recent review of aesthetic evolution of L-systems can be found in [31]. Applications of EAs to art and design are also presented in three edited collections: [27] [6] [5].

The idea of using generative algorithms and procedures in architecture is not new. One of the early pioneers was John Frazer who began his work at the Architectural Association in the 1960's [12]. The title of his book "An Evolutionary Architecture" conveys his interest. It investigates "fundamental form-generating processes in architecture, paralleling a wider scientific search for a theory of morphogenesis in the natural world". His early projects include the Reptile structural system which incorporates notions of growth and evolution, albeit without an explicit specification from a formal language. The basic strategy was to use growth involving a small set of tiles repeatedly (rep-tile) starting from a minimal seed. His group has also evolved surfaces, although it used a mathematical description rather than a growth language. The fact that his EA-based form generating experiments are implemented physically allows them to be "understood in architectural terms as an expression of logic in space". Frazer's digital simulation efforts were impeded by the computing resources available at the time. Instead, many of his projects were implemented as physical devices using custom-built hardware, sensors and actuators. The Generator project uses embedded electronics in each component to create a building with distributed intelligence. The result is a reconfigurable space which can respond to varying needs. The most fascinating of Frazer's many projects is the Universal Constructor, a working model of a self-organizing interactive environment. The project is a physical 3D CA, realized using custom-built cubes which can respond to the environment and interact with the user. In another example of environmental influence, his group developed tools to help architects understand the impact of the geometry of the sun. The tools went beyond the standard, poorly understood stereographic projections. Frazer has also embraced the idea of creative design tools [52] [7] and in particular he has advocated the use of EC for exploration rather than optimization [21].

CAs have been used by other architecture researchers to generate form and structure. Kicinger *et al.* [23] used a CA representation of topologies for structural steel systems in tall buildings to evolve efficient structures. CAs have also been used to generate form in more architectural applications. One example of how to interpret the result of a CA is to use Conway's Game of Life and "stack" the planes to obtain a 3D form [26] [9].

L-systems have also been used in previous architectural applications, in particular by the group lead by Paul Coates. They have combined L-systems and an EA to create form on an iso-spatial grid. Using an environment to simulate, for example, sun and wind, they evolve structures that are optimized for certain performance criteria such as enclosing space. Their work also explores how the biological concepts of symbiosis and co-evolution can be incorporated in a form-generating algorithm. They point out that evolutionary optimization would probably have had a strong appeal to modernist architects, such as Le Corbusier and Van der Rohe, since the aesthetics of the outcome are purely based on function [8]. The EDG has also explored the potential of L-systems in the design tool MoSS [47]. A more design-oriented investigation of L-systems was pursued by Hornby and Jackson [18]. They developed a generative design system called GENRE which has been applied to table- and robot-design. Moreover, they conduct quantative experiments to demonstrate the advantage of generative over non-generative encodings. The generative approach is more compact and it makes it easier to evolve re-usable design modules.

A dominant way of describing architectural form is shape grammars [44] [34]. They have been combined with EAs [3]. EAs are a popular method to generate form and many of the previously mentioned authors have incorporated such a component in their work. Further applications include the evolution of floor plans [40] [32] and 3D design of buildings inside a CAD tool [50].

3 Overview of Genr8

One goal of the Genr8 project is to demonstrate that a combination of a growth algorithm and EA is useful for form exploration within the architectural design process. As a proof of concept, the algorithms have been implemented in C++ and thereby we have developed a software tool which can be (and is) used in educational practice. This section gives an overview of the software system, which has two main components: the HEMLS growth engine and the EA (see Figure 1). The growth engine uses the HEMLS interpreter to parse a rewrite system. It geometrically interprets the axiom and set of rewrite rules of the parsed system. A rewrite rule set (or system) is a context-free grammar. We name the combination of HEMLS interpreter with Genr8's rewrite rule set syntax and semantics a HEMLS: Hemberg Extended Map L-System and dub the results HEMLS surfaces. The HEMLS growth engine's growth process is strongly influenced through its interaction with a computationally simulated physical environment. This environment is the architect's means of abstractly imposing influence on the growth and of representing design space elements which should interact with the growth process. It is important to emphasize that the tool can be (and has been!) used with the growth algorithm alone.

The EA is an optional, added-value feature which leverages Genr8's expressive power by automatically generating and evaluating a large number of rewrite rule sets. Of note is the fact that Genr8 evolves *instructions* for growing surfaces rather than the surfaces themselves. The EA's selection and variation components provide the principles of design evolution. It gives the architect an additional perspective: that of parallel, ancestral traversal through a space of multiple designs. The procedural loop of the

EA is standard. Iteratively a new population is generated through selection and mutation of parents of the current one, then each member is evaluated for fitness. Genr8 employs a linear genome of integers as per Grammatical Evolution [35]. The genome is mapped to a set of rewrite rules (i.e. context free grammar) with the auxiliary aid of a Backus-Naur Form (BNF) representation that defines the syntax of the context-free grammar. Then the surface is grown by the growth engine in the simulated environment using the interpreted set of rewrite rules. It is the resulting surface that is evaluated for fitness.

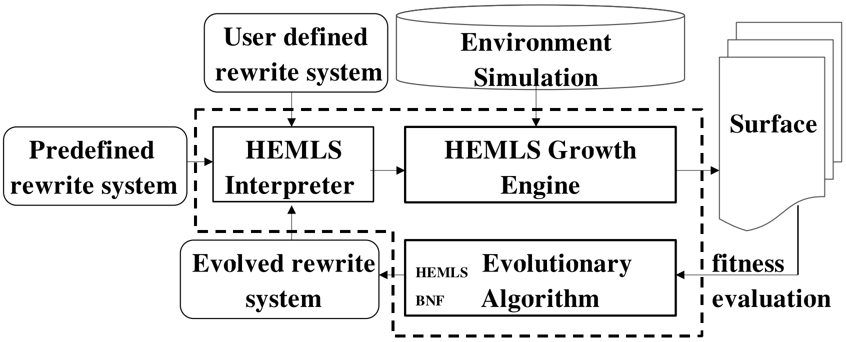

Fig. 1 A system level overview of the Genr8 system which is encircled by the dashed line. At the heart of the system is the HEMLS growth engine which is an algorithm for growing HEMLS surfaces. The growth process is influenced by the environment. The system contains a HEMLS interpreter that parses a rewrite system. When users provide their own rewrite system or use a predefined one, the evolutionary algorithm is inactive. Otherwise, GENR8 uses the EA's evolved rewrite systems. The genome of each individual in the EA is mapped to a HEMLS rewrite system using Grammatical Evolution and a Backus Naur Form definition. The surface corresponding to the genome is subsequently grown and evaluated for fitness.

We chose to implement Genr8 as a plug-in for the 3D modelling software package Maya [30]. It has a well-documented Application Programmer's Interface which is easy for software developers to work with. The integration into an existing platform also makes Genr8 more accessible for its end-users. Genr8's data object level integration within Maya implies that a HEMLS surface is also a regular addressable Maya surface. This advantageously allows a user to manipulate it in the usual Maya ways. A compiled version of Genr8 as well as the source code is available on the web[1]. The implementation is not Maya-specific and in principle it could be implemented as a plug-in for another 3D modeller or as a stand alone software.

---

[1] http://projects.csail.mit.edu/emergentDesign/genr8/

4 Extending L-systems to a model capable of generating surfaces

Our aim is to create surfaces with qualities that are reminiscent of those in natural forms. To achieve this goal we employ an algorithm which is an extension of L-systems. Even though L-systems have been successfully used to model trees and flowers (see Figure 2) [37], it should be pointed out that they are purely phenomenological and not derived from the physical properties of a growing plant. This is of little concern to us since we are interested in creating a tool which is useful to designers. Thus, the growth model presented in this paper should be appreciated from a metaphorical point of view and not as a faithful description of biological growth.

Fig. 2 Left, a rendered image of *Casilleja coccinea* (Indian paintbrush) created using L-systems [38]. Centre, a photograph of the the gametophyte *Microsorium linguaeforme*. Right, a map L-system used to model the gametophyte in the middle [37].

We begin by presenting ordinary L-systems and show how they can be extended to create surfaces in 3D space. Special focus is placed on the context-free grammar used to describe the L-systems since it plays a central role in Genr8. First we introduce notation from formal language theory [41, 46, 37].

Definition 1 (Grammar) *A* ***context free grammar*** *G is a finite specification of the (possibly infinite) sets of sentences of a language. It can formally be expressed as a tuple $G = (N, T, S, P)$ where:*

- *N is a finite set of* ***non-terminals****.*
- *T is a finite set of* ***terminals****.*
- *S is a* ***start symbol*** *such that $S \in N$.*
- *P is a set of* ***productions*** *(or* ***production rules*** *of the form $B \rightarrow \alpha$, where $B \in N$ and α is a string of symbols from $N \cup T$.*

Context free grammars are of interest since they are powerful enough to describe most programming languages. To express context free grammars, a syntactic metalanguage known as Backus Naur Form (BNF) is often used. BNF is useful for formal descriptions of a language since it provides a succinct representation using a limited number of rules. The BNF notation has a number of advantages that makes it useful as a syntactic metalanguage; it is concise, precise, natural, general, simple, self-describing and linear. The last point can also be considered the biggest limitation of BNFs. They are often inadequate for defining more complex grammars. Fortunately, they are relatively easy and natural to extend if one needs to handle more complex situations [19].

A BNF specification uses a number of **derivation rules** written as `<Non Terminal> ::= <String of Non-Terminals and Terminals>`. This should be interpreted in such a way that the left hand side can be replaced with whatever is on the right hand side. In 1977, Wirth [53] introduced a number of metasymbols to further facilitate the use of BNFs. The symbols that are relevant to this paper are presented in Table 1.

Table 1 Meta symbols used to describe a BNF [46,53].

Symbol	**Meaning**
|	A vertical bar is used to denote alternatives.
[ ]	Square brackets denote the optional appearance of a symbol or a group of symbols.
{ }	Curly braces indicate zero or more repetitions of a symbol or a group of symbols.

4.1 Lindenmayer systems

In 1968, biologist Aristid Lindenmayer introduced L-systems as a way to describe the growth of multi-cellular organisms. L-systems are based on rewrite systems, a concept invented by Thue [2]. A rewrite system consists of a seed and a number of rewrite rules that are repeatedly applied to the string. Rewrite systems are simpler than grammars in Definition 1 in that they do not have any distinction between terminals and non-terminals. Theoretical computer scientists soon took an interest in L-systems since they are similar to context free grammars. In this paper, we use a definition of L-systems from [37].

Definition 2 (L-system) *A **0L-system** (also called a context-free or parametric L-system) is an ordered triplet $G = (V, \omega, R)$ where*

- *V is an alphabet.*
- *$\omega \in V^+$ is a **seed** or axiom.*
- *$R \subset V \times V^*$ is a finite set of **rewrite rules** (these are often called productions, but we shall avoid that term here since it clashes with Definition 1). The notation $a \rightarrow \alpha$ is used for rewrite rules where the letter*

$a \in V$ is called the ***predecessor*** *and the string α is called the* ***successor****. The string α is called the successor and consists of one or more symbols from Σ. It is assumed that there exists at least one rewrite rule for each letter $a \in V$ and if none is specified, the identity rule $a \rightarrow a$ is implicitly assumed.*

A 0L-system is ***deterministic*** *iff there is exactly one successor for each predecessor.*

To produce images with the aid of L-systems the resulting string is graphically interpreted. The most widely used graphical interpretation based on turtle graphics [1] was introduced by Prusinkiewicz [37]. Briefly, the idea is to interpret the string of symbols generated by the L-system as instructions for an imaginary turtle moving a stylus in 3D space. The most commonly used turtle instructions can be found in Table 2. Two instructions that are of particular interest are the push and pop operators, "[" and "]". When the turtle encounters a push-operator, its current state is saved on a stack and when it encounters a pop-instruction, it aquires a new state from the top of the stack. This feature creates branching structures and is key to modelling plants and trees.

Table 2 Turtle commands and their meaning. The parameter δ specifies the turn angle [1].

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.
[,]	Push/pop state on stack.
$\sim$	Change direction of edge.

4.2 Map L-systems

To study cellular development of organisms that do not have a tree-like topology, Map L-systems were introduced by Lindenmayer and Rozenberg [29]. Map L-systems are L-system type grammars that are applied to maps. The parallel map generating systems introduced by Lindenmayer and Rozenberg are formally referred to as binary propagating map 0L-systems, or BPM0L-systems for short. To convey how Map L-systems differ from ordinary L-systems, we will give an informal definition from [28] (a formal definition of maps can be found in [49]).

Definition 3 (Map) – *A map is a finite set of* ***regions****. Each region is surrounded by a boundary consisting of a finite circular sequence of* ***edges*** *that meet at* ***vertices****.*

- *Each edge has one or two vertices associated with it. The one-vertex case occurs when an edge forms a loop. The edges cannot cross without forming a vertex and there are no vertices without an associated edge.*
- *Every edge is part of a boundary of a region.*
- *The set of edges is connected.*

An L-system is effectively a string rewriting system where the geometric interpretation is independent of the grammar. Map L-systems on the other hand can not be separated from the map topology and they can not be considered as one dimensional strings alone. Again, we take our definition from [28]

Definition 4 (Map L-system) *A **map L-system** consists of a finite set of edge labels Σ, a starting map ω with labels from Σ and a finite set of rewrite rules (as in Definition 2). Pairs of matching brackets "[" and "]" delimit **branch points** which specify possible attachement sites for region dividing edges.*

In each derivation step of a Map L-system, we must also add a procedure for matching branch points to either form new edges or have them removed.

4.3 Hemberg Extended Map L-systems

To obtain a description of surface growth which is more suited to our needs, we have extended the Map L-systems model. HEMLS allow a wide variety of surfaces to be grown in 3D (unlike Map L-systems that are confined to 2D). From a formal perspective, HEMLS can be described as map rewriting systems. Note that the original specification (Definition 4) only considers planar maps and that the description only specifies the map topology: there is no need to specify spatial coordinates or quantitative relationships of the node and edge positions. Unlike L-systems, HEMLS can not be considered simply as rewrite systems, the geometric interpretation is an integral part of the definition of HEMLS.

Definition 5 (Hemberg Extended Map L-systems) *A context-free HEMLS consists of*

- *A finite set of edge labels and turtle commands Σ.*
- *A **seed** or **axiom** $\omega \in \Sigma$.*
- *A set of edge **rewrite rules**, R. Each rewrite rule is of the form $a \rightarrow \alpha$ as in Definition 2.*
- *Two numerical values in the interval [0, 90] that specify a **turn angle** and a **branch angle**. There is also a boolean variable, **sync**, that determines the method for joining the branches (discussed below).*

In the following, we shall use the term HEMLS rewrite system or simply rewrite system to denote a sentence specifying a seed, a set of rewrite rules and parameters (i.e. what is included in Definition 5). To grow a HEMLS surface, the seed is first placed in the starting position. Next, a number of **derivation steps** take place. Each derivation step consists of:

1. **Increase size**. A displacement vector is calculated for each vertex. The vector originates from the center of the surface. Each vertex is moved along the line defined by the displacement vector **r**. The distance moved is determined by a **scale factor** s. We allow for $s < 1$ which means that the surface can shrink as well. Vertices are also affected by the environment which may result in further modifications or truncations of the growth displacement vector (see Section 4.7).
2. **Apply rewrite rules**. Each rewrite rule in R is applied to edges with corresponding labels. The edge is replaced by the successor starting from the start vertex. The predecessor is divided into a number of new edges as indicated by the rewrite rule. This step is similar to the derivation step in Map L-systems. Edges that are drawn between one or more push-pop pairs will not have an end vertex. These edges are called **branches** and in order to restore the map topology, they will be removed or pairwise joined into complete edges.
3. **Join branches**. There are two modes for joining the branches, synchronous or asynchronous. The latter is default and the former is used if the keyword `sync` is included in the rewrite system. In synchronous mode, all the rewrite rules are applied before the branches are joined. If the keyword is not used, branches are joined after each rewrite rule has been applied. Branches that could not be joined are kept through the remainder of the derivation step which means that they will have several chances of getting joined. When joining branches all pairs of branches are checked to see if they fulfill the criteria for forming a new edge together (as illustrated in Figure 3). To join two branches, the following criteria must be fulfilled:
   - The branches must appear in the same region.
   - They must have the same type.
   - The scalar product of their orientation must be less than a given tolerance. The tolerance is given as an angle by the `BranchAngle` keyword.
   - The branch points are within a specified maximum distance (which can be set to infinity).

   When all rewrite rules have been applied and there have been one or more attempts to connect the branches, any remaining unconnected branches are removed.

4.4 Example of a HEMLS rewrite system

There are two rewrite systems that we considered especially interesting and useful. Therefor they have been included in Genr8 as predefined rewrite systems (see Figure 1). The rewrite systems start from a square or an equilateral triangle and proceed by subdividing the original shape into four new squares or triangles. The rewrite system for the square is shown in Table 3.

Table 3 The rewrite system for the squares shown in Figure 3.

ω	:	A + B + A + B
A	→	A [[+ B] - B] A
B	→	B [[+ A] - A] B
Angle	:	90
BranchAngle	:	90

Here the first line is the seed, derived from the `<Axiom>` non terminal (see Figure 4). Lines 2 and 3 are rewrite rules and the fourth line specifies the turn angle for the turtle. Figure 3 shows the geometric interpretation and application of these rules.

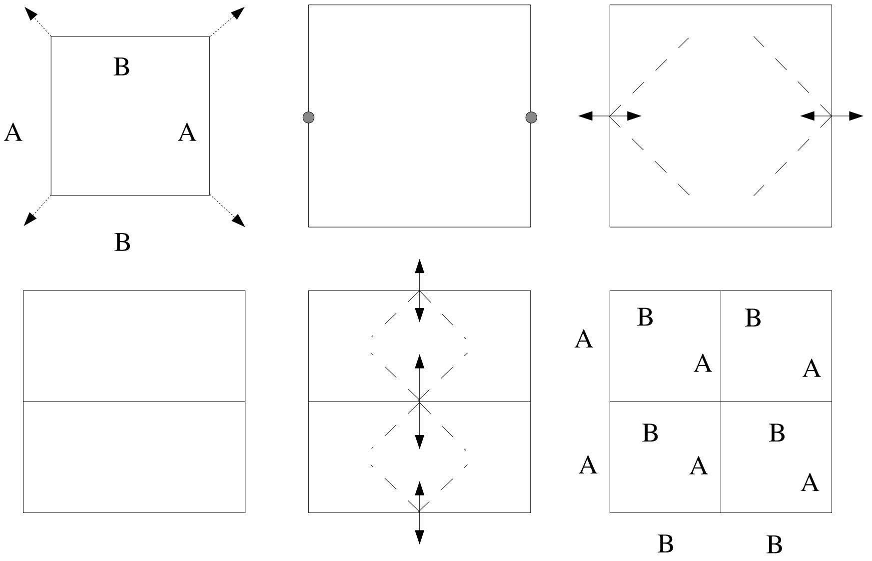

Fig. 3 One derivation step of the rewrite system in Table 3. There are two types of edges: the horizontal ones are of type B and the vertical are of type A. The square to the upper left is the seed and the arrows indicate how the vertices should be moved when increasing the size. In the next phase, the A edges have been split and the new vertices are indicated by circles. Next, the branches are drawn and connected. The dashed lines denote the cones determined by the `BranchAngle` parameter which are scanned for oppposite branch points. The same procedure is applied to the B edges in the middle panel on the bottom. The figure on the bottom right shows the surface after one iteration of the rewrite rules.

4.5 HEMLS variants

There are a number of variations and extensions of L-systems in the literature [37]. These can easily be applied to HEMLS as well to provide a more powerful growth model. A natural extension of L-systems is to make them context-sensitive, that is rewrite rules require a specific context in order to be activated. To represent this, we use the notation $A < B > C- > \alpha$, where A and C are edge labels that need to be connected to the start and end node respectively of the edge that we are trying to rewrite.

Another interesting variant is to introduce stochasticity in such a way that there are more than one possible rewrite rule that can be applied to each edge. A probabilistic HEMLS is specified by a tuple $G = (\Sigma, \omega, R, \pi)$ where π is a probability distiribution mapping the set of rewrite rules to the set of production probabilities. Every time a rewrite rule is applied to an edge, one of the available rewrite rules is randomly chosen based on the probability distrubtion π.

Timed L-systems allow us to change the rewrite rules that are applied to edges over time. This can for example be used to model plants that first grow a stem and then a flower. This feature is achieved by adding an **age** parameter to every edge (represented as a subscript). Again, each rewrite-rule can have multiple successors and the age of the edge determines which one is chosen.

4.6 The HEMLS BNF in Genr8

Genr8 includes a parser that can be used to grow surfaces specified by a HEMLS supplied by the user within a text file (the user-specified rewrite systems in Figure 1). In order to make the file easy to parse the general description presented above had to be restricted. The grammar is represented in Figure 4 using the BNF notation.

This description merits some comments since there are a few features that could not be captured by the BNF. The first three lines list all the non-terminals (`N`) used in the HEMLS grammar. Next is the set of terminals (`T`) and the start symbol (`S`). Lines 8-37 show the derivation rules (`P`) which describe how to construct a rewrite system. The expansion of the start symbol, `<L-system>`, gives the overall structure of a rewrite system with an `<Axiom>` that describes the seed, one or more `<RewriteRule>` and finally the parameters (the semicolons indicate end of line). Each `<RewriteRule>` consists of a `<Predecessor>` and one or more `<Successor>`. Lines `20-22` show how context sensitivity can be introduced. The successor is a string of turtle commands that are used to replace the predecessor. The turtle commands are generated using the `<Modifier>` non-terminal. There are two constraints on the grammar imposed by the geometric interpretation. First, there must be an equal number of push and pop symbols. This constraint is handled by the production rule on line `31`. Second, when rewriting an edge,

```
 (1) N = { <L-System>, <Axiom>, <RewriteRule>, <Predecessor>,
 (2)       <Successor>, <Modifier>, <Condition>, <Segment>,
 (3)       <Constant>, <Weight> }
 (4) T = { +, -, &, ˆ, \, /, '[', ']', '<', '>', ->, Edge,
 (5)       Angle, Sync, Edge_i, Edge_i+1, Edge_i-1, If, i, '='
 (6)       BranchAngle, '0', '1', '2', ... }
 (7) S = { <L-System> }
 (8) P = {
 (9) <L-System> ::= <Axiom> ';' <RewriteRule> ';'
(10)               { <RewriteRule> ';' } Angle <Constant>
(11)               ';' [ Sync ';' ] BranchAngle <Constant>
(12) <Axiom> ::= <Segment> [ ˜ ] + <Segment> [ ˜ ] + <Segment>
(13)            <Segment> { [ ˜ ] + <Segment> }
(14) <RewriteRule> ::= <Predecessor> -> <Successor> [ <Condition> ] |
(15)                   <Predecessor> -> <Successor> [ <Weight>
(16)                   <Constant> ] { -> <Successor> [ <Weight>
(17)                   <Constant> ] }
(18) <Successor> ::= { <Modifier> } <Segment>
(19) <Predecessor> ::= <Segment> { <Segment> } |
(20)                   <Segment> '<' <Segment> |
(21)                   <Segment> '>' <Segment> |
(22)                   <Segment> '<' <Segment> '>' <Segment>
(23) <Modifier> ::= { <Segment> } |
(24)               + <Modifier> - |
(25)               - <Modifier> + |
(26)               & <Modifier> ˆ |
(27)               ˆ <Modifier> & |
(28)               \ <Modifier> / |
(29)               / <Modifier> \ |
(30)               ˜ <Modifier> |
(31)               <Modifier> [ <Successor> ] <Modifier>
(32) <Segment> ::= Edge | EdgeX | EdgeY | EdgeZ | Edge_i |
(33)              Edge_i+1 | Edge_i-1
(34) <Condition> ::= If i '<' <Constant> |
(35)                 If i '<' <Constant> |
(36)                 If i '=' <Constant>
(37) <Constant> ::= 0 | 1 | 2 | ...  }
```

Fig. 4 The BNF representation of the language for describing HEMLS that can be understood by Genr8's parser.

the turtle must return to the position where the old edge ended. Thus, the number of left turns must equal the number of right turns, etc. Lines `24-29` make sure that this balance is preserved.

The `<Segment>` non-terminal is used to generate the `Edge` terminals. A couple of things need to be said about this terminal since it is not a terminal in the strict sense. The conventional L-system notation for lines is capital letters. To make it easier to parse, Genr8 uses a nomenclature where edges are identified by the keyword `Edge` followed by a non negative integer that defines the **type** (or **ID**), eg `Edge0`. All terminals starting with `Edge` require a parameter specifying the ID. There are other versions of the `Edge` terminal and they all require a type. The variations on the `Edge` terminal, `Edge_i`, `Edge_i+1` and `Edge_i-1` are used for time-varying HEMLS where the subscript represents the age of the edge. The counter is initialized to 0 when the terminal is created and can be incremented by the rewrite rules.

4.7 Environment

In nature, the environment plays an important role in the growth of an organism. The obvious way to alter the resulting surface in Genr8 is to modify the rewrite system. In practice it is often easier and more intuitive to use the alternative approach of changing the environment. One example of environmental influence on growth is *tropism*, which can be defined as the response of an organism to external stimuli. A simple example of tropism is a tree that turns its branches towards the sun as it grows. We wish to mimic this type of reactive influence of the environment in the growth of HEMLS surfaces. Again, we proceed by adopting existing ideas from the L-systems literature.

In Genr8 there are two different environmental features, **forces** and **boundaries**. These are easy and intuitive to use and by combining them one may produce non-linear and unexpected outcomes, that are often desired by architects. There are three different kinds of forces; **attractors**, **repellors** and **gravity**. The metaphor underlying attractors and repellors is a magnet which forces the surface to grow towards or away from it. When moving a vertex during the derivation step, the new position is modified by the forces in the environment. For each vertex a resultant force vector is calculated by summing the forces from all environmental features. The default in Genr8 is to have the magnitude of the force depend on the inverse square of the distance between the vertex and the attractor/repellor. The resulting vector $\mathbf{f}$ is added to the displacement vector $\mathbf{r}$ to find the new position of the vertex. Two adjacent vertices can potentially move different distances and in different directions and thereby distort the surface.

Boundaries are used to constrain the growth by preventing vertices to move through them. Boundaries can be very powerful as they can be used both as enclosures or as obstacles (as in the left panel of Figure 5). The environment can have a profound impact on the growth due to the non-linear interactions. It is almost impossible to predict the exact outcome when there are more than two elements in the environment. This is clearly seen in Figure 5 which shows two examples of the square rewrite system from Table 3.

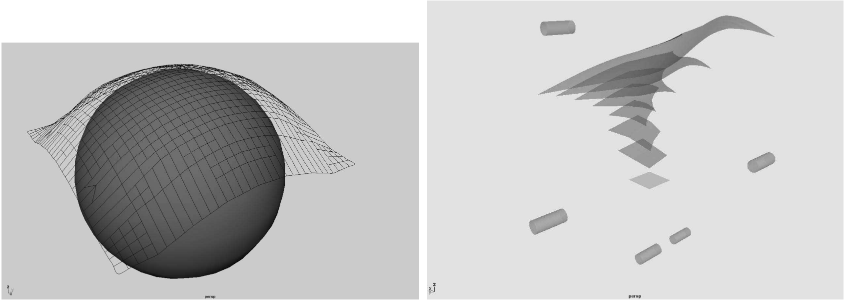

Fig. 5 The figure shows two examples of the square rewrite-system from the example in Table 3 grown in two different environments. Left, the seed was placed above the sphere and pulled down by gravity as it grew. The surface was prevented from growing through the sphere and instead it drapes the sphere. Right, the surface is pushed upwards by the two repellors beneath it. During the development it was further deformed by the repellors (the five repellors are located to a single point, but drawn as cylinders). The figure shows all derivation steps overlaid and the shape of the seventh surface is far from the original flat square.

5 Evolutionary computation

Creating a rewrite system that grows interesting surfaces by hand is a very hard task. This stems mainly from the difficulties of imagining what a given rewrite system will look like after repeated iterations. The influence of the environment only serves to exacerbate this problem. There is also the additional complication of making sure that the rewrite system is syntactically correct. It is with these concerns in mind that we have added an EA to Genr8. The EA will automatically create valid rewrite rules and the user can exert high-level control over the process through the fitness function. This has the benefit that the user does not have to know anything about HEMLS or how the surfaces are created. It is merely required that the user has a conceptual understanding of the nature of EAs and the growth process.

5.1 Grammatical Evolution

There are many variants of the EA paradigm, particularly in how they represent the genome or how they handle selection. The specific EA used in Genr8 is called Grammatical Evolution (GE). It was introduced by Ryan and O'Neill in 1997 [35] and it combines the strengths of genetic algorithms (GAs) [33] and genetic programming (GP) [24]. What makes GP such a powerful algorithm is that it allows us to evolve executable structures represented as trees directly. Unfortunately, the genetic operations often become very complicated as one has to deal with subtrees which must be ensured to be compatible when swapped. GAs on the other hand are very convenient when applying the genetic operators since the genome is represented as an array of integers. The drawback is that one is restricted to work with integers. GE uses the same representation of the genome as GAs. The significant invention in GE is that it can be applied to *any* language whose grammar can be represented in Backus-Naur Form. When interpreting a genome, the genes are used to select production rules as the sentence is expanded (see example in Section 5.2). The expansion of the genome via the BNF can be viewed as a tree similar to those used in GP. When applying genetic operators to these structures there would be considerable overhead if we had to ensure that the swapping of two subtrees results in a syntactically correct outcome. However, genetic operators can be directly applied to the linear genome without any syntactic violations. Thus, GE provides a strict separation between the genetic operations and the language that we wish to use. The constraints of the language are automatically and efficiently handled by the BNF representation of the grammar.

The BNF used by the EA is more restricted than the one that Genr8's parser can handle (Figure 4). This is because we want to facilitate the search. The full BNF provides too many options and we have found that the EA finds interesting results faster using the restricted BNF. An additional reason for restricting the BNF for the EA is that some of the non-standard features of the BNF would be quite complicated to implement with GE. Despite the simplifications, the BNF in Figure 6 is still more complicated than the ones mostly used in GE. For this reason, we had to extend the standard GE in order to handle the more complicated features of the BNF. Below we present a summary of our extensions and modifications of GE. For more details, see [16].

In BNF notation, the symbols { ... } indicate that the string appearing between them can be repeated zero or more times (the symbols [...] have a similar function in that the string can be used once or not at all). This feature is used extensively in Genr8, for instance it allows us to have an arbitrary number of rewrite rules (line `8` in Figure 6). Because of the importance of genetic inheritance and the propagation of genetic characteristics in an EA, it is important that this optional quality is consistent through all decodings of the genotype (i.e. from one generation to the next or for multiple copies of a genotype in the population). Moreover, we want a scheme where there is no fixed upper limit on the number of times the string is used. The best way to achieve this consistency is to use the genotype itself to determine how many times a string should be expanded. In Genr8 this is accomplished with a simple algorithm that works as follows:

1. When a { is encountered, a counter is initialized to zero.
2. Read a gene from the genome and if `gene mod ( counter + 2 ) <= counter` do not write any more of the optional symbols.
3. Write the string of optional symbols (until the }).
4. Increase the counter and go back to step 2.

The above scheme gives the same result for identical sequences of genes and there is no fixed upper limit on the number of expansions although

```
N = { <L-System>, <Axiom>, <RewriteRule>, <Predecessor>,
      <Successor>, <Modifier>, <AngleValue>, <BranchAngleValue> }
T = { +, -, &, ^, \, /, '[', ']', '<', '>', ->, Edge, ';',
      Angle, Sync, EdgeX, BranchAngle, 15, 30, 45, 60, 75 }
S = { <L-System> }
P = {
<L-System> ::= <Axiom> ';' <RewriteRule> ';'
               { <RewriteRule> ';' } Angle <AngleValue>
               ';' [ Sync ';' ] BranchAngle <BranchAngleValue>
<Axiom> ::= Edge [ ~ ] + Edge [ ~ ] + Edge [ ~ ] + Edge
            { [ ~ ] + Edge }
<RewriteRule> ::= <Predecessor> -> <Successor>
<Successor> ::= { <Modifier> } <Segment>
<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. 6 The BNF for HEMLS used by the EA. It is more restricted than the one in Figure 4 to speed up the search. There are fewer non-terminals in the restricted BNF. The `<Segment>` non-terminal is no longer used since we are no longer using time-dependent rewrite-rules. Furthermore, we have removed the probabilistic extension and we have restricted the choice of angle values.

the probability of adding to the expansion decreases with the number of strings expanded. The constant 2 in the second step above can be adjusted to change the expected number of times we will go through the loop.

The reader may recall that the `Edge` terminal requires an additional integer parameter. When the expansion of the rewrite system results in an `Edge`, we use a scheme similar to the one for multiple terminals to select the ID for the `Edge`. The system has a global counter, `max_ID` that keeps track of the total number of edge IDs used so far. When an `Edge` symbol is encountered, the ID is determined by the value of the next gene and `max_ID` through the relationship `ID = geneValue mod ( max_ID + 1 )`. As for the optional number of symbols, this scheme enables the system to introduce new edge types at a marginally decreasing rate.

Another heuristic is introduced by the `EdgeX` terminals. If a production contains an `EdgeX` terminal all those terminals will get the same ID in a manner that is akin to variable binding in PROLOG. This is used in lines `26-27` where it is ensured that the branches will always come as pairs with the same ID. This makes it more likely that the branches will join in the rewrite system. Joining branches is crucial since they provide the internal structure of the surface. If the branches do not join, the surface will simply consist of a perimeter, making it very dull.

When the ID of the predecessors is chosen randomly, it is likely that the rewrite system will include edge types without an explicitly defined rewrite rule. This is often undesired since these edges will not be modified during the growth process. To overcome this issue, we have introduced a repair mechanism in the Genr8 EA. When the rewrite system has been expanded, the algorithm checks to make sure that all edge types have an explicit rewrite rule. If not, the rewrite system is extended by the addition of new rewrite rules.

One problem of using GE to map the genome to a rewrite system is that the expansions tend to get very long. This is not a problem that is inherent to GE, but an effect of the HEMLS grammar. In particular, it is caused by the productions for the `<Modifier>` non-terminal (lines `18-27` in Figure 6) where seven of the ten productions contain a new `<Modifier>`. To combat this hefty expansion, Genr8 includes a new mechanism for restricting the length of the expanded grammars. The expanded rewrite system can be viewed as a tree where the leaves are terminals and the internal nodes are non-terminals (see Figure 7). We can keep track of the depth of the tree during the expansion to measure how large the tree is. When the expansion reaches a specified limit `max_depth`, the expansions are brought to a halt. This is achieved by changing the way productions are chosen. Instead of choosing from all the production rules, only the subset of rules that do not contain any non-terminals are used For the `<Modifier>` it means that only lines `18` and `26-27` will be used. This scheme allows the user to control the size of the rewrite systems generated by the EA while at the same time making sure that the genome will be consistently mapped.

Because of the use of GE and BNF notation, it is powerful yet simple to enable the EA to evolve additional classes of rewrite rule sets. For example, we have implemented two additional grammars in Genr8 that are minor variations of the one in Figure 6. The first of these grammars allows the EA to evolve probabilistic rewrite rules. The second grammar is more constrained than the default grammar and generates symmetrically balanced surfaces. It ensures that there will always be at least one axis of symmetry if the surface is undistorted by the environment. To achieve this, the seed is enforced to be symmetric and two new terminals, `EdgeY` and `EdgeZ` analogous to `EdgeX` are introduced. The grammar enforces operations prescribed by the rewrite rules to preserve the symmetry.

5.2 Example expansion

To illustrate the mapping of a genome to a rewrite system, we shall give a brief example, showing part of the expansion procedure. The genome that we shall be mapping starts with

$$\texttt{212}, \texttt{187}, \texttt{632}, \texttt{832}, \texttt{800}, \texttt{517}, \texttt{338}, \texttt{39}, \texttt{878}, \texttt{185}, \texttt{954}, \texttt{863}, \ldots.$$

To expand a rewrite system, we begin with the start symbol `<L-system>`. There is only one production rule, so the gene value (`212`) is irrelevant in this case. The rewrite system is expanded to

```
<Axiom> ';' <RewriteRule> ';'{ <RewriteRule> ';' }[ Sync ';' ]
Angle <AngleValue> ';' BranchAngle <BranchAngleValue>
```

Next, the axiom is expanded using the gene with value `187` and again there is only one production available. We proceed to determine the type of the first `Edge` in the `<Axiom>`. When the EA is initialized, there are two edge types (with IDs `0` and `1`), so the type of the edge is determined by `632 mod 3 = 2` (thus introducing a new edge type). There are two more `Edge` terminals and their types are determined by `832 mod 4 = 0` and `800 mod 4 = 0`. The next four symbols are { `'+'` `'Edge'` } indicating that the sequence `'+'` `'Edge'` should occur zero or more times. A counter to keep track of how many times the string has featured is initialized to zero. The counter is used to determine whether or not another string should be expanded. The test for adding another string is `gene mod (counter + X) > (counter + Y)` where `X` and `Y` are parameters that can be adjusted to determine the expected number of expansions. In Genr8, `X = 0` and `Y = 2` which gives `517 mod 2 = 1 > 0`, i.e. another `'+'` and `Edge` should be added. The type of the `Edge` is set to `338 mod 4 = 2` and testing for further expansions we find that `39 mod 3 = 0 < 1`. This yields a rewrite system of the form:

```
Edge2 + Edge0 + Edge0 + Edge2
<RewriteRule> ';'
{ <RewriteRule> ';' }
[ Sync ';' ]
Angle <AngleValue> ';'
BranchAngle <BranchAngleValue> ';'
```

The expansion of the rewrite systems can be graphically viewed as a tree as shown in Figure 7. So far we have expanded the axiom of the rewrite system and we see that it corresponds to a square with two types of `Edge`s. The expansion continues by using additional genes to replace the `<RewriteRule>` non-terminal with a predecessor and a successor. Subsequently we may add an optional number of rewrite-rules before specifying the parameters of the rewrite system.

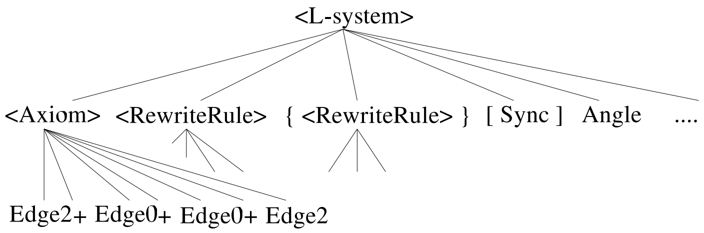

Fig. 7 An example of how the expansion of the genome, via the BNF in Figure 6, can be viewed as a tree. Internal nodes are non-terminals and the rewrite system can be obtained by reading the terminals at the leaves.

5.3 Fitness Evaluation

A crucial component of an EA is the fitness evaluation which guides the search towards better solutions. In design, there is no general way of algorithmically defining a 'good' surface. Coming up with a useful fitness evaluation scheme for design applications is still an open research question [39] [45]. The most common approach is that of Interactive Evolutionary Computation (IEC) where the user acts as fitness function and evaluates every design [45]. The main drawback of IEC is that the search space is restricted since it is tedious for the user to evaluate surfaces [42]. Another difficulty stems from the user being inconsistent in his or her evaluations [10]. An alternative approach is to try to use an artificial neural network or some other learning algorithm to learn the user's preferences. This implies mapping the complex high-dimensional tastes of the user to a low dimensional representation. Unfortunately, this scheme usually does not work very well in practice [39].

In Genr8 we use a fitness evaluation scheme that gives the user high level control of the evolutionary search. In practice this is achieved by a multi-parametric fitness function. Each parameter represents a specific feature of the surface. The user may set target values for each parameter as well as weights to determine the importance of each criteria. The total fitness is $F_{tot} = \sum w_i F_i$, where i runs over the six different criteria and the weight w_i is a non-negative real number indicating the relative importance of each criterion. The user can modify both the target values F_i and the weights at any point during the run, providing for a high degree of interactive flexibility.

- **Size**. Measures the size of the surface in the x and y directions. It is defined as $F_{size} = \max_{i,j} |x_i - x_j| + \max_{i,j} |y_i - y_j|$, where i and j run over all the vertices.
- **Smoothness**. A local measure of the variation in the $z-$direction. The smoothness is defined as

$$F_{smooth} = \frac{1}{2}\sum_{i=0}^{N}\sum_{j=0}^{N_i} |z_i - z_j|,$$

where z_i is the coordinate value for node i, N is the total number of nodes and N_i the number of neighbors for node i.
- **Soft boundaries**. If this criterion is used, the surface is allowed to grow through boundaries. However, it occurs at a fitness penalty for each node which is on the wrong side of the boundary.
- **Subdivisions**. This measure is defined as the number of vertices divided by the number of edges. Thus, it will be 1 for a surface which does not have any internal structures (such as the top left one in Figure 3) and lower for surfaces with many internal vertices and edges.
- **Symmetry**. This metric is a rough way of assessing the degree of symmetry of the surface. It is defined as $F_{sym} = (sym(x) + sym(y))/2$, where the function $sym(x)$ returns the ratio of the number nodes at either side of the line parallel to the x-axis running through the centre of the surface.
- **Undulation**. This measure is a global measure of the variation in the z-direction. It is defined as $F_{und} = \max_{i,j} |z_i - z_j|$.

It is important to point out that in most situations there are many different ways to attain a given fitness value. That is, the fitness function is degenerate in mapping the surfaces to a single fitness value. Consequently, there are many different surfaces which are equally good solutions for a given set of fitness criteria. This is usually advantageous since it makes it easier to maintain a diverse population. Moreover, some of the criteria are more or less in conflict with each other. This means that the EA must negotiate a trade-off between the different criteria. These situations lead to the most interesting outcomes and also help increasing the variation in the population.

6 Genr8's Surfaces

Evolutionary design tools differ from mainstream EA applications since they are not trying to solve optimization problems in the traditional sense. Instead, the focus has been shifted from optimization to exploration. We also note that it is hard to define objective criteria to measure the success of applying an EA to a specific design problem. We consider Genr8 a successful design tool since it is being actively used by architects at universities around the world.

Genr8 is a novel software tool which operates along different principles than most other design tools available today. Thus, it provides designers with a whole new range of opportunities as well as challenges. The first problem for the designer is to obtain a proper understanding of the underlying theories of L-systems and EAs so that he or she can use the tool to best effect. Our experience from observing students using Genr8 shows that this can be significant problem since most architects lack the necessary computer science background. Nevertheless, many students manage to be very creative with the tool despite conceptual confusion of the workings of the

Fig. 8 An example of a surface grown using the square rewrite system in Section 4.4 from the "Butterfly machines" project by Steve Fuchs at the Southern Californian Institute of Architecture. One of the surfaces was later used as the starting point for the design of a chair. The environment contains five attractors, one repellor and gravity.

growth algorithm and the EA. Another problem stems from the fact that the space of possible surfaces is huge. Due to the structure of the HEMLS, the user is forced to search for the actual form and its representation at the same time. The tool would probably be more powerful if these two issues were more separated so that the user could focus on one at a time. The ultimate solution to this problem would involve a Lamarckian interaction whereby the user can make modifications of the phenotype and have them mapped back to the genotype. We note that given certain restrictions to the grammar, it is possible to find a unique genome that will produce a given rewrite system. However, the non-linear mapping from the rewrite-system, influenced by the environment, to the final surface makes it impossible to reverse-engineer the rewrite system from the surface.

We end this section with a short presentation of three projects that have used Genr8. More information on these and three additional projects can be found in [17]. Since the aim of the architects is to produce novel creative designs they do not report details of the parameter settings in their experiment. There is simply no desire in their community to allow or enable other people to be able to repeat the experiment. Furthermore, Genr8 is just one of the many tools used in the design process. There is often a significant amount of work remaining (e.g. lofting, rendering, etc) once the final surface has been grown. The first project we only mention briefly by pointing out that it does not use the EA. It only employs the pre-defined square rewrite system. An screen shot from Steve Fuchs' project can be found in Figure 8.

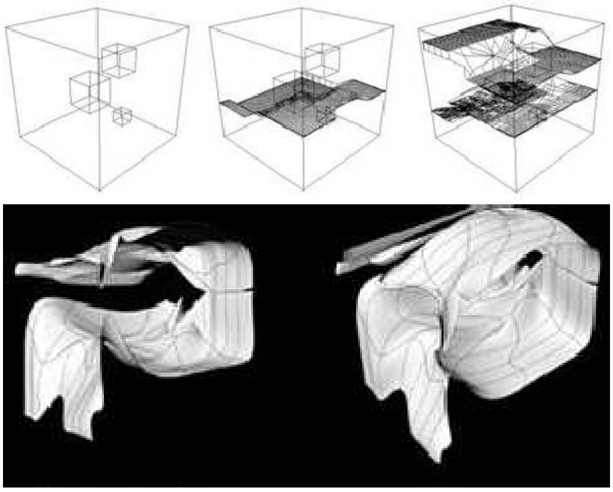

Fig. 9 (Top) The environment set up by Michel da Costa Goncalves for his Genr8 project consisting of four cubes, repellors (not shown) and gravity. The preliminary experiments were used to calibrate the parameters so that the surface would fill the bounded space. (Bottom) The surfaces evolved in the environment in the top panel were exported from Maya and post-processed to form a set of lofted wire frames.

In a project by Michel da Costa Goncalves, the environmental setup consists of four cubes as shown in Figure 9. The first step was to calibrate the parameters and the lengths of the boundary cubes in such a way that the surface will fill the bounded space. A number of repellors as well as a gravity component were also introduced in order to provide a conceptual setup for the sketching of a house. Next, two surfaces were evolved in this environment as illustrated in Figure 9. The resulting surfaces were exported to Rhino [4] where they were transformed, lofted and smoothed.

A third project is a pneumatic strawberry bar designed by Achim Menges of the Architectural Association, London, UK [36]. Recognizing the difficulties of incorporating material analysis in Genr8, he worked with inflatable structures which allowed him to consider the Genr8 surface as a membrane. Based on an external structural evaluation of the surfaces, he was able to modify the parameters in Genr8 in such a way that better surfaces would be produced. Menges set up an iterative scheme whereby he would evolve a population inside a bounding box for a number of generations. Then he would use one of the evolved surfaces as boundary for the subsequent evolution, thereby gradually refining the evolved surface. Genr8's high-level fitness function allowed Menges to perform much longer runs involving hundreds of individuals over thousands of generations.

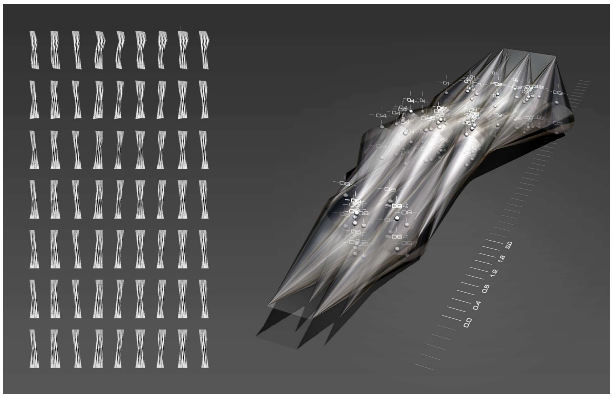

Fig. 10 Left, an overview of some of the surfaces obtained during the evolutionary process of creating the strawberry bar. The righthand image shows a rendering of the final result. The bar does not consist of one surface, but of three different ones melded together.

7 Conclusions

Architects have long strived to express aesthetically pleasing organic form within the character of their buildings. Due to their recognition of Nature's means, they have embraced both growth processes as a means of deriving such form and an evolutionary process as a means of exploring and creating ideal form. Genr8 is a creative design tool for architects which provides them with a growth model for surfaces and means of evolutionary discovery. To meet the ambitions of architects, at Genr8's technical core is the combination of HEMLS and an evolutionary algorithm. HEMLS are an extension of map L-systems that generate (or grow) 3D surfaces interactively within a bounded environment that contains attractors and repellers. HEMLS are described using context-free grammars that themselves can be described in Backus-Naur Form. Genr8's EA uses the BNF description in tandem with Grammatical Evolution to evolve rewrite systems in lieu of hand written ones. The EA's fitness function is a weighted combination of specific surface features.

8 Acknowledgements

We would like to thank Peter Testa, Devyn Weiser, Simon Greenwold and the rest of the EDG for their help in developing this project. We are grateful to Michael Hensel, Achim Menges and Mike Weinstock at the Architectural Association for many stimulating and interesting discussions. We would also like to thank Michel da Costa Goncalves, Steve Fuchs and all the other users that have given us valuable feedback about the software. Finally, we would like to thank Professor Przemyslaw Prusinkiewicz for kindly providing permission to use the images in Figure 2.

References

1. Hal Abelson and Andrea diSessa. *Turtle geometry: The computer as a medium for exploring mathematics*. MIT Press, 1980.
2. Jean-Paul Allouche and Jeffrey Shallit. The ubiquitous Prouhet-Thue-Morse sequence. In C. Ding, T. Helleseth, and H. Niederreiter, editors, *Sequences and their applications, Proceedings of SETA'98*, pages 1–16, 1999.
3. Abimbola O. Asojo. Exploring algorithms as form determinants in design. In *Proceedings of the 3rd international space syntax symposium*, volume 3, Atlanta, USA, 2001.
4. Robert McNeel & Associates. Rhino, http://www.rhino3d.com/.
5. Peter Bentley and David Corne, editors. *Creative Evolutionary Systems*. Morgan Kaufmann, 2002.
6. Peter J. Bentley, editor. *Evolutionary Design by Computers*. Morgan Kaufmann, May 1999.
7. Peter J. Bentley and Una-May O'Reilly. Ten steps to make a perfect creative evolutionary design system. In *GECCO 2001 Workshop on Non-Routine Design with Evolutionary Systems*, 2001.
8. Terence Broughton, Paul Coates, and Helen Jackson. Exploring 3D design worlds using lindenmayer systems and genetic programming. In Peter J. Bentley, editor, *Evolutionary Design by Computers*, chapter 14. Morgan Kaufmann, May 1999.
9. Hong-Sheng Chen. Generation of three-dimensional cellular automata. In *Generative Art*, Milan, Italy, 2003.
10. Dragan Cvetkovic and Ian C. Parmee. Preferences and their application in evolutionary multiobjective optimization. *IEEE Transactions on evolutionary computation*, 6(1):42–57, 2002.
11. Eric H. Davidson and Douglas H. Erwin. Gene regulatory networks and the evolution of animal body plans. *Science*, 311:796–800, 2006.
12. John Frazer. *An Evolutionary Architecture*. Architectural Association, London, 1995.
13. Frédéric Gruau. Genetic micro programming of neural networks. In Kenneth E. Kinnear, editor, *Advances in Genetic Programming*, pages 495–518. MIT Press, Cambridge, MA, 1994.
14. Peter Hammerstein, Edward H. Hagen, Andreas V. M. Herz, and Hanspeter Herzel. Robustness: A key to evolutionary design. *Biological Theory*, 1(1):90–93, 2006.
15. Martin Hemberg. Genr8 - a design tool for surfaces. Master's thesis, Chalmers University Of Technology, 2001.
16. Martin Hemberg and Una-May O'Reilly. Extending grammatical evolution to evolve digital surfaces with Genr8. In Maarten Keijzer, Una-May O'Reilly, Simon M. Lucas, Ernesto Costa, and Terence Soule, editors, *Genetic Programming 7th European conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 299–308, Coimbra, Portugal, 2004. EvoNet, Springer-Verlag.
17. Martin Hemberg, Una-May O'Reilly, Achim Menges, Katrin Jonas, Michel da Costa Goncalves, and Steve Fuchs. Genr8: Architects' experience using an emergent design tool. In Penousal Machado and Juan Romero, editors, *The Art of Artificial Evolution*. Springer, 2007.
18. Gregory S. Hornby and Jordan B. Pollack. The advantages of generative grammatical encodings for physical design. In *IEEE Congress on Evolutionary Computation*. IEEE Press, 2001.

19. ISO. *ISO/IEC 14977 : 1996(E)*, 1996.
20. Christian Jacob. Genetic L-system programming. In *Parallel problem solving from Nature*, volume 866 of *Lecture notes in computer science*, pages 334–343. Springer Verlag, 1994.
21. Patrick Janssen, John Frazer, and Ming Xi Tang. Evolutionary design systems and generative processes. *Applied intelligence*, 16(2):119–128, 2002.
22. Colin G. Johnson and Juan Jesus Romero Cardalda. Evolutionary computing in visual art and music. *Leonardo*, 35(2):175–184, 2002.
23. Rafal Kicinger, Tomasz Arciszewski, and Kenneth A. De Jong. Morphogenic evolutionary design: cellular automata representations in topological structural design. In Ian C. Parmee, editor, *Adaptive computing in design and manufacture*, volume VI, pages 25–38, 2004.
24. John R. Koza. *Genetic programming: on the programming of computers by means of natural selection.* MIT Press, 1992.
25. 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.
26. Robert Krawczyk. Architectural interpretation of cellular automata. In *Mathematical Connections in Art, Music, and Science,*, ISAMA/Bridges, Granada, Spain, 2003.
27. Sanjeev Kumar and Peter J. Bentley, editors. *On growth, form and computers.* Elsevier, 2003.
28. Aristid Lindenmayer. An introduction to parallel map generating systems. In *Proceedings of the 3rd International Workshop on Graph-Grammars and Their Application to Computer Science*, volume 291 of *Lecture Notes In Computer Science*, pages 27–40, 1986.
29. Aristid Lindenmayer and Grzegorz Rozenberg. Parallel generation of maps: Developmental systems for cell layers. In *Proceedings of the International Workshop on Graph-Grammars and Their Application to Computer Science and Biology*, volume 73 of *Lecture Notes In Computer Science*, pages 301–316, 1978.
30. Autodesk Maya. http://usa.autodesk.com/.
31. Jon McCormack. Aesthetic evolution of L-systems revisited. In *Applications of Evolutionary Computing (Evoworkshops 2004)*, pages 477–488, 2004.
32. Jeremy J. Michalek, Ruchi Choudhary, and Panos Y. Papalambros. Architectural layout design optimization. *Engineering optimization*, 34(5):461–484, 2002.
33. Melanie Mitchell. *An introduction to genetic algorithms.* MIT Press, 1996.
34. William J Mitchell. *The logic of architecture.* MIT Press, 1990.
35. Michael O'Neill and Conor Ryan. *Grammatical Evolution - Evolving programs in an arbitrary language.* Kluwer Academic Publishers, 2003.
36. Una-May O'Reilly, Martin Hemberg, and Achim Menges. Evolutionary computation and artificial life in architecture: Exploring the potential of generative and genetic algorithms as operative design tools. *Architectural design*, 74(3):48–53, 2004.
37. Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The algorithmic beauty of plants.* Springer-Verlag New York, Inc., 1996.
38. Przemyslaw Prusinkiewicz, Lars Mundermann, Radoslaw Karwowski, and Brendan Lane. The use of positional information in the modelling of plants. In

International Conference on Computer Graphics and Interactive Techniques, volume Proceedings of the 28th annual conference on computer graphics and interactive techniques, pages 289–300, 2001.

39. Juan Romero, Penousal Machado, Antonio Santos, and Amilcar Cardoso. On the development of critics in evolutionary computation artists. In *EvoMusart workshop, 6th European Conference on Genetic Programming*, Essex, April 2003.
40. Mike Rosenman and John Gero. Evolving designs by generating useful complex gene structures. In Peter J. Bentley, editor, *Evolutionary Design by Computers*, chapter 15. Morgan Kaufmann, May 1999.
41. Grzegorz Rozenberg and Arto Salomaa. *Handbook of formal languages.* Springer-Verlag, 1997.
42. Tomoya Sato and Masafumi Hagiwara. Idset: Interactive design system using evolutionary techniques. *Computer-Aided Design 33*, pages 367–377, 2001.
43. Karl Sims. Evolving three-dimensional morphology and behaviour. In Peter J. Bentley, editor, *Evolutionary Design by Computers*, chapter 13. Morgan Kaufmann, May 1999.
44. George Stiny and James Gips. Shape grammars and the generative specification of painting and sculpture. In C. V. Freiman, editor, *Information Processing*, volume 71, 1972.
45. Hideyuki Takagi. Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation. *Proceedings of the IEEE*, 89(9):1275–1296, 2001.
46. Patrick D. Terry. *Compilers and compiler generators.* International Thomson Computer Press, 1996.
47. Peter Testa, Una-May O'Reilly, Markus Kangas, and Axel Kilian. Moss: Morphogenetic surface structure - a software tool for design exploration. In *Proceedings of Greenwich 2000; Digital Creativity Symposium*, 2000.
48. D'Arcy Thompson. *On growth and form.* Cambridge University Press, 1917.
49. William T. Tutte. *Graph theory.* Cambridge University Press, 1984.
50. Nikolaos Vassilas, George Mialoulis, Dionysios Chronopoulos, Elias Konstantinidis, Ioanna Ravani, Dimitrios Makris, and Dimitri Plemenos. MultiCAD-GA: A system for the design of 3D forms based on genetic algorithms and human evaluation. In *Methods and Applications Of Artificial Intelligence: Second Hellenic Conference on AI.* Springer, 2002.
51. John von Neumann. *The theory of self-reproducing automata.* University of Illinois Press, 1966.
52. Xiatong Wang, Ming Xi Tang, and John Frazer. Creative stimulator: An interface to enhance creativity in pattern design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 15:433–440, 2001.
53. Niklaus Wirth. What can we do about the unnecessary diversity of notation for syntactic definitions? *Communications of the ACM*, 20(11):822–823, 1977.
54. Tong Zhou, Jean M. Carlson, and John Doyle. Mutation, specialization, and hypersensitivity in highly optimized tolerance. *Proceedings of the National Academy Of Sciences*, 99(4):2049–2054, February 2002.