GENR8 - A Design Tool for Surface Generation

by

Martin Hemberg

Author .....
June 29, 2001

Certified by .....
Una-May O'Reilly
Research Scientist
Thesis Supervisor

Accepted by .....
Peter Nordin
Assistant Professor

GENR8 - A Design Tool for Surface Generation

by

Martin Hemberg

Submitted to the Department of Physical Resource Theory
on June 29, 2001, in partial fulfillment of the
requirements for the degree of
Master of Science Engineering Physics

Abstract

GENR8 is an architect's design tool that generates surfaces. It is powerful and innovative because it fuses expressively powerful universes of growth languages with evolutionary search. Unlike traditional CAD-tools, GENR8 can create new designs and help the user to come up with new ideas.

Developed via the API of Alias|Wavefront's Maya, it combines 3D map L-systems, that are extended to an abstract physical environment with evolutionary computation. GENR8 uses Grammatical Evolution and a BNF of the grammar to specify the grammar that governs the growth. GENR8 addresses key issues arising from exploiting evolutionary adaption within a creative interactive tool framework. EAs typically adapt 'off-line' but GENR8 is designed to sensitively accommodate the nature of the back and forth control exchange between user and tool during on-line evolutionary adaptation.

GENR8 addresses how users may interrupt, intervene and then resume an EA tool. It also forgoes interactive subjective design evaluation for computationalized multi-criteria evaluation that permits wider search in shorter time spans.

Thesis Supervisor: Una-May O'Reilly
Title: Research Scientist

Acknowledgments

I would like to thank all my advisors. Una-May O'Reilly who has been my principal advisor during this work. I have learned a lot from her and she has been of great help. Peter Testa, the head of the Emergent Design Group, for taking me on as a member of the group and Peter Nordin at Chalmers whose connections made it possible for me to come to MIT in the first place.

I am also grateful for the support of the other members of the Emergent Design Group, in particular Simon Greenwold, Devyn Weiser and Janet Fan.

Finally I would like to thank everyone who has supported me on this work. Here I would like to mention my family; my mother, my father and my brother.

Contents

1 Introduction **15**

1.1 Background and motivation 15

1.2 Purpose 17

1.3 Description of GENR8 17

1.4 Related work 18

1.5 Contributions 21

1.5.1 Computer Science 21

1.5.2 Architecture 22

2 Growth Model **25**

2.1 L-systems 25

2.2 Turtle rules 27

2.3 Brackets 28

2.4 Map L-systems 28

2.5 Hemberg Extended Map L-systems 30

2.5.1 Branches 31

2.5.2 Context sensitivity 33

2.5.3 Time variation 33

2.5.4 Stochastic rules 34

2.6 Implementation of GENR8 34

2.6.1 Internal representation 35

2.6.2 Maya representation 36

2.7 Environment 36

2.7.1 External forces . . . 38
2.7.2 Internal forces . . . 40
2.7.3 Boundaries . . . 43
2.7.4 Parameters . . . 45

3 Evolution **47**
3.1 Evolutionary Algorithms . . . 47
3.1.1 Grammatical Evolution . . . 48
3.2 BNF-grammar . . . 50
3.2.1 BNFs for the EA . . . 52
3.2.2 Mapping the genotype to BNF grammar . . . 55

4 GENR8 as a tool **57**
4.1 Integration into Maya . . . 59
4.1.1 Setting up the environment . . . 59
4.1.2 Examining the output . . . 61
4.2 Interruption, intervention and resumption . . . 61
4.2.1 regn8 . . . 63
4.3 Design evaluation . . . 64
4.3.1 Fitness function . . . 65

5 Future Work **69**
5.1 Solids . . . 69
5.2 3D-Visualization . . . 69
5.3 Learning User Behaviour . . . 70

6 Conclusions **71**
6.1 Computer science . . . 71
6.1.1 ALife . . . 71
6.1.2 Evolutionary computation . . . 71
6.1.3 Combining EC and HEMLS . . . 72
6.2 Architecture . . . 72

A BNF Specifications **73**

A.1 Default 73

A.2 Reversible 75

A.3 Symmetric 76

A.4 Probabilistic 78

B Example of how a genotype is mapped to a grammar **81**

List of Figures

1-1 The field of Emergent Design. It is the union of EC, ALife and design. 16

1-2 The Guggenheim Museum in Bilbao, a building where novel materials and techniques have been used (Frank O Gehry and partners, 1997). . 23

2-1 An example of an ordinary L-system. The string is interpreted using turtle rules to produce a realistic image of a plant. 29

2-2 A simple map L-system. The productions are shown in the top row. The successive application of these productions to the seed, leftmost in the middle row, are shown in the two bottom rows. 30

2-3 This is the same grammar and environment as in Figure 2-5, only the branch merging mode has been changed from asynchronous to synchronous. 32

2-4 A schematic view of GENR8 and how it is incorporated into Maya. . 35

2-5 A surface grown in an empty environment 37

2-6 The growth of a surface in an environment with five repellors. 37

2-7 The displacement as a function of distance for an attractor/repellor. . 39

2-8 This surface is pulled downwards by gravity, but it is blocked by the sphere. 39

2-9 A surface where the cells have been made uniform, this is to be compared with the other figures in this chapter, where the cells tend to be subdivided by straight lines. 40

2-10 A surface with fixed perimeter in an environment with a repellor. The internal vertices are free to move and they are pushed away by the repellor. 41
2-11 A surface with fixed center in the vicinity of an attractor. 42
2-12 A surface where the random noise introduces some ruggedness. 43
2-13 Surface inside a bounding box. The repellor in the upper left corner pushes the surface into the corner of the bounding box. 44
2-14 Surface inside a bounding box. The repellor in the upper left corner pushes the surface into the corner of the bounding box, but once the perimeter of the surface hits the walls, they stop moving (compare with Figure 2-13). 44

3-1 A comparison between biology, grammatical evolution and GENR8. . 49
3-2 A surface that was evolved using the symmetric BNF. 53
3-3 A surface that was evolved by the system from a population size of 15 after 20 generations 55

4-1 The Koch-curve, also known as the snow-flake. 58
4-2 A screenshot of the GUI. Menus can be expanded or collapsed by clicking on the bars with the arrows. 60
4-3 A screenshot of the output window. The visibility of the individuals is turned on and off as the menus are expanded or collapsed. 62
4-4 A rugged surface. The view is from the side. 66
4-5 A surface with many subdivisions. 67
4-6 A randomized starting individual. Since no branches have merged, there are no subdivisions. 68

List of Tables

1.1 Categorization of the related work. 20

2.1 Turtle commands and their meaning. 28

2.2 Context sensitivity for the L-systems in GENR8. 33

2.3 A simple stochastic time-varying context-sensitive map L-system. . . 34

Chapter 1

Introduction

1.1 Background and motivation

Evolutionary algorithms have traditionally been used to solve optimization problems. However people have tried to use them for creative purposes, for instance to generate artificial life forms [28] or as a design aid. Lately they have been used for evolutionary art [29] [12], helping the artist to create new forms by exploring a wide range of forms. In evolutionary design (ED), focus is less on aesthetics, but more on the functional and engineering perspectives of creating artifacts. Bentley gives an overview [3] and a description of evolutionary design projects [1].

This thesis is in the field of emergent design; it is an area that combines evolutionary computation (EC), artificial life (ALife) and design. The concept of the field is to combine the generative and combinatorial aspects of ALife with an evolutionary search component. This allows us to generate novel designs bottom-up, combining primitive elements in a non-linear fashion.

Emergence can be defined as (there is no formal definition) 'the whole is more than the sum of the parts'. This means that we can not predict the outcome of the system just by studying the mechanics and constituents at the local level. The most striking example is perhaps the human brain. Each brain cell is in itself incapable of thought; it is an emergent property of the brain as a whole.

The thesis is part of the work of the Emergent Design Group at MIT [7]. It unites

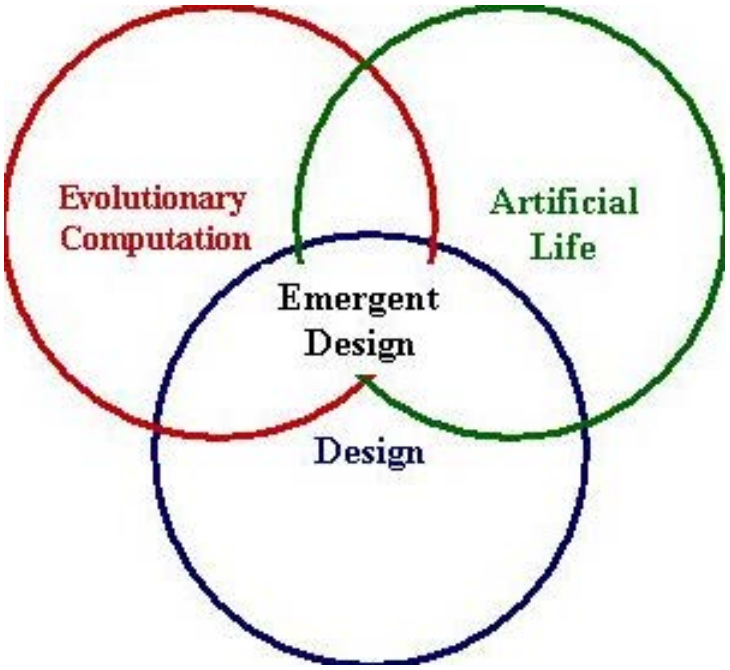

Figure 1-1: The field of Emergent Design. It is the union of EC, ALife and design.

architects from the School of Architecture and Planning and computer scientists from the Artificial Intelligence Lab (and visiting students from Chalmers). The architect members of the group embrace a process of design that stresses emergence. The task of the computer scientists is to develop new software tools that allow the architects to explore new designs with the aid of emergent techniques. These investigations entail such different realms as morphology, material, structure and program.

Developing this kind of software is obviously a challenging task in itself for a computer scientist, but it also entails some interesting points within the computer science (CS) field. Complex systems and ALife have previously been used mostly in pure research, to investigate various phenomena, like flocking [21], plant growth [20] and chemical reactions. With GENR8 we are trying to use these techniques as a foundation for a software application. GENR8 solves its task, creating surfaces, with a complex systems' approach. This is a necessary step in order to bring complex systems out of the world of academia and use them as a basis for new technology, where they have great potential.

More information and the latest news about GENR8 (including the tool itself and

the source code) can be found at the GENR8 Official Website (http://www.ai.mit.edu/projects/emergentDesign/genr8/).

1.2 Purpose

One purpose of this thesis is to create a tool that is useful to a designer. It is certainly beyond the scope of this thesis to create a tool that takes all aspects of architectural design in consideration. The focus of GENR8 is on growth, and in particular we want to mimic biological growth. Architects are intrigued by natural form and they find organic qualities in design very compelling. Furthermore, the architects desire a growth process that takes place in a specific environment and reacts to the elements of the environment.

To narrow it down even further, we decided to create a tool that generates surfaces. Surfaces are obviously very useful to a designer and they can be used in a wide range of contexts; from buildings and cars to handbags and remote controls.

1.3 Description of GENR8

GENR8 is a surface modeling tool that simulates organic growth of surfaces in an environment. We found map L-systems to be suitable model for this kind of growth. However, we had to extend the map L-system model [20] to make it work in three dimensions, and we have added environmental elements that influence and interact with growth. The details of GENR8's growth process will be described in Chapter 2.

The universe of possible surfaces described by extended map L-systems is overwhelming and it is impossible to explore by hand. It is also very hard to construct a map L-system grammar by hand and we needed to automate the process; so that a designer can merely look at the final result, the surface. To search the universe of possible surfaces, GENR8 uses evolutionary computation. Using evolutionary algorithms (EAs) has two benefits; the first is that the parallel population based search gives us multiple solutions to the design problem. This is a feature that was requested by the

designers; they do not want just one solution, but several alternatives on the same theme. Evolutionary algorithms also accommodate discovery within the extremely large universe of surfaces; it is selective and explorative yielding adaptation and discovery. GENR8 uses an EA called grammatical evolution (GE), it will be discussed in Chapter 3.

Yet, EAs also have short-comings. The fitness evaluation is a particularly tricky part for creative evolutionary systems. In GENR8 we forgo a subjective user inspection and approach via ranking an automated solution that must embody quantitative criteria. Fitness evaluation will be discussed in Section 4.3.

Another issue for evolutionary design tools is control of the creative process. Who has control, the tool or the user? The traditional way is to set up the evolutionary run, let it run to completion and then give the user access to the final output. However, architects desire more control of the creative process. They want to be able to direct the system and have it react to their changing desires at any point in the design process. In Section 4.2, we discuss how user tool control is negotiated and implemented in GENR8.

If GENR8 is to have any practical value to a designer it must fit into the architects' tool box. Architects have a wide range of tools that can be used for different purposes and in different stages in the design process. In Section 4.1 we elaborate upon how GENR8 fits a niche within a more comprehensive design process.

1.4 Related work

In Table 1.1 we present an overview of some of the work that is related to GENR8. We focus on three fields, architecture, evolutionary design and plant modeling. We investigate how EC, environmental influence and developmental models have been used in these projects. The common trait for these projects is that they focus on creating designs, plants, creatures etc.

The table column headers are:

Research project The name of the project or the person(s) behind it.

Application The field that the work is done within.

Level of interactivity How much can the user interact with the tool? There are three levels: none, interactive evolutionary computation (IEC, described in Section 4.3) and interruption, intervention and resumption (IIR, described in Section 4.2).

Influence from the environment Does the tool create its artifacts within a simulated environment that affects the outcome and is it reactive to the environment?

EA Does the tool contain an evolutionary search component?

Model for development Is there a developmental model (eg L-systems) that governs the creation of new designs?

A short description of each of the related systems follows:

MoSS [31] [18] is the predecessor of GENR8. There is no evolutionary search, the environment is more restricted and it does not have any user interaction.

AgencyGP [19] is another project by the Emergent Design Group. The purpose of the project is to develop a new tool for designing office spaces for a non-hierarchical organization.

Broughton et al [5] use ordinary L-systems and genetic programming to evolve architectural designs. This work has a completely different approach than GENR8, the fitness evalution is done with a traditional optimization function and the structures are displayed with spherical building blocks.

Rosenman and Gero [22] evolve architectural floor plans.

Nishino et al [17] work with traditional IEC, that is, the fitness evaluation is entirely up to the user. This work involves a 3D modeler for creating rough sketches rapidly.

Research project	**Application**	**Level of Interactivity**	**Influence from the environemt**	**EA**	**Model for development**
GENR8	Architecture	IIR	Yes	Yes	Yes
MoSS	Architecture	None	Yes	No	Yes
AgencyGP	Architecture	IIR	No	Yes	No
Broughton et al	Architecture	None	Yes	Yes	Yes
Rosenman and Gero	Architecture	IEC	No	Yes	No
Nishino et al	Design	IEC	No	Yes	Yes
GADES	Design	None	Yes	Yes	No
Cambrian art	Art	IEC	No	Yes	No
Kókai et al	Image representation	None	No	Yes	Yes
Curry	Plant modeling	IEC	No	Yes	Yes
Jacob	Plant modeling	None	No	Yes	Yes
Green	Plant modeling	Unknown	Yes	Yes	Yes
L-parser	Plant modeling	IEC	Yes	Yes	Yes
Kovács	Plant modeling	None	Yes	No	Yes
Lantin and Fracchia	Cellular structures	None	Yes	No	Yes
Sims	Artificial creatures	None	Yes	Yes	Yes

Table 1.1: Categorization of the related work.

GADES [2] is a multi-purpose design tool. It can be used to design cars, heat sinks, coffee tables and hospitals.

Cambrian art [12] is a website where the user can create works of art through IEC.

Kókai et al [10] use EC to find an L-system that will generate a pattern that matches the blood vessels in the eye. This allows for efficient storage of the pattern.

Curry [6] uses a genetic algorithm to control the parameters of an L-system. The purpose of his research is to create a tool for plant modeling that can be used without knowledge of the underlying L-system grammars.

Jacob [9] models L-systems with the aid of *Mathematica*.

Green [15] is a tool for modeling natural elements using stochastic L-systems.

L-parser [14] is an L-system interpretor that is available online.

Kovács [11] uses an L-system model that focus on the effects of tropism and boundaries.

Lantin and Fracchia [13] try to model the cellular development of organisms.

Sims [28] uses evolutionary algorithms to create artificial creatures living in an artificial world.

1.5 Contributions

1.5.1 Computer Science

There are several issues when using evolutionary computation for design applications. Some of the things to keep in mind are:

- The control of the creative process, how is it mediated between the tool and the user.

- How the evolutionary process is understood by the human designer.
- How to allow genotype and phenotype modifications for a given representation.
- To make the EC component seamlessly integrated into the tool.
- Design evaluation.
- What is the output of the tool and how is it represented.

Moreover, we have explored and extended the map L-system growth model to make it work in three dimensions (3D). We have added some of the more advanced concepts from ordinary L-systems to the map L-system model. This thesis also shows that map L-systems can be useful outside their biological origins.

We have also applied the fairly new concept of grammatical evolution to a real problem and showed that it was a very useful and powerful abstraction. In doing this, we gave the map L-systems a Backus-Naur formulation (BNF). Furthermore, we identified several interesting sub-universes of this BNF that were particularly useful to GENR8.

1.5.2 Architecture

For a designer, the tool itself is a central part of the design process. Increasingly powerful computers give designers entirely new possibilities but in order to be useful, a designer has to be able to use it comfortably. Computation allows for more exploration than would be possible with just pen and paper and it can be viewed as a new paradigm in architecture.

Today there are several computer aided design (CAD) tools available. The problem with these tools is that they can be categorized as drawing aids (nevertheless powerful ones). They are not generative or creative in any sense and they do not provide any help on that part. The architect still has to come up with what to draw. Our goal is to develop a tool that is cooperative and stimulating to work with and that can help the designer to come up with new ideas.

Another novelty that has recently been introduced to the field of architecture are new materials. This has brought a radical change to how a building can be constructed. It is no longer necessary to have supporting structure as before, the walls can be supporting in themselves.

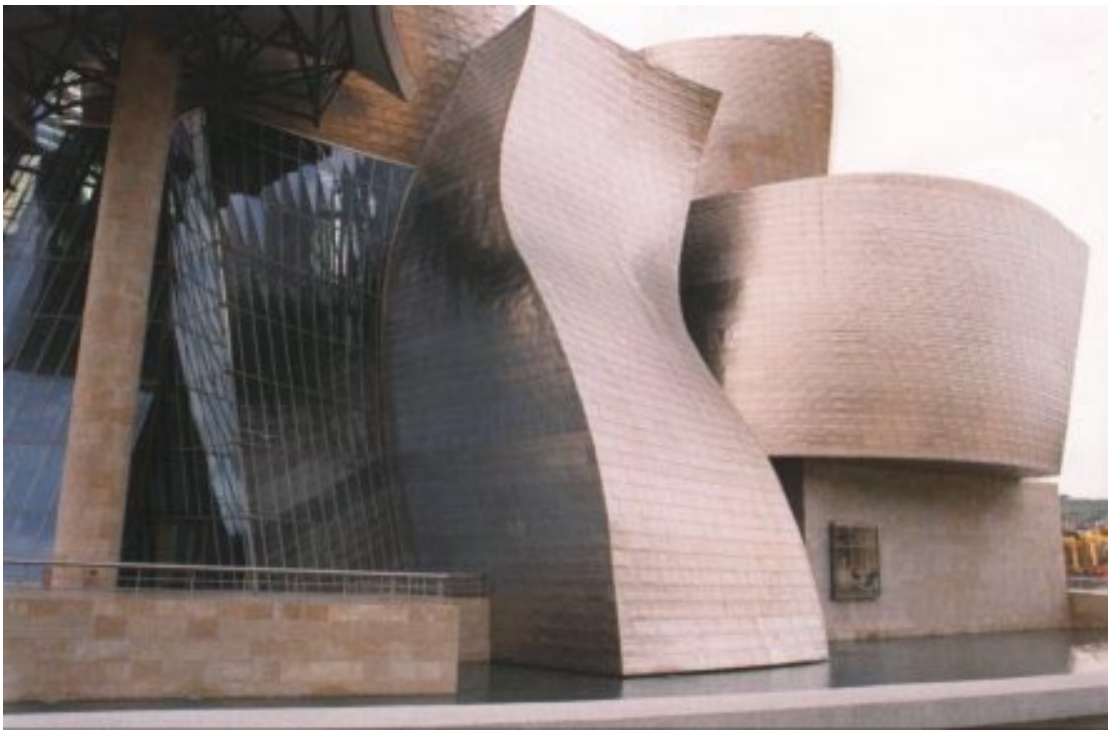

Figure 1-2: The Guggenheim Museum in Bilbao, a building where novel materials and techniques have been used (Frank O Gehry and partners, 1997).

A related issue are the new techniques for casting and construction. It is possible to combine structural elements in completely new ways. A famous example of this is the Guggenheim Museum in Bilbao. To explore these new concepts, architects need new tools.

This work is part of a larger process of developing and understanding design tools based on computation. The advent of powerful computers has brought new possibilities to the field of architecture. Emergence is a phenomenon that is particularly well suited to study with the aid of computers. It also harbors many interesting concepts that so far has not been explored for use within the field of architecture. Nevertheless it is a concept that architects can easily grasp and it is something that they use

in their work. However, the architects lack adequate software tools to exploit and harness the concept of emergence.

Chapter 2

Growth Model

GENR8 needs to grow surfaces in an organic fashion. We use map L-systems, an extension of the more familiar L-systems as a basis for its growth model. To search the universe of possible grammars, we use the grammatical evolution EA. The EA relies on a BNF representation of the map L-systems, defining a universe of grammars. Grammatical evolution employs a BNF to map a fixed length integer genome to an executable structure. GENR8 has two mapping processes, one that maps a genome to a grammar and another that interprets the grammar and constructs a surface (phenotype). We will start by describing the second step, the growth model. This may seem odd, but it is necessary in order to understand what we are trying to achieve in the first step.

2.1 L-systems

Lindenmayer systems, or L-systems for short, were introduced by biologist Aristid Lindenmayer in 1968 as a method to describe the development of plants. L-systems have also been used to generate a wide variety of fractals and space filling curves. A comprehensive description of L-systems can be found in [20].

However, L-systems have also interested computer scientists as a basis for formal language theory. From this perspective L-systems can be seen as parallel rewriting of strings (in contrast to the Chomsky grammars which are sequential). The simplest

L-system are deterministic and context-free (later we will relax both these conditions) and here follows a formal definition (according to [20]) of their operation.

L-systems 1 (Alphabet) *Let V denote an alphabet, V^* the set of all words over V, and V^+ the set of all nonempty words over V.*

L-systems 2 (Context free) *A string context-free L-system is an ordered triplet $G =< V, \omega, P >$ where V is the alphabet of the system, $\omega \in V^+$ is a nonempty word called the axiom and $P \subset V \times V^*$ is a finite set of productions.*

L-systems 3 (Productions) *If a pair (a, χ) is a production, we write $a \rightarrow \chi$. The letter a and the word χ are called the predecessor and the successor of this production, respectively. It is assumed that for any letter $a \in V$, there is at least one word $\chi \in V^*$ such that $a \rightarrow \chi$. If no production is explicitly specified for a given predecessor $a \in V$, we assume that the identity production $a \rightarrow a$ belongs to the set of productions P.*

L-systems 4 (Deterministic) *A context-free L-system is deterministic if and only if for each $a \in V$ there is exactly one $\chi \in V^*$ such that $a \rightarrow \chi$.*

L-systems 5 (Generation) *Let $\mu = a_1...a_m$ be an arbitrary word over V. We will say that the word $\nu = \chi_1...\chi_m \in V^*$ is generated by μ and write $\mu \Rightarrow \nu$ if and only if $a_i \rightarrow \chi_i$ for all $i = 1, ..., m$. A word ν is generated by G in a derivation of length n if there exists a developmental sequence of words $\mu_0, \mu_1, ..., \mu_n$ such that $\mu_0 = \omega, \mu_n = \nu$ and $\mu_0 \Rightarrow \mu_1 \Rightarrow ... \Rightarrow \mu_n$.*

Before we go on and explore how L-systems can be used to represent fractals and surfaces, let us illustrate the mechanics of the rewrite rules with a simple example. Our productions are:

$$a \rightarrow ab$$

$$b \rightarrow bc$$

$$c \rightarrow ba$$

Starting from a single letter a, the above productions generate the following sequence.

a

ab

abc

$abba$

$abbbab$

$abcbcbcabc$

$abbabbabbaabba$

...

2.2 Turtle rules

Now that we have a growth model, we must find a suitable graphical interpretation so that the strings can be viewed as surfaces. We also need to show that this model actually bears any resemblance to organic growth. To do this, all the information needed to generate the surface must be present in the string.

Drawing an L-system is often described as a set of turtle-rules [20]. One can think of it as a turtle moving around in 3D space and drawing lines. The state of the turtle is defined by its spatial coordinates (x, y, z) and its orientation. The orientation is represented by three vectors, **H**, **L**, **U** indicating the turtles heading, direction to the left and up, respectively. These vectors are orthogonal to each other and normalized. A change of direction can be described by a 3×3 rotation matrix **R**.

$$[\mathbf{H}' \quad \mathbf{L}' \quad \mathbf{U}'] = [\mathbf{H} \quad \mathbf{L} \quad \mathbf{U}]\mathbf{R}$$

For ordinary L-systems we need to define to parameters for the turtle, the length of each step and the turn angle. For map L-systems, we no longer need the first one, except for the starting seed.

2.3 Brackets

With the above rules, we are only able to draw a single line. All the characters in the interpreted string represent line segments that must be connected to each other.

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

Table 2.1: Turtle commands and their meaning.

Therefore we introduce two new symbols, '[' and ']'. When then '[' is encountered, the current state of the turtle is saved and pushed on to a stack. As the ']' is encountered (they must come in pairs), a state is popped from the stack to replace the current state of the turtle. This allows us to draw branches from the main line. All of the turtle commands used by HEMLS can be found in Table 2.1.

2.4 Map L-systems

Ordinary L-systems interpreted by a set of turtle-rules have successfully been used to produce realistic images of plants and models of how real plants develop (an example of this can be found in Figure 2-1) [11] [15] [9] [6]. However, the very nature of the model gives us an arboreal structure, which is clearly not suitable for surfaces.

Map L-systems were originally developed as a model for cellular development. Formally it can be seen as a method for rewriting planar graphs with cycles. They work in a similar fashion to ordinary L-systems, but during each growth step we now have two phases. During the first phase segments are rewritten as before. In the second phase pairs of matching branches are connected to form new segments. An

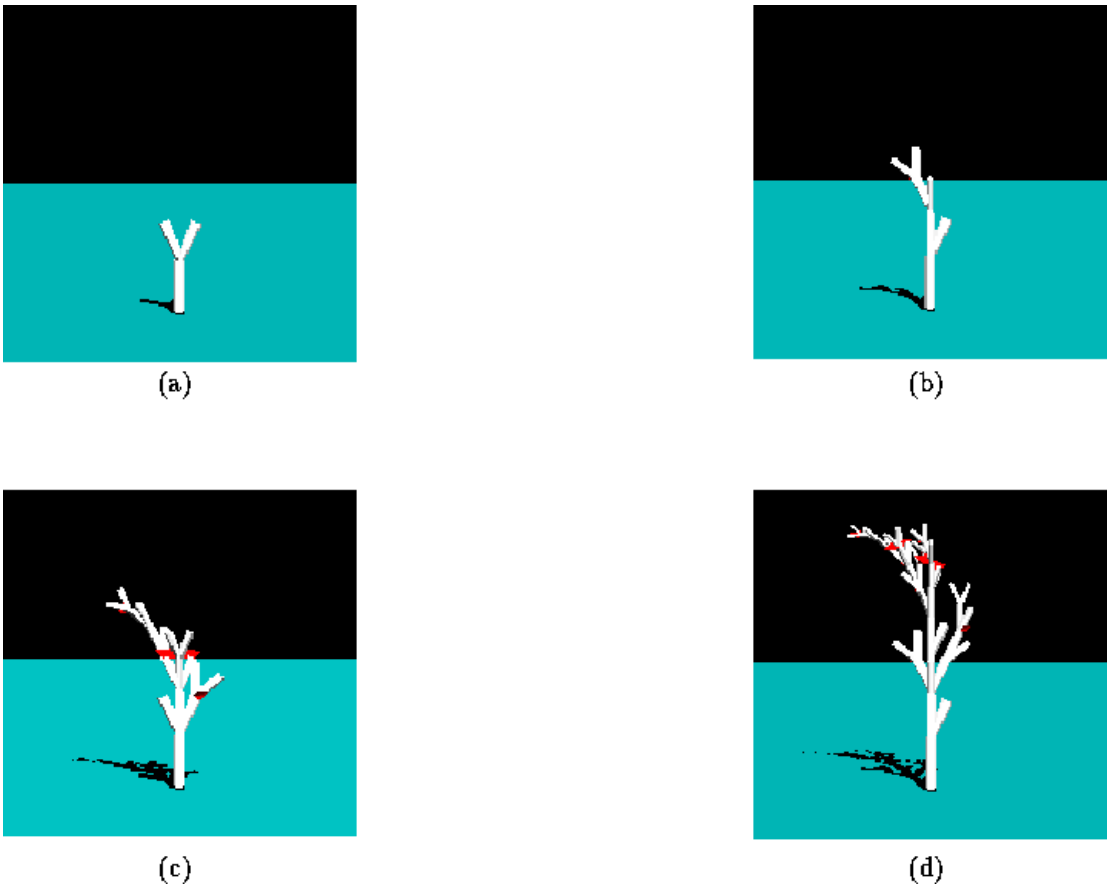

Figure 2-1: An example of an ordinary L-system. The string is interpreted using turtle rules to produce a realistic image of a plant.

example of a simple map L-system can be found in Figure 2-2.

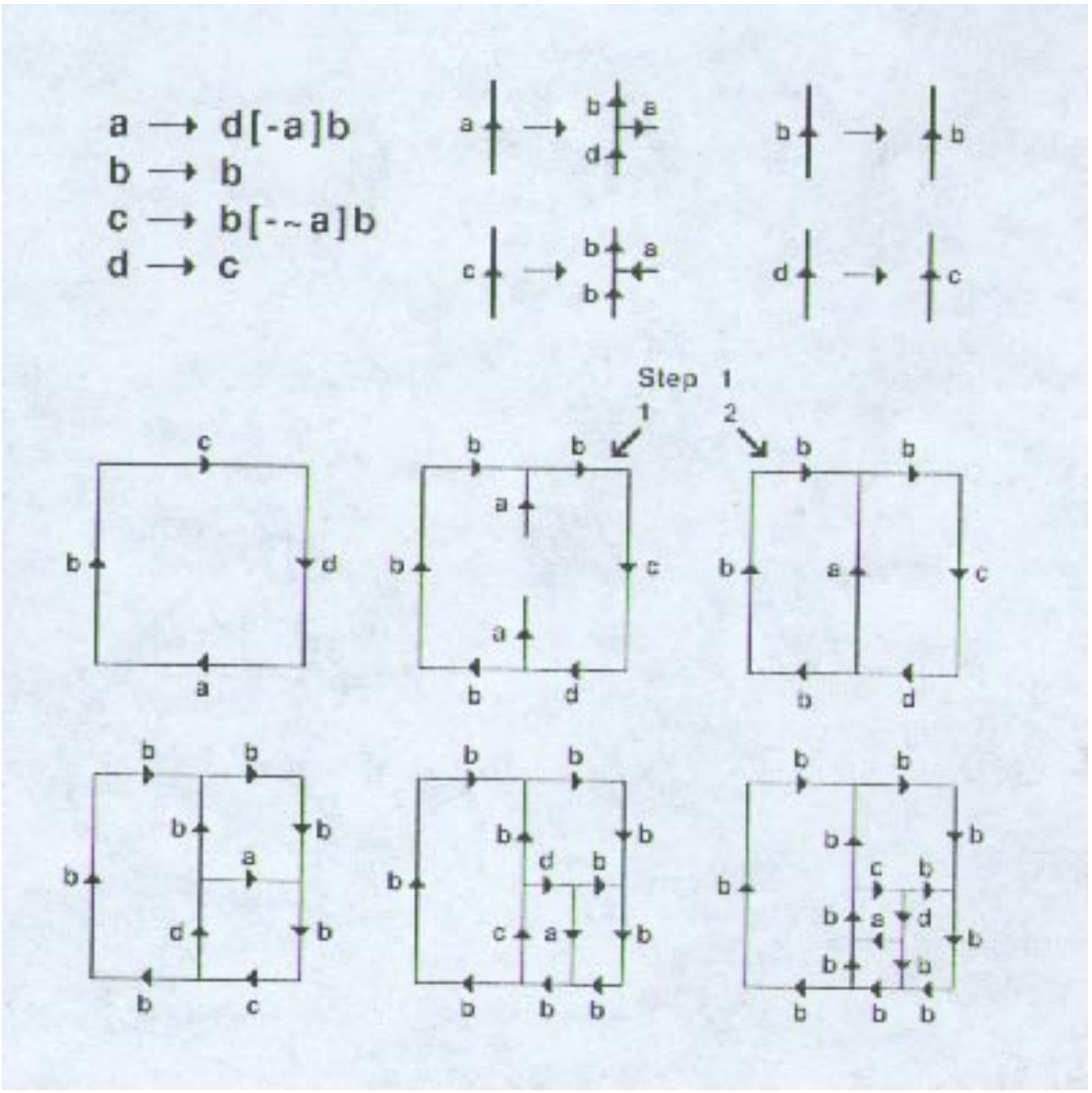

Figure 2-2: A simple map L-system. The productions are shown in the top row. The successive application of these productions to the seed, leftmost in the middle row, are shown in the two bottom rows.

2.5 Hemberg Extended Map L-systems

The map L-systems are defined in two dimensions and since we want to handle surfaces in 3D space it was necessary to extend the model. We have also added some of the variations that exist for ordinary L-systems to our model. We call our model Hemberg extended map L-systems (HEMLS) and our additions will be described in

the following sections.

An interesting property for HEMLS (and all L-systems for that part) is that growth is a local phenomenon. All productions are applied locally (synchronously or asynchronously, as we shall see in the next section) and we only need a very limited knowledge of the global state. This makes our growth process emergent, according to the definition in Chapter 1.

2.5.1 Branches

In the basic model we only have to consider what region the branch is pointing into. However, when we extend to three dimensions we must keep track of the spatial orientation of the branch instead. For ordinary map L-systems, the following criteria must be fulfilled if two branches are to connect:

- They enter the same region.
- They have the same type.
- One branch is oriented away from the main axis, while the other is oriented towards the main axis.

For HEMLS, the last criterion has been replaced by:

- The dot product of their orientations is less than a given tolerance.

The tolerance is a parameter (`BranchAngle`, it can be set by the user), that controls how easily branches are merged. If the tolerance is low, the branches need to be better aligned in order to merge.

In the HEMLS model there are two ways of connecting the branches, synchronously or asynchronously. The synchronous method is the same that is used by ordinary map L-systems, first all the productions are applied and then the branches are connected. In the asynchronous mode, we try to connect branches after each production has been applied. The choice between the synchronous and asynchronous mode can have great impact on how a surface will be subdivided. An example of this can be found in Figure 2-3 and 2-5.

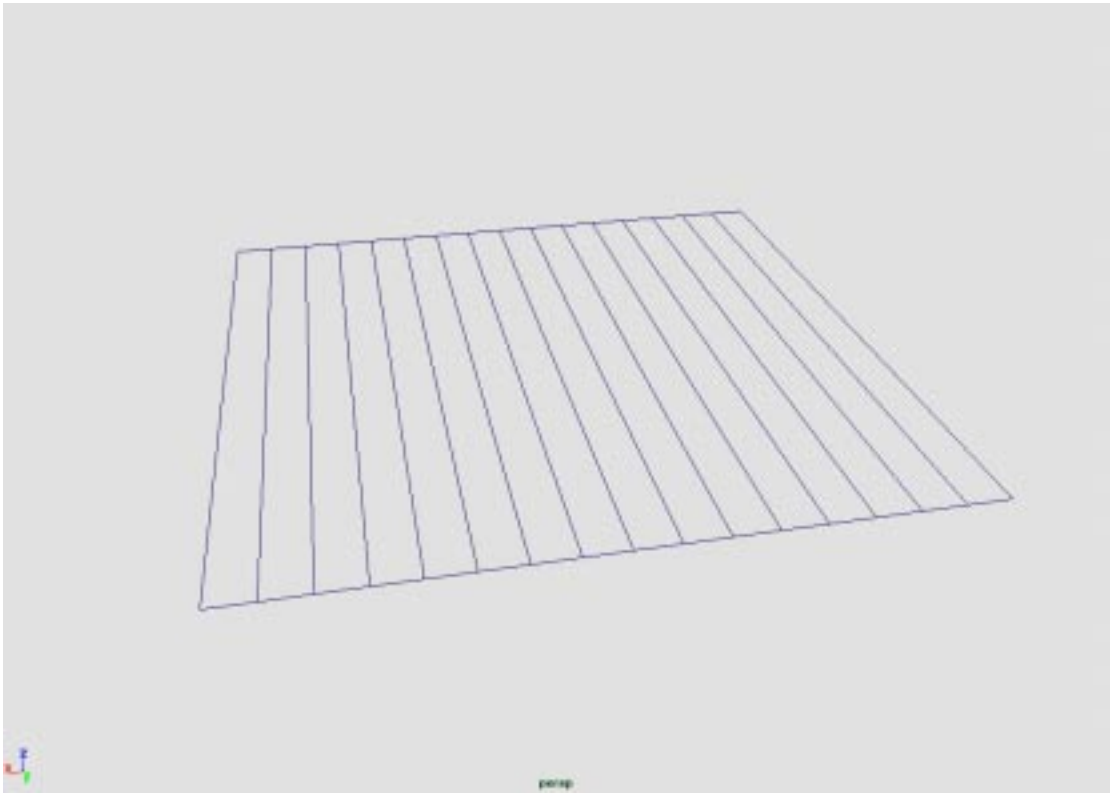

Figure 2-3: This is the same grammar and environment as in Figure 2-5, only the branch merging mode has been changed from asynchronous to synchronous.

2.5.2 Context sensitivity

We wish to have some sort of interaction between the elements of the string, this will affect the resulting surface and the growth will be more non-linear and unpredictable. One way to achieve this is to introduce context sensitivity. That means that we only apply a production to a letter if it has the right context, ie it is surrounded by some specified letters. This allows more complex productions and more expressive grammars where different parts of the string behave in differing ways. The context sensitivity can be used to model information exchange between neighboring segments.

Formally, context sensitive L-systems are on the form *(k, l)L-systems* where k is the number of letters to the left and l is the number of letters to the right. For a (1, 1)L-system we have $b < a > c \rightarrow \chi$, the letter a can produce the the word χ if and only if it is preceded by the letter b and followed by the letter c. The context sensitivity used in GENR8 is presented in Table 2.2.

Symbols	**Context**
a	Context-free.
$b < a$	Left-side context sensitive.
$a > b$	Right-side context sensitive.
$b < a > c$	Left and right side context sensitive.

Table 2.2: Context sensitivity for the L-systems in GENR8.

Since we represent the strings as directed graphs, each segment can be preceded and followed by several segments. Thus it is enough if one of the predecessors or successors satisfy the context constraint. In Table 2.3 line (1) is a context sensitive production.

2.5.3 Time variation

By introducing a counter (written as an index) to each segment, we can have different behaviors depending on the value of the counter. Phases, delay mechanisms and temporal variations can now be incorporated into the growth model. The growth process may be altered as it progresses so that we use different productions for each time step. Lines (2) and (3) in Table 2.3 are examples of time varying productions.

2.5.4 Stochastic rules

The deterministic L-systems that we have seen so far will always produce the same results. One way to introduce variation is to have stochastic productions. We do this by assigning a probability to each production. We have probability $\pi(p)$ for applying each production to a given predecessor. In Table 2.3, lines (4) and (5) are examples of stochastic productions.

Seed:	$A + B + C + D$			
(1)	$A < B$	$\rightarrow$	$A\,[\,[\,+ + D\,]\,- - D\,]\,B$	
(2)	C_i	$\rightarrow$	$C_{i+1}\,[\,[\,+ + D\,]\,- - D\,]\,C_{i+1}$	If $i < 2$
(3)	C_i	$\rightarrow$	C_{i+1}	If $i > 1$
(4)	D	$\rightarrow$	$+D - -D$	$p = 0.5$
(5)	D	$\rightarrow$	$-D + +D$	$p = 0.5$
	Angle		45	

Table 2.3: A simple stochastic time-varying context-sensitive map L-system.

2.6 Implementation of GENR8

GENR8 is implemented through the Maya application programmer's interface (API). The code is written in C++, the language used by the API. Some of the communication with Maya is done via the Maya embedded scripting language (MEL). GENR8 has its own internal model of the surface, it does not rely on the Maya representation, and we try to do as much as possible of the computation in that model. One reason for this is speed, the computations are faster if we can limit the interaction with Maya. Another reason is control; in our own model, we have full control of what is going on. If we ask Maya to calculate something for us, we can not be sure of exactly what is being done or how it is being done. Also, it is beneficial to have the growth model as independent as possible, facilitate a switch to another host software should that be necessary in the future. The general strategy is to use Maya as a user interface, it has great capabilities for setting up the scene and inspecting the output. We use C++ where it comes to best use, fast calculations.

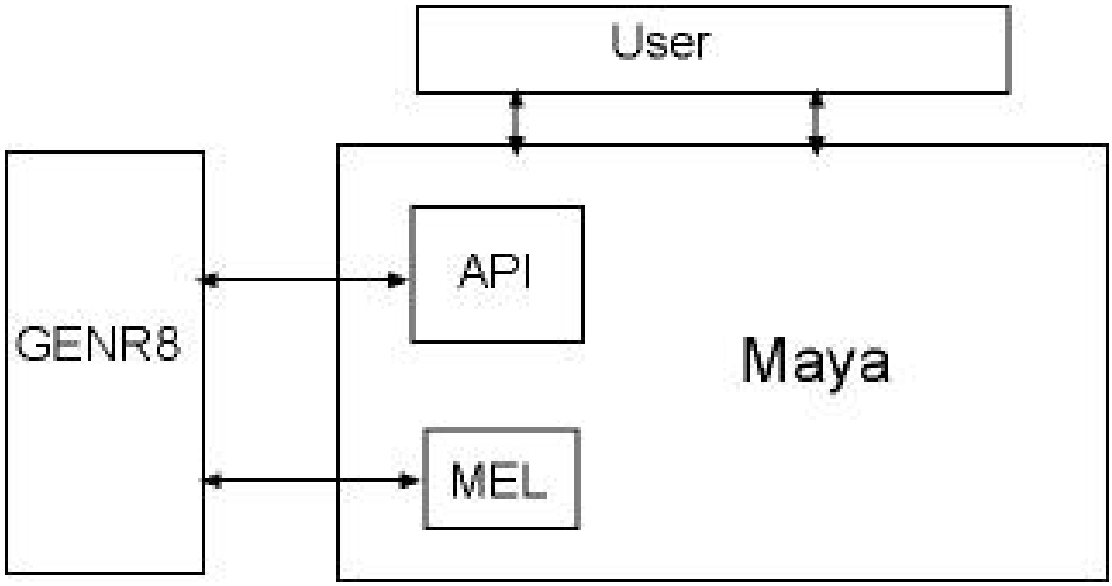

Figure 2-4: A schematic view of GENR8 and how it is incorporated into Maya.

2.6.1 Internal representation

The HEMLS model is represented as a graph data structure. Since the graph is sparse, it is stored as a linked list. In GENR8, it is necessary to keep track of information regarding the vertices, the edges and the regions (the areas enclosed by the edges), in order to be able to perform all our operations. This means basically that there is more book-keeping, it does not increase the complexity of the algorithms.

It is the vertices of the graph that are moved by the growth and the environment. On the other hand it is the edges that are affected by the productions and drawn in the Maya scene. Since the graph is in 3D, it is essential to keep track of the regions, the areas enclosed by edges, in order to connect branches.

What is interesting about this representation is that we only have topological knowledge about the graph, we do not have explicit knowledge about the spatial ordering of the nodes. The main advantage of this data structure is that it is fast and easy to add new nodes (complexity *O(1)*) and traverse the graph (complexity *O(n)*). The main drawback is that we have very little information that is globally structured. Any information that can not be obtained locally, eg if two non-neighboring nodes are close to each other, is expensive to obtain.

2.6.2 Maya representation

When GENR8 is started it takes elements from the Maya scene (boundaries, user-defined seeds, attractors and repellors) and extracts the relevant information from the Maya objects. The only use of Maya during the computation is to find intersections with the boundaries (Maya has a powerful algorithm that allows us to find the intersection of a ray and an arbitrary surface). Finally, we use the API to draw the resulting surface in the scene before handing over control to the user. At all points (except when the CPU is busy) the user has full access to all features in Maya.

HEMLS are general in the sense that they can generate arbitrary surface. Unfortunately, Maya is not as versatile, and it is hard to construct a nurbs-surface that is not four-sided, when working from the API. There are two solutions to this problem. One is to use Maya's built in meshing-algorithm. The main limitation of this approach is that the surface is polygonal. The other solution is to use a scaffold model to visualize the surface. This means that we draw ordinary lines instead. It is trivial to interpret the scaffold as a surface, they have been used in all the figures in this thesis. The scaffold model is not very useful if one wants to continue working with the surface in Maya, or export it to another program. But, since the lines are B-splines, they are smoother than the polygonal surface mesh.

2.7 Environment

One of our goals was to have a reactive growth model, one that dynamically interacts with the environment. In the L-system vocabulary, this is called tropism and it can be understood as the influence of external forces on the growth.

GENR8 has a very powerful environment that has great impact on the development of a surface. In the general case it is impossible to predict what a certain grammar will result in, unless you know what the environment looks like.

Figure 2-6 shows the growth of a surface in an environment with five repellors. The grammar is the same as in Figure 2-5, but the repellors affect the growth.

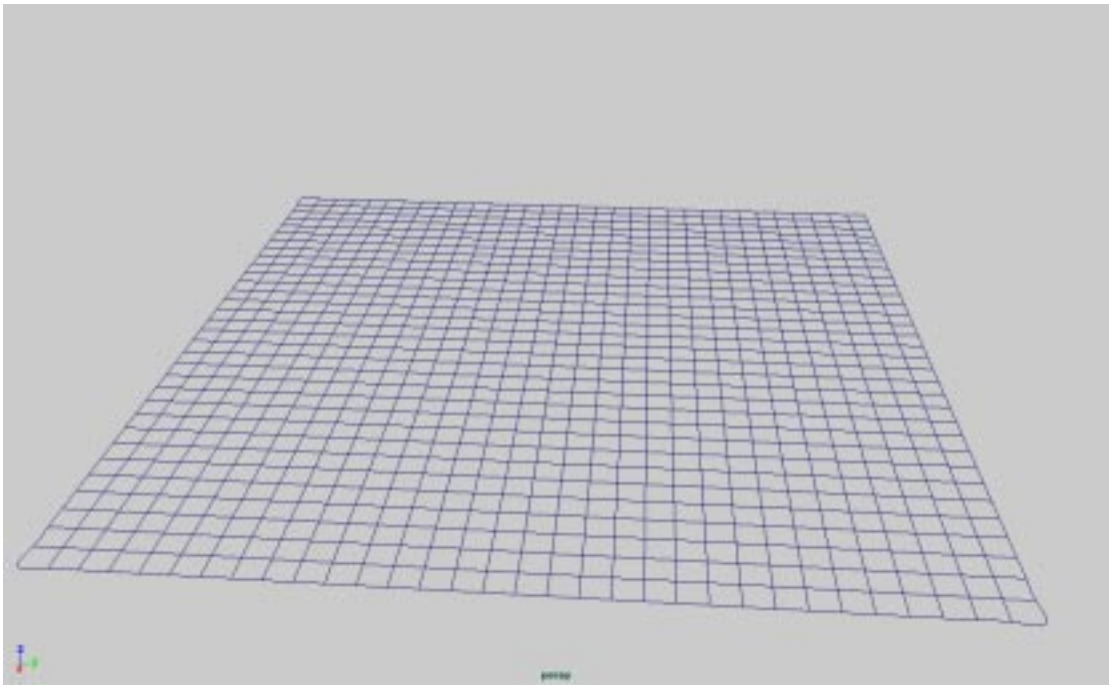

Figure 2-5: A surface grown in an empty environment

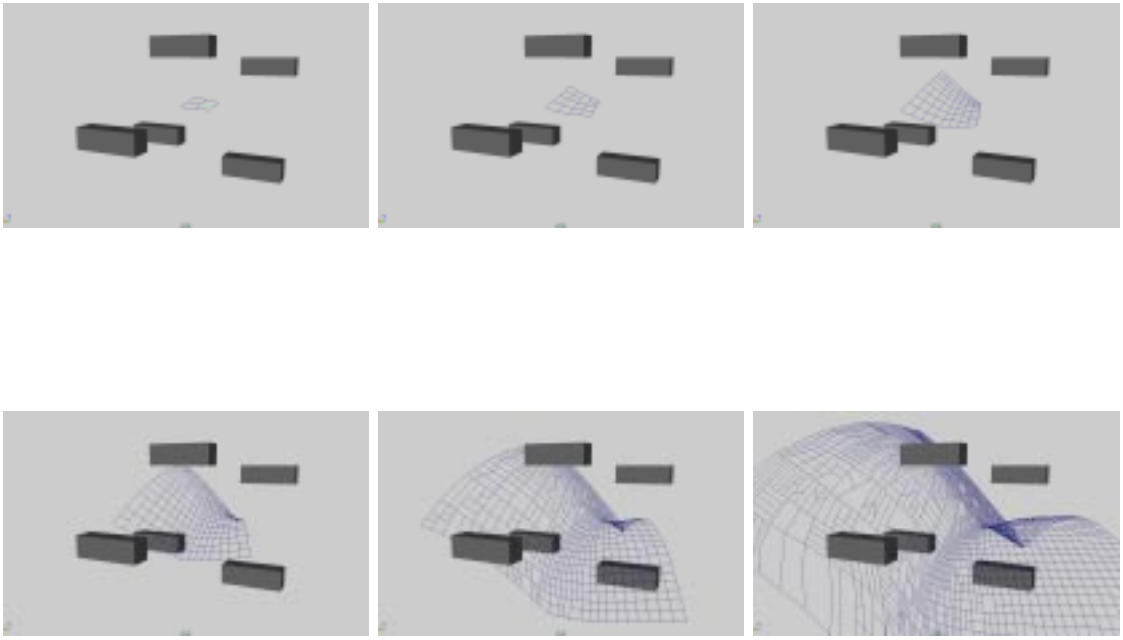

Figure 2-6: The growth of a surface in an environment with five repellors.

2.7.1 External forces

The GENR8 environment models forces. There are three types of forces; attractors, repellors and gravity. The forces have a Newtonian character and are intuitive to use.

Attractors and repellors

The user may place attractors and repellors in the scene. Attractors and repellors work like magnets, and can be used to control the direction of the growth. They are situated in space and their effect depends on the relative location of the surface.

An attractor (or repellor) has two parameters, a *constant, c* (a non-negative real number), and *exponent, e* (a non-negative integer). Together with the *distance, d*, and the *mass, m*, of a surface element they give the magnitude of the *displacement, f* on that part of the surface through the equation $f = c \cdot d^{-e}/m$. The direction of the displacement is in the direction of growth for an attractor and the opposite for a repellor.

The singularity in $d = 0$ is obviously a source of trouble. If a vertex gets too close, it will move unpredictably and in very large steps. To deal with this problem, we have implemented another attractor function, that does not have this singularity. It can be written as;

$$d = \begin{cases} \frac{c}{md^e} & d > 1 \\ c \cdot d & d \leq 1 \end{cases}$$

Gravity

Gravity is a uniform force that has the same effect on all vertices, regardless of its position (unlike attractors and repellors). The effect of gravity can be seen in Figure 2-8, where we once again have the same grammar as in Figure 2-5.

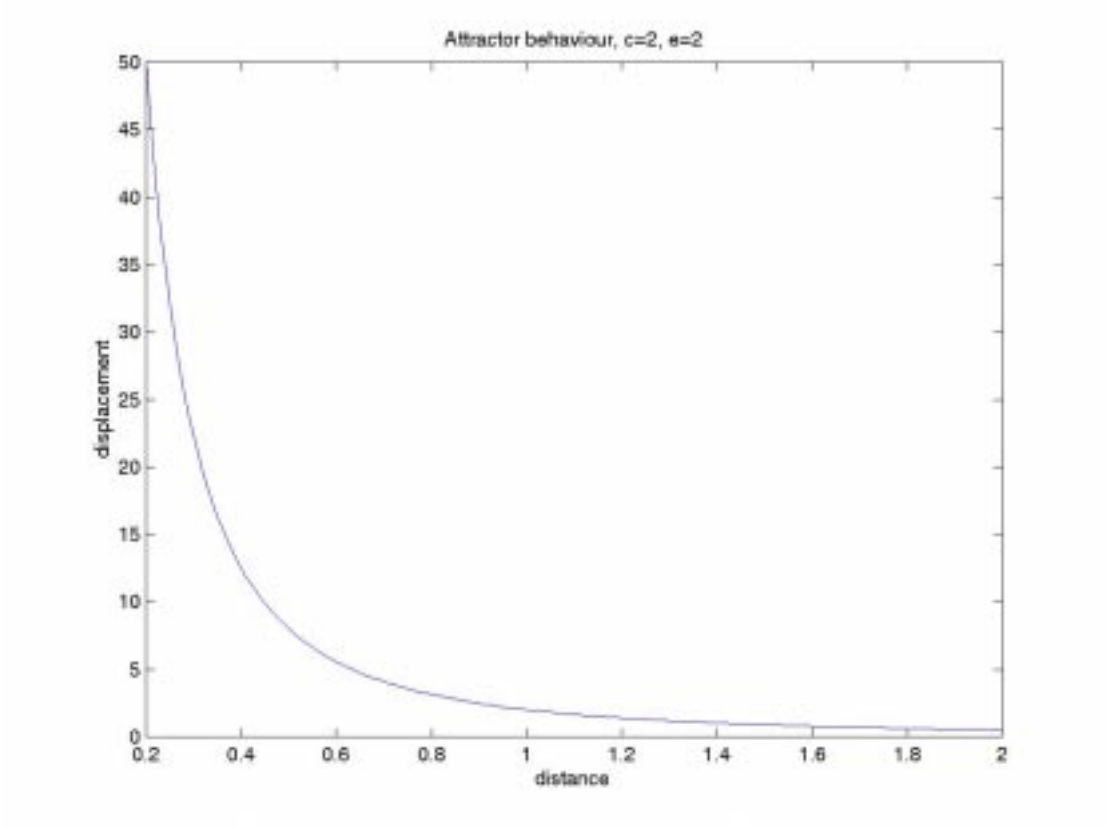

Figure 2-7: The displacement as a function of distance for an attractor/repellor.

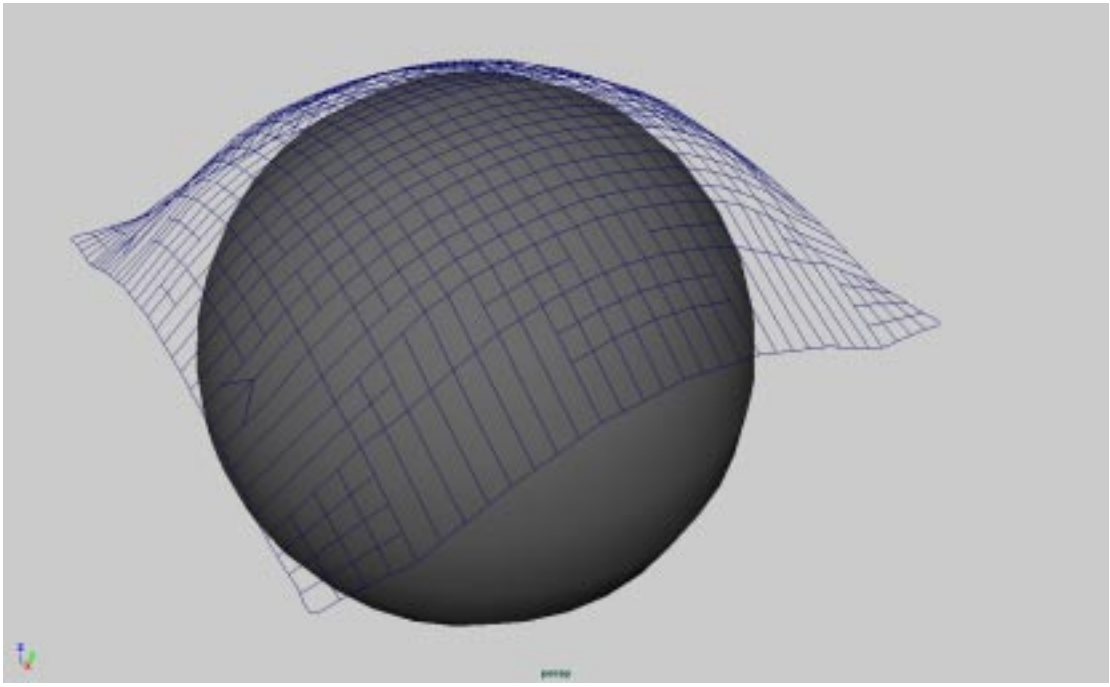

Figure 2-8: This surface is pulled downwards by gravity, but it is blocked by the sphere.

2.7.2 Internal forces

The heading of this section is somewhat misleading, although some of the parameters to be described can be considered Newtonian forces. The parameters control the growth in such a way that some vertices are forced to behave in a different way than in an empty environment.

Repelling points

This is actually a true force, in the same sense as the external forces. Each vertex gets a small repelling force, affecting all the other vertices in the graph. This will force each vertex to move to push away all the other vertices. This adds interaction on the phenotype-level and not only in the grammar.

Uniform cells

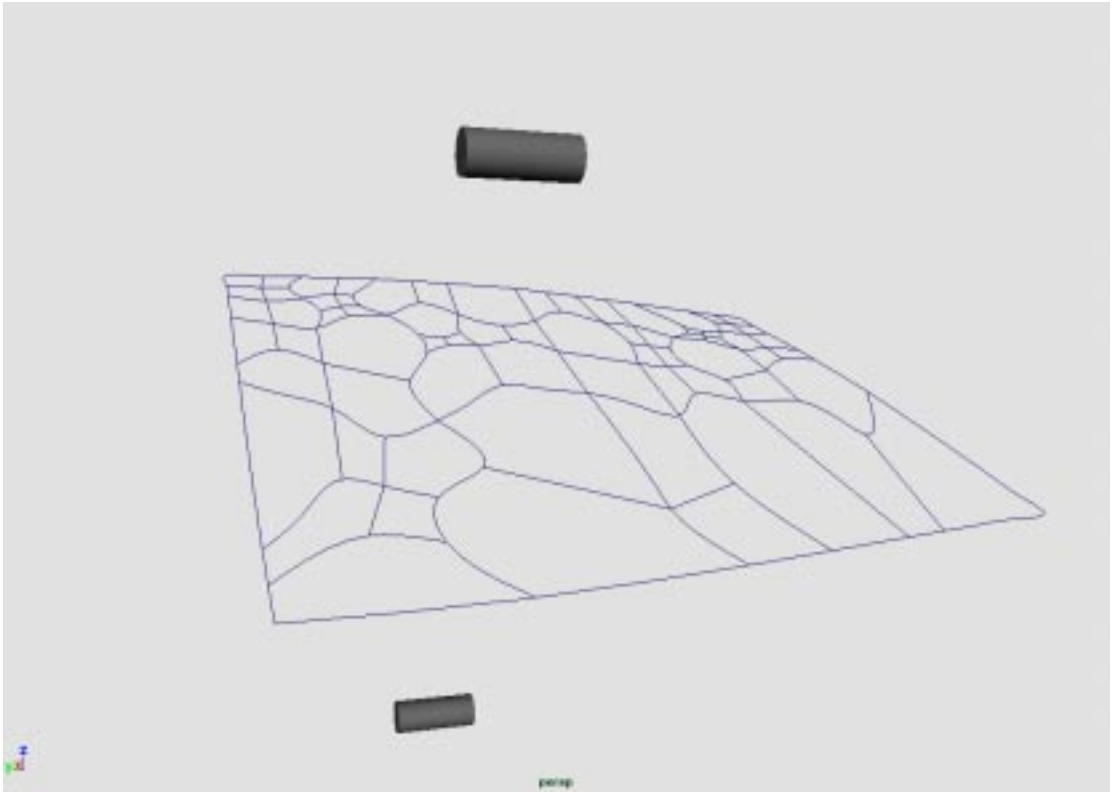

Figure 2-9: A surface where the cells have been made uniform, this is to be compared with the other figures in this chapter, where the cells tend to be subdivided by straight lines.

This parameter is similar to the repelling points, but another algorithm is used to reorder the vertices. After each growth step, all the vertices on the perimeter are fixed. The position of the remaining vertices are adjusted so that they try to maximize the distance to their neighbours. As the name implies, the subdivisions get a more uniform size and a more organic appearance; this can be seen in Figure 2-9.

Fixed perimeter

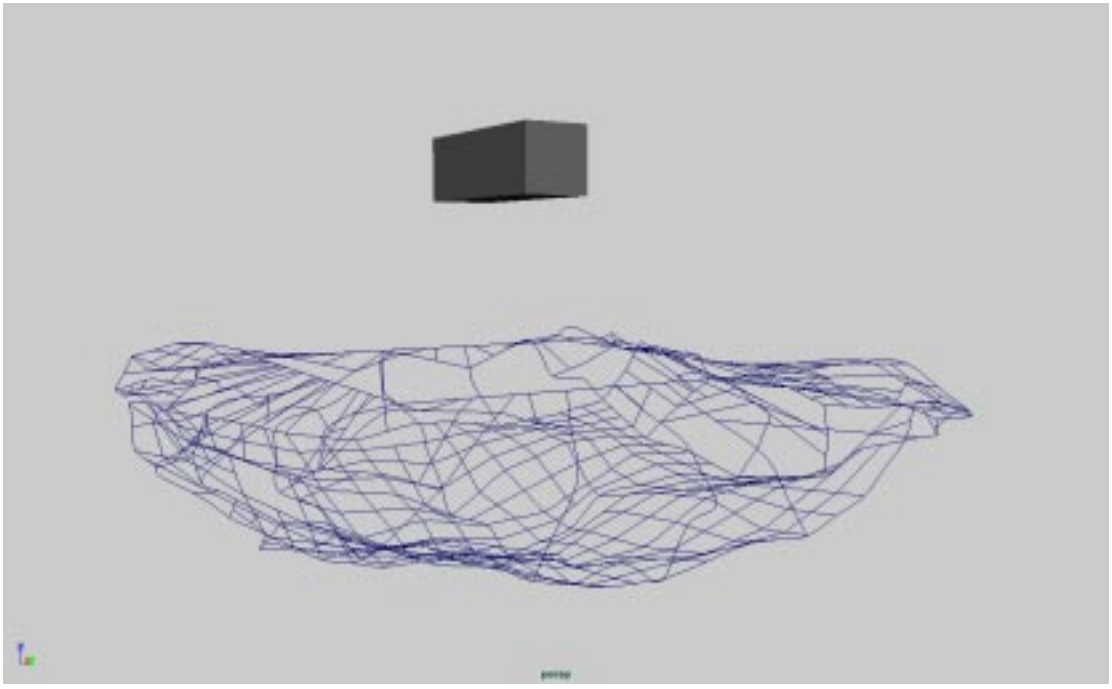

Figure 2-10: A surface with fixed perimeter in an environment with a repellor. The internal vertices are free to move and they are pushed away by the repellor.

We can choose to fix the perimeter, giving us a membrane like behaviour. This analogy should not be pushed too far though, since the other vertices tend to move away from the center as the surface grows. An example of this can be seen in Figure 2-10, where we have bag-shaped design.

Fixed center

We can also choose to fix the center. This means that the center of the surface always stays in the same position. In Figure 2-11, we can see how the surface is stretched

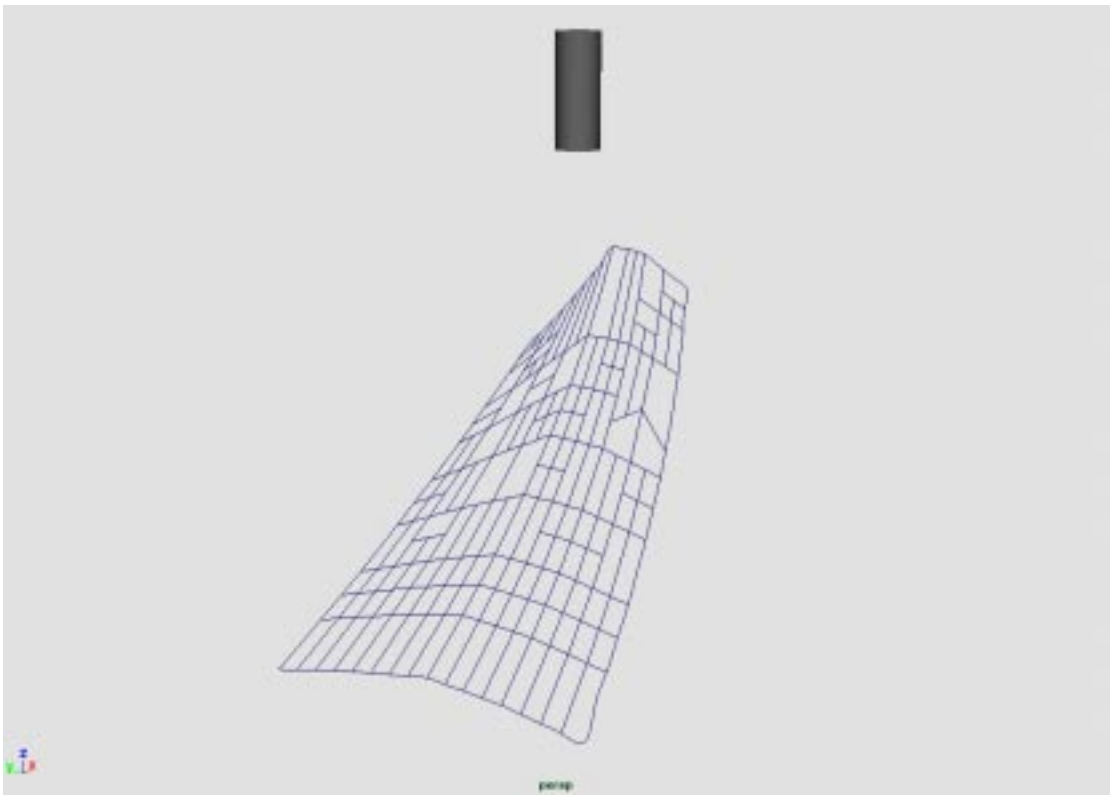

Figure 2-11: A surface with fixed center in the vicinity of an attractor.

out towards the attractor, rather than moved towards it.

Random noise

Finally we can also add noise to the environment so that the growth is no longer deterministic. There are two types of noise, one that affects the position of the vertices and one that affects the segment type. The latter kind of noise changes the symbols in the string being interpreted. In Figure 2-12 we once again have the same grammar and empty environment as in Figure 2-5. This time however, we have a random noise component which gives us a somewhat rugged surface. At some places the vertices have been perturbed so much by the noise that the branches were unable to connect, leaving larger cells.

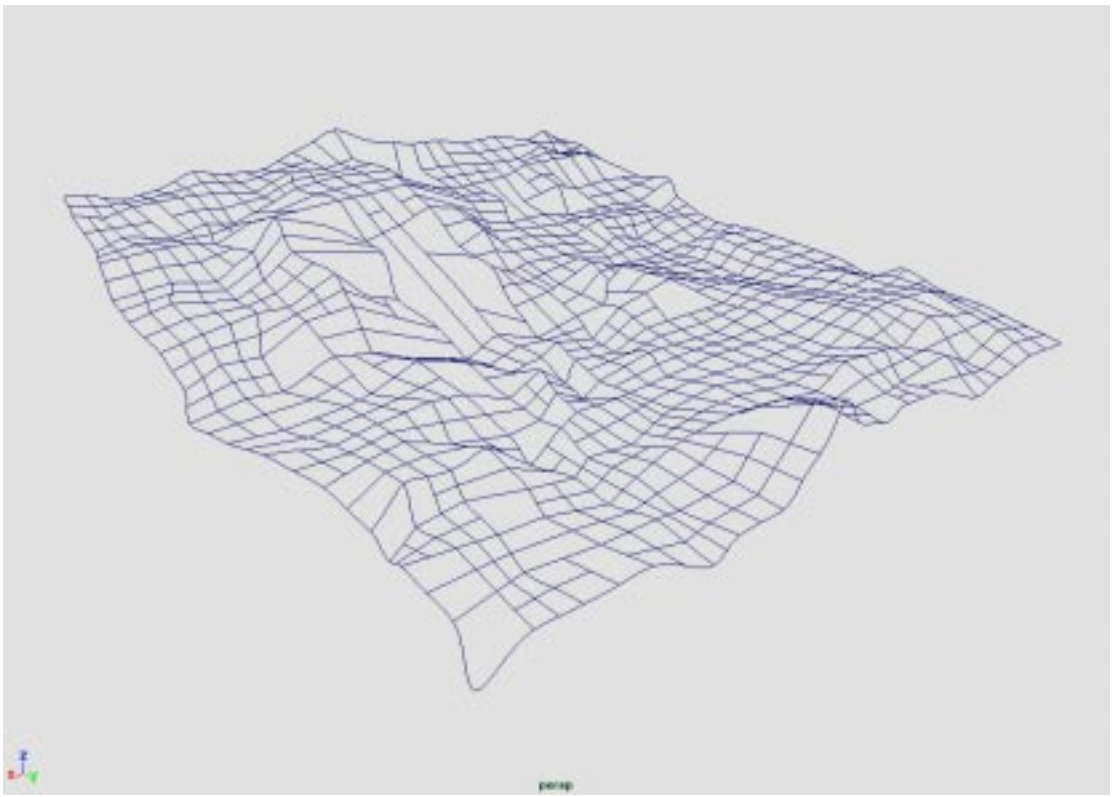

Figure 2-12: A surface where the random noise introduces some ruggedness.

2.7.3 Boundaries

The user may draw surfaces and volumes that act as boundaries. Convex boundaries are guaranteed to work, though concave boundaries will work in many cases. The walls can affect the growing surface in three different ways yielding different results.

Default

Using the default wall behaviour, the surface will 'slide' along the boundary as it collides and continues to grow. The default wall behaviour has been used in Figure 2-8 and 2-13.

Cut-off

If we instead use the cut-off behaviour, the parts of a surface that hits the boundary stops dead and it will not continue to grow. In Figure 2-14 we have the same parameters and values as in Figure 2-13, except for the wall behaviour.

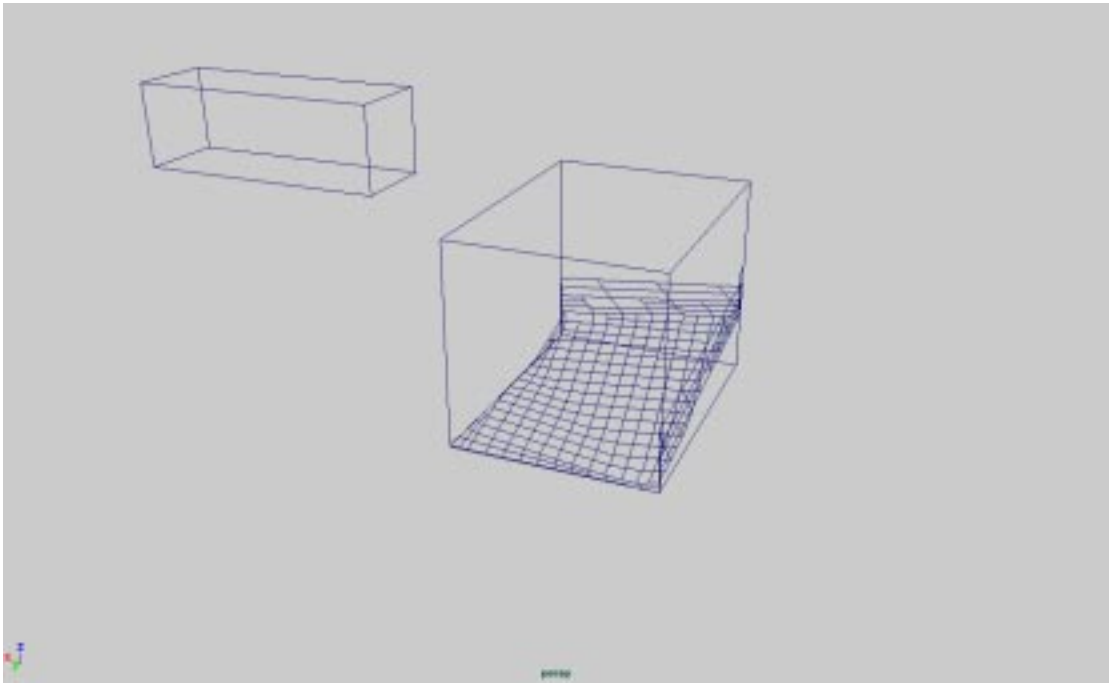

Figure 2-13: Surface inside a bounding box. The repellor in the upper left corner pushes the surface into the corner of the bounding box.

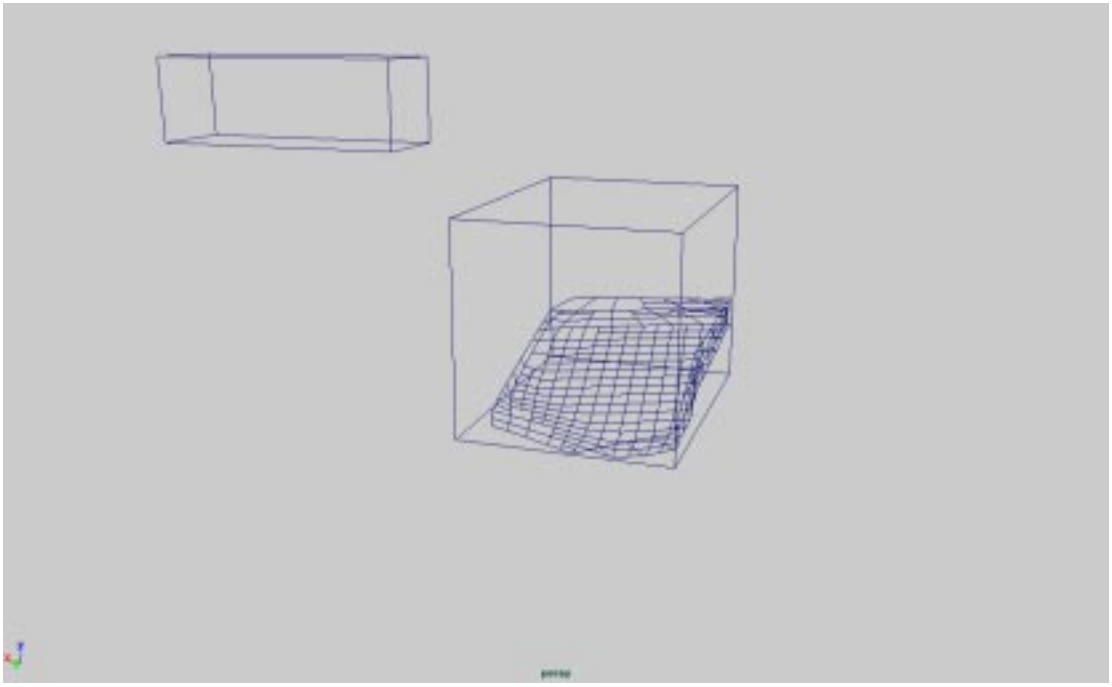

Figure 2-14: Surface inside a bounding box. The repellor in the upper left corner pushes the surface into the corner of the bounding box, but once the perimeter of the surface hits the walls, they stop moving (compare with Figure 2-13).

2.7.4 Parameters

The remaining parameters are relatively simple and straightforward to understand and they will only be described briefly.

Steps The number of growth steps. Since the growth is exponential, the surface grows very fast. None of the surface depicted in this chapter have used more than six growth steps.

Scale This parameter controls how much the surface will grow in each step. It governs how much the vertices will move during each step and thus the spatial extent of the surface. It is possible to have differing values for the scale in the x, y and z directions.

Start length The length of the sides of the genotypically encoded starting seed.

Start position The coordinates for the center of the starting seed.

Random seed Set the seed for the random number generator. This does not affect the growth, but it is useful if you want to be able to repeat a run.

Mass The only way to set the mass for a vertex is if you define your own starting seed. Otherwise, all vertices have the same mass.

Angle This parameter controls how much the turtle will turn. It must be encoded in the grammar.

Chapter 3

Evolution

Even though the recursive growth and the complex interactions of HEMLS are deterministic, the emergent properties of the growth make it very hard to predict the outcome. To predict what kind of surface a HEMLS grammar will produce is indeed a very hard task. Constructing grammars by hand is tedious and difficult[1] and we can not expect a person without considerable expertise to accomplish this. In order to make HEMLS useful for designers who are interested in creating surfaces, we use an EA to generate and evaluate grammars.

3.1 Evolutionary Algorithms

Evolutionary Algorithms is an umbrella term for a set of algorithms that use Darwinian evolution as an inspiration. They share the conceptual basis of a population of candidate solutions to a problem. The individuals of this population are tested for fitness, ie they are evaluated to find out how well they solve the problem at hand. This is all done in a generational loop and the individuals undergo recombination and mutation during the process.

[1]To exemplify this, it can be said that the author of this thesis is able to construct L-systems that do what he desires, provided that the task is not too complex and that he has a lot of time. His advisor, who has a PhD in computer science, has a hard time understanding the rules as he explains them to her. However, there remains some doubt if this is due to the difficulty of the grammars, the pedagogic abilities of the author or the intellectual capabilities of the advisor.

EAs provide a massively parallel search and they do so based on a phenotypical selection mechanism. Each member of the population has a genotype that is mapped to a phenotype. The inheritance is done with a blind mechanism operating on the genotype. This ensures that on the average, the population will improve in the next generation.

3.1.1 Grammatical Evolution

GENR8 employs an EA invented by O'Neill and Ryan called grammatical evolution [24] [23] [25], based on standard genetic algorithms (GAs). It uses standard genetic operations on a fixed length vector of bits. These integers are then used to generate an executable structure from a BNF-specification of the language. This introduces an additional mapping that does not exist in traditional GAs.

With this extra step, there is a clear distinction between the genetic operations and the language used to solve the problem at hand. This modularity makes it easier to use; the two issues of representation and recombination are completely separated and there is no need to constrain the genetic operations (as in genetic programming) to make sure that the expression is valid. At the same time, the language is more expressive (we can use any language with a BNF) than the parameterized values as is most common in GA.

Ryan and O'Neill [24], argue that GE is more similar to 'real' evolution, since it mimics more of the biological mapping steps. 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 translate these integers into a grammar.

In GENR8 the terminals of the 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 GENR8.

Grammatical evolution provides genetic degeneracy. That is, there are multiple

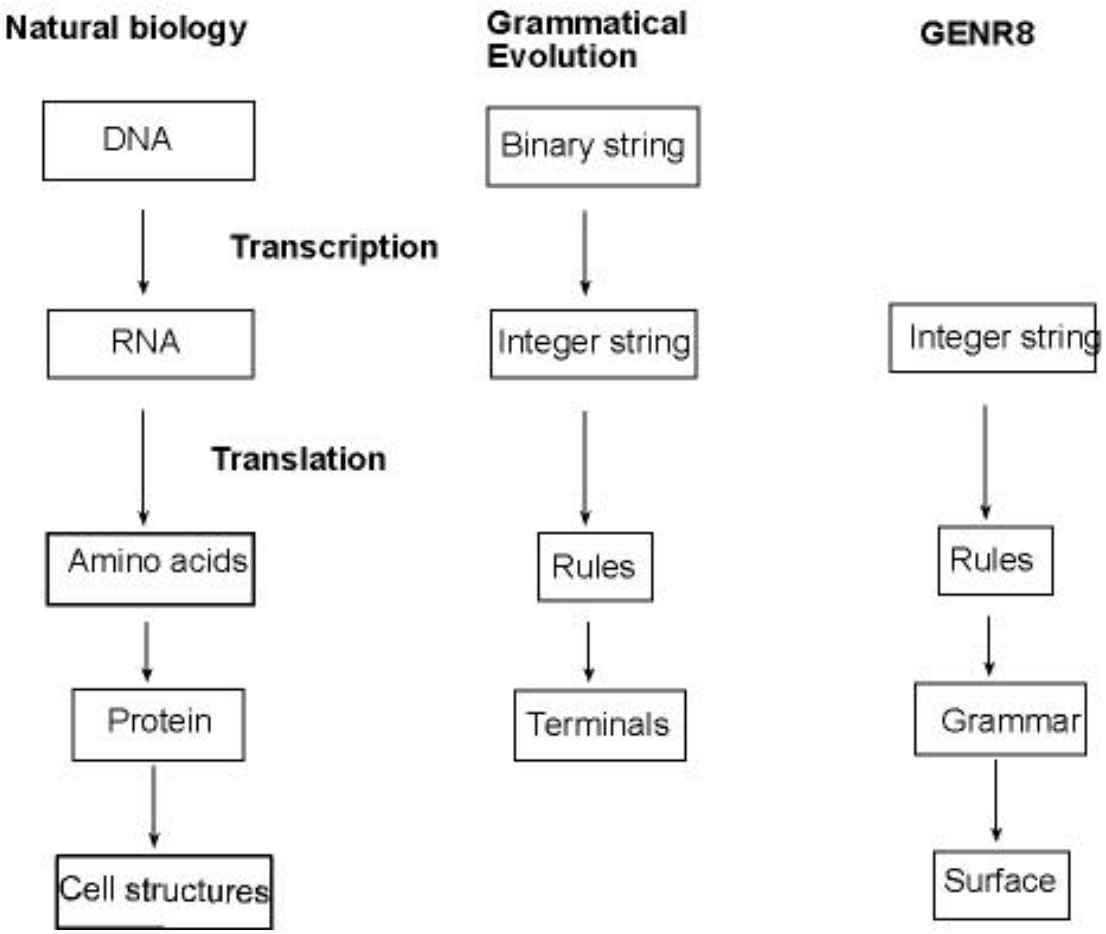

Figure 3-1: A comparison between biology, grammatical evolution and GENR8.

gene encodings that map to one decoding. For example, if there are three outcomes of a production rule, the gene values 42 and 897 will result in the same mapping, since $42 \text{ Mod } 3 = 897 \text{ Mod } 3 = 0$. With the first step of its two step genetic mapping process (BNF to HEMLS to surface), grammatical evolution allows GENR8 to provide users with different universes of HEMLS. For example, there is a subset of BNF that produces symmetric surfaces.

3.2 BNF-grammar

For GENR8 we have adopted a syntax that varies slightly from the one used in Chapter 2 and [20]. The motive for this is mainly to have something that is easier to parse.

BNF is a formal method for specifying grammars that is widely used in computer science [16] [4]. A grammar is represented by a tuple {`N, T, S, P`}, where `N` is a set of non-terminals, `T` is a set of terminals, `S` is a start-symbol (it has to a member of `N`) and `P` is a set of production rules that maps the elements of `N` to `T`. For GENR8 our most general BNF is:

```
N = { L-System, Axiom, RewriteRule, Operator, Predecessor, Successor,
      Modifier, Condition, Segment, Constant }

T = { +, -, &, ^, \, /, ~, [, ], <, >, ->, Edge, i, If, Angle, Sync,
      BranchAngle, Weight, EdgeX, EdgeY, EdgeZ, Edge_i, Edge_i+1,
      Edge_i-1, = }

S = { <L-System> }
```

The production rules are defined as

```
<L-System> ::= <Axiom> <RewriteRule> { <RewriteRule> } Angle Constant
               [ Sync ] [ BranchAngle Constant ]
```

```
<Axiom> ::= <Segment> [ ~ ] + <Segment> [ ~ ] + <Segment>
            { [ ~ ] + <Segment> }

<RewriteRule> ::= <Predecessor> -> <Successor> [ <Condition> ]

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

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

<Modifier> ::= { <Segment> } |
               <Modifier> ‘‘[‘‘ <Successor> ‘‘]’’ <Modifier> |
               <Operator> <Modifier>

<Operator> ::= + |
               - |
               & |
               ^ |
               \ |
               / |
               ~

<Segment> ::= Edge |
              EdgeX |
              EdgeY |
              EdgeZ |
              Edge_i |
              Edge_i+1 |
```

```
                Edge_i-1 |

<Condition> ::= If i ‘‘<’’ Constant |
                If i ‘‘>’’ Constant |
                If i ‘‘=’’ Constant

<Constant> ::= 0 | 1 | 2 | 3 | ...
```

The terminal `Angle` is the parameter that controls how much the turtle should turn. If the `Sync` terminal is present, the branch merging is synchronous; if not it will be asynchronous. `BranchAngle` controls how the the tolerance when merging the branches.

Each `Edge` is assigned a type, written as a number after the word `Edge`. All `EdgeX` terminals in a production will be assigned the same type. `EdgeY` and `EdgeZ` work in the same way.

It is also possible to have probabilistic `RewriteRule`, which means that there are several possible `Successor` to a `Predecessor` and the choice is random (according to some predefined distribution). The `Weight` terminal gives the relative weight of a `Successor` when determining which production should be chosen for stochastic grammars.

3.2.1 BNFs for the EA

For the GE we use four BNFs, symmetric, reversible, probabilistic and default, that are slightly more restricted than the BNF described above. The reason for this is to narrow down the search space and obtain interesting results faster. These BNFs define different universes of surfaces, which gives the user greater control and flexibility. Full details on these BNFs can be found in Appendix A.

If a grammar contains an `Edge` that does not have any productions (or just the identity production), that `Edge` is essentially a ‘dead’ symbol (unless it is used in the

context of the other productions). To avoid a situation where none of the productions can be applied and to generate more interesting grammars, we have added a repair mechanism to the EA. It checks that there is at least one production for each edge type and if not, it creates one.

Since some of the rules are recursive, there is a maximum depth that restricts the expansion, limiting the length of the productions. When this depth is reached, we choose a production that only produces terminals. The default value for the maximum depth is five.

Symmetric

One of the BNFs create symmetric surfaces. This is achieved by starting out from a symmetric surface and then making sure that all productions conserve this property. In that way, we can guarantee that the surface will have at least one line of symmetry. Figure 3-2 shows a surface that was created using this BNF.

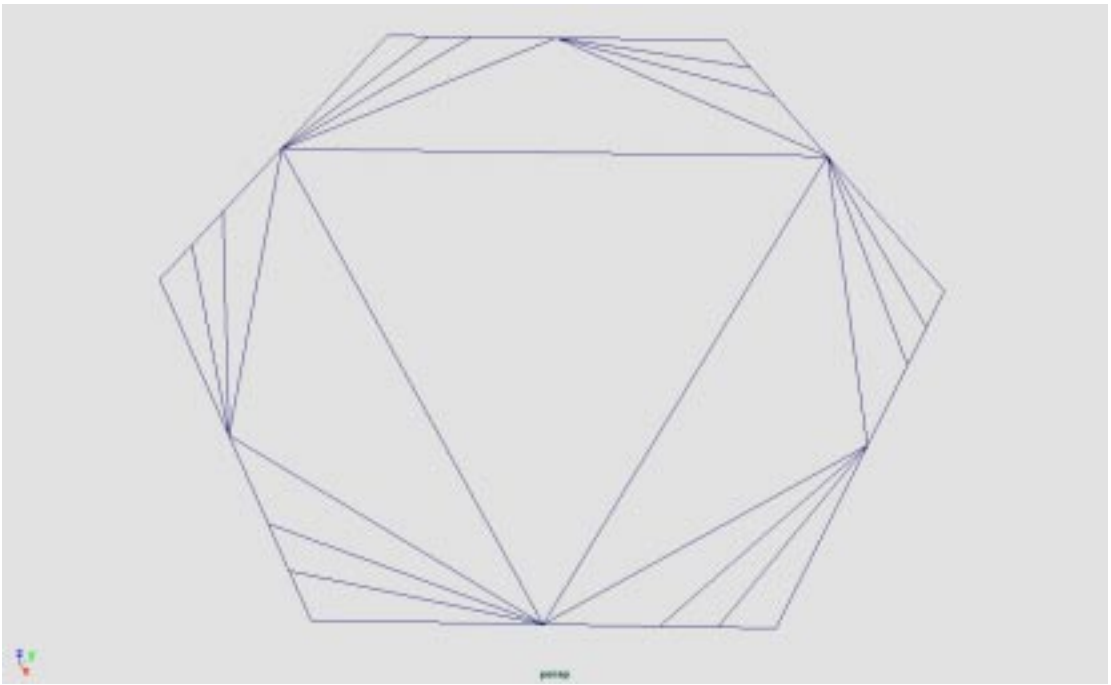

Figure 3-2: A surface that was evolved using the symmetric BNF.

Probabilistic

There is also one that creates probabilistic rewrite rules (the other three always produces deterministic rules). This leads to more unpredictable behaviour, but it also opens up new possibilities in a larger universe of surfaces.

Reversible

The third BNF produces reversible rules. We have implemented a function that can invert the mapping for this subset of BNFs. This allows us to map a grammar back to a genotype. More on this can be found in Section 4.2.1.

Default

There is a BNF that is set as default for the EA. It has been chosen to produce a lot of branches, which in turn produces many subdivisions. This is in general a desired property (otherwise the tendency is to get a loop that scales up and not much is happening, as can be seen in Figure 4-6). A grammar generated by the default BNF may look like this:

```
Edge1 + Edge1 + Edge2 + Edge2 + Edge2 + Edge3

Edge1 > Edge1 -> & [ [ - - Edge1 ] + + Edge1 ] ^ Edge3

Edge2 > Edge2 -> [ [ - - Edge0 ] + + Edge0 ] Edge2

Edge1 < Edge2 -> ~ [ [ - - Edge3 ] + + Edge3 ] Edge0

Edge3 -> + [ [ - - Edge3 ] + + Edge3 ] - Edge1

Edge0 > Edge0 -> / Edge3 Edge3 \ Edge2

Angle 45
```

```
BranchAngle 75
```

The resulting surface can be seen in Figure 3-3.

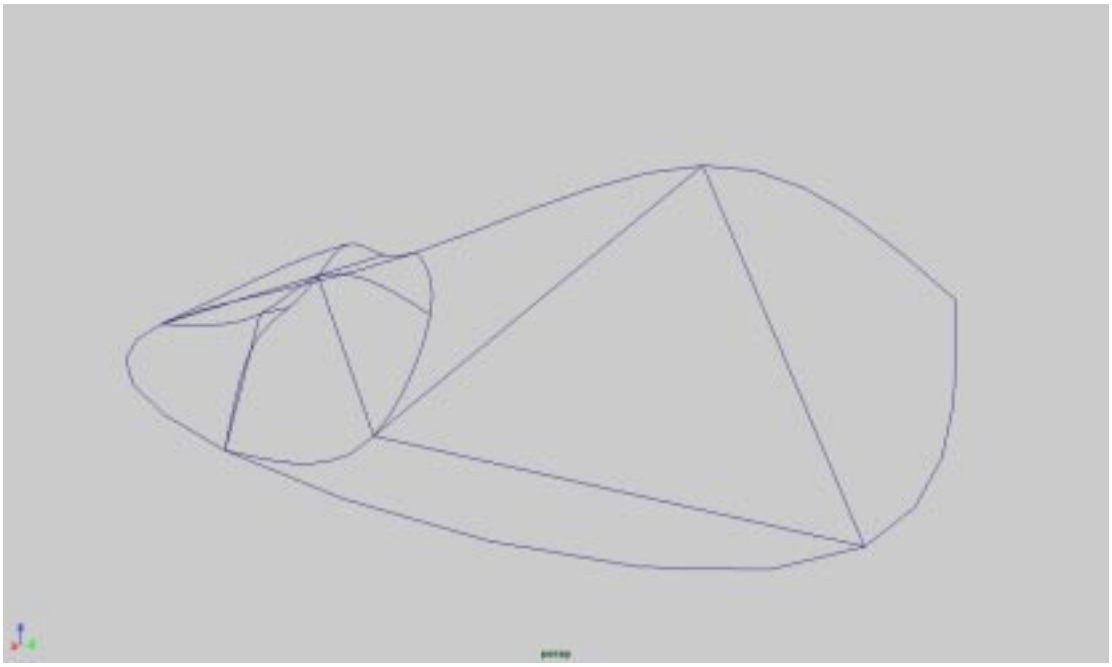

Figure 3-3: A surface that was evolved by the system from a population size of 15 after 20 generations

3.2.2 Mapping the genotype to BNF grammar

Commencing from a start symbol, `S`, the genome is read to determine what production rule should be used. Standard to Ryan and O'Neill, the gene values dictate the corresponding production rule. In the cases where a terminal or non-terminal in the production rule is optional or may occur multiple times, the genes are used to make the choice. In the case of multiple occurrences we use a method that is similar to an exponential probability distribution, so that there is no fixed upper limit to the number of occurrences of the terminal (or non-terminal). We illustrate the whole procedure with a short example, a longer example can be found in Appendix B.

Example

We start with a genome:

```
617 666 800 8
```

And we are want to expand the following expression:

```
{ <Segment> }
```

The brackets surrounding the non-terminal indicate that we are going to have an optional number of `<Segment>`. 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 expanding the `<Segment>` terminal. We have seven productions to choose from so the choice is made by taking `800` Mod $7 = 2$. Finally we determine the type of the `Edge` by taking `8` 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

Chapter 4

GENR8 as a tool

Evolutionary Algorithms can be exploited successfully in a design software. They offer powerful search within the universe of possible designs. However, this is not enough to make them a useful tool. The EA component has to be integrated into the rest of the tool seamlessly so that it can be used in an intuitive way.

To make GENR8 easily accessible for designers it has been implemented as a plug-in to an existing design tool. In this way it is easier to become familiar with it and, from a developing point of view, we get a lot of functionality for free. The downside of this is that we are restricted by the host software. Designers have a plethora of 3D tools at their disposal. These tools occupy specific niches and architects use them at different stages of the design process; this implies that it is paramount that designs are portable among the different tools. After consulting the architects in the Emergent Design Group, the choice of host software was Alias|Wavefront Maya. Many designers use this software and it has good support for writing plug-ins. The API is easy to use as it is implemented as C++ classes. In addition it has MEL that works on a higher level (it is a scripting language) and in many cases it is more handy than the API.

The EA component of GENR8 can be seen as a tool within a tool within a tool. At the top level, Maya is just one of many 3D-modeling software tools available to designers. GENR8 is just one of the many tools available when using Maya. Finally EAs is the tool used by GENR8 to create novel surfaces.

The user needs to have a basic understanding of Darwinian evolution to use EA

component of GENR8. It is not required that the user possesses in-depth knowledge about EAs, one barely has to grasp the essential notion of evolution; fitter individuals are recombined to produce new offspring. All this is intuitive to most people and there should not be any difficulties understanding the basic concepts of GENR8.

GENR8 includes two predefined grammars that are very useful, they produce regular three and four sided surfaces. Most of the figures in Chapter 2 used the four sided grammar. These grammars are a good way for the designer to get familiar with GENR8 and its environment since they produce a predictable result (with a complex environment and an evolved grammar it is hard to tell if it was the grammar or the environment that caused a certain phenomenon). Even without the EA, GENR8 is a powerful tool and a lot can be accomplished just with these two grammars.

There are also two predefined grammars that generate fractals. They have limited use when designing a surface. But fractals are interesting in themselves and generating fractals comes 'for free' with our model. In Figure 4-1 the Koch-curve has been generated with the aid of GENR8.

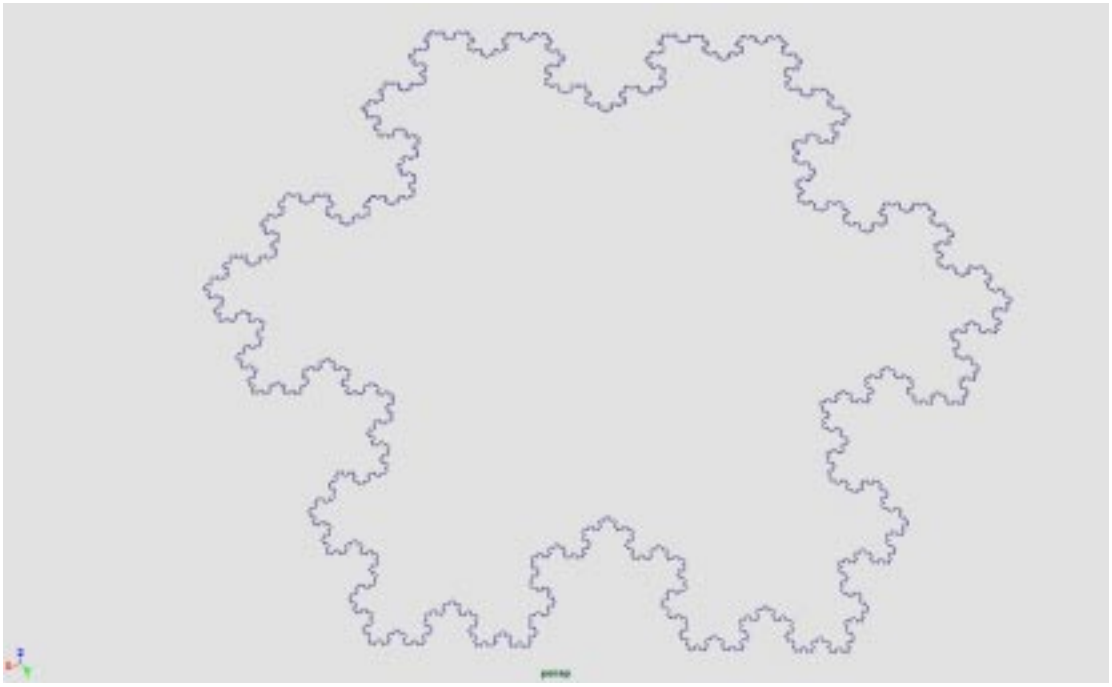

Figure 4-1: The Koch-curve, also known as the snow-flake.

The third way to provide the system with a grammar is to write your own and save

it to a file. The file can be parsed by GENR8 and the grammar will be interpreted as a surface. This feature is particularly useful if you want to try a small modification to a grammar produced by the system. The downside is that it is quite hard to understand what a grammar does and to know what changes to make to get a certain surface.

4.1 Integration into Maya

GENR8 is implemented as a MEL-command (Maya Embedded scripting Language) and one invokes it via command line in the same way as any other MEL command. Because MEL is based on a command line interface, which makes it somewhat unwieldly, we have implemented a graphic user interface (GUI).

From the GUI (and the command-line), one can set all the parameters (environmental and EA-related) for a design foray. The GUI is more user-friendly since it prevents the user from entering invalid combinations. To make life even easier for the user, there are help files in HTML-format readily available. Figure 4-2 shows a screen shot of the GUI.

The reader has probably deduced from Chapter 2 and 3 that there are a lot of parameters that can be adjusted. Fortunately, they all have default values, so one does not have to set them each time. Moreover it is possible to save the settings to a file. By doing this it is possible to reload them at a later time, sparing the user of the job of remembering the settings and changing the default values each time.

The user can set up a scene with the environment, working with Maya as usual. This is done outside GENR8 and the experienced Maya user will feel comfortable at home-turf.

4.1.1 Setting up the environment

Boundaries are set up using ordinary Maya surfaces and they will be treated as walls by GENR8 if they are on the selection list when the run is started. Attractors and repellors can be placed and edited using special-purpose commands. They are drawn

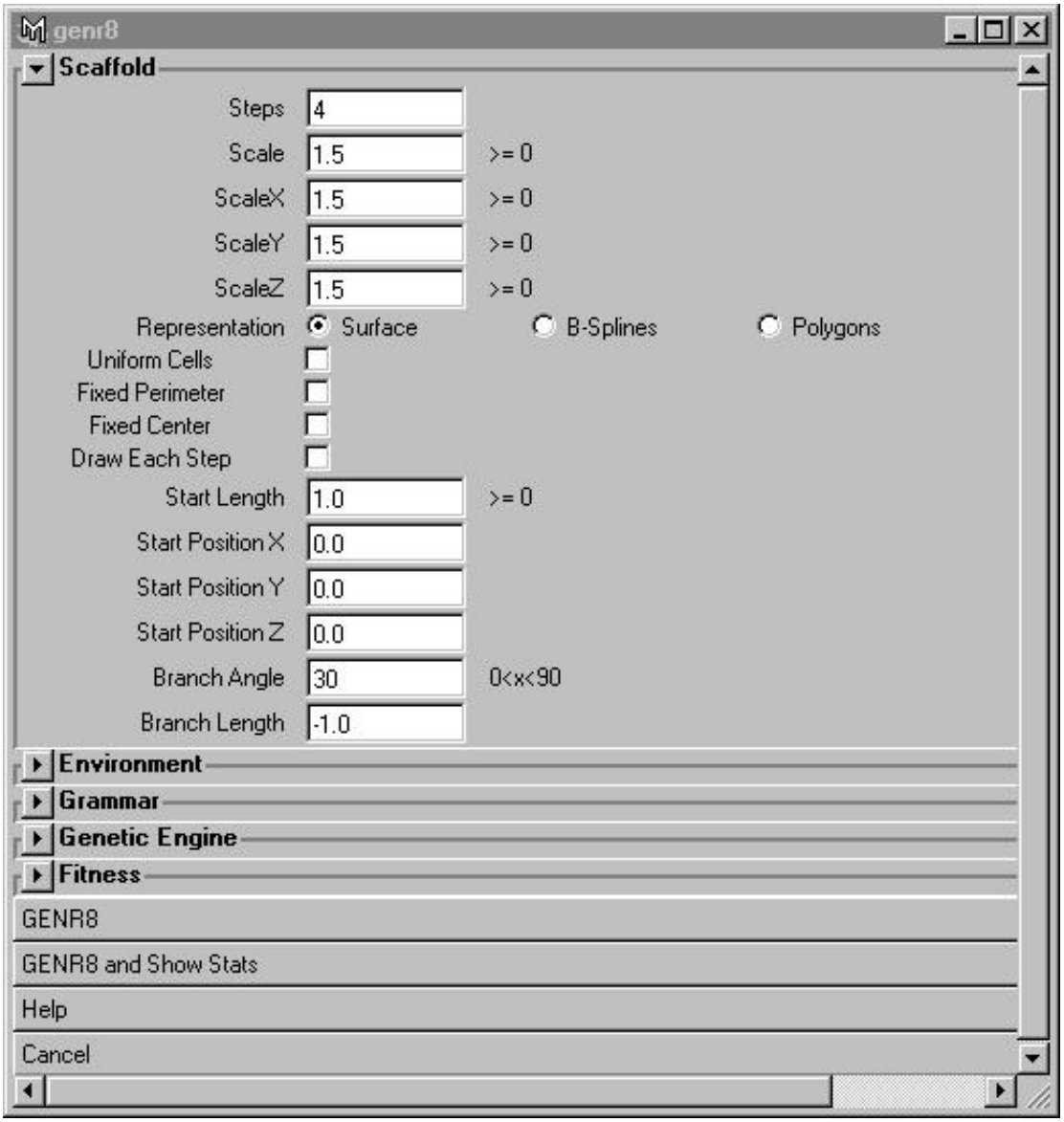

Figure 4-2: A screenshot of the GUI. Menus can be expanded or collapsed by clicking on the bars with the arrows.

in separate layers, so that it is easy to make them invisible after the run.

The user may draw a curve and that curve will be used as a starting point for the growth (this will override the genotypically encoded `Axiom` which always is a regular polygon). If the user chooses not to provide the system with a starting curve, it is possible to specify the location and size of the seed.

4.1.2 Examining the output

When the design foray is finished, relevant data is presented in a window. It includes the ranking of the population and the fitness value for each individual. For each member of the population, the grammar that generated it and the different fitness criteria (see Section 4.3) are displayed. The genome, the grammar or the Maya-scene can be saved by pressing a button in the GUI.

The surfaces are drawn in separate layers which makes it easy to toggle the visibility. This is very useful when inspecting the population. If the user wants to study an individual closer, there is a feature that allows one to re-grow that member and thus study it in closer detail. It is also possible to save both the grammar, the genome and the actual Maya surface.

4.2 Interruption, intervention and resumption

The traditional way to use EAs in design tools is to have the user set up a design foray and then wait for the output. This can be very frustrating to a designer since it may alienate him or her from the process. We do not want to create a black box that simply spits out a finished design; instead we are trying to make a tool that *cooperates with the designer*.

Our goal is to allow the designer to take an active role in the evolutionary process and to have a sense of control. An analogy that might be helpful is that of a car on cruise control. The car goes forward by itself but the driver is still in control of the steering and may hit the brakes, regaining full control. We have labeled this idea interruption, intervention and resumption [19]. The user should be able to interrupt

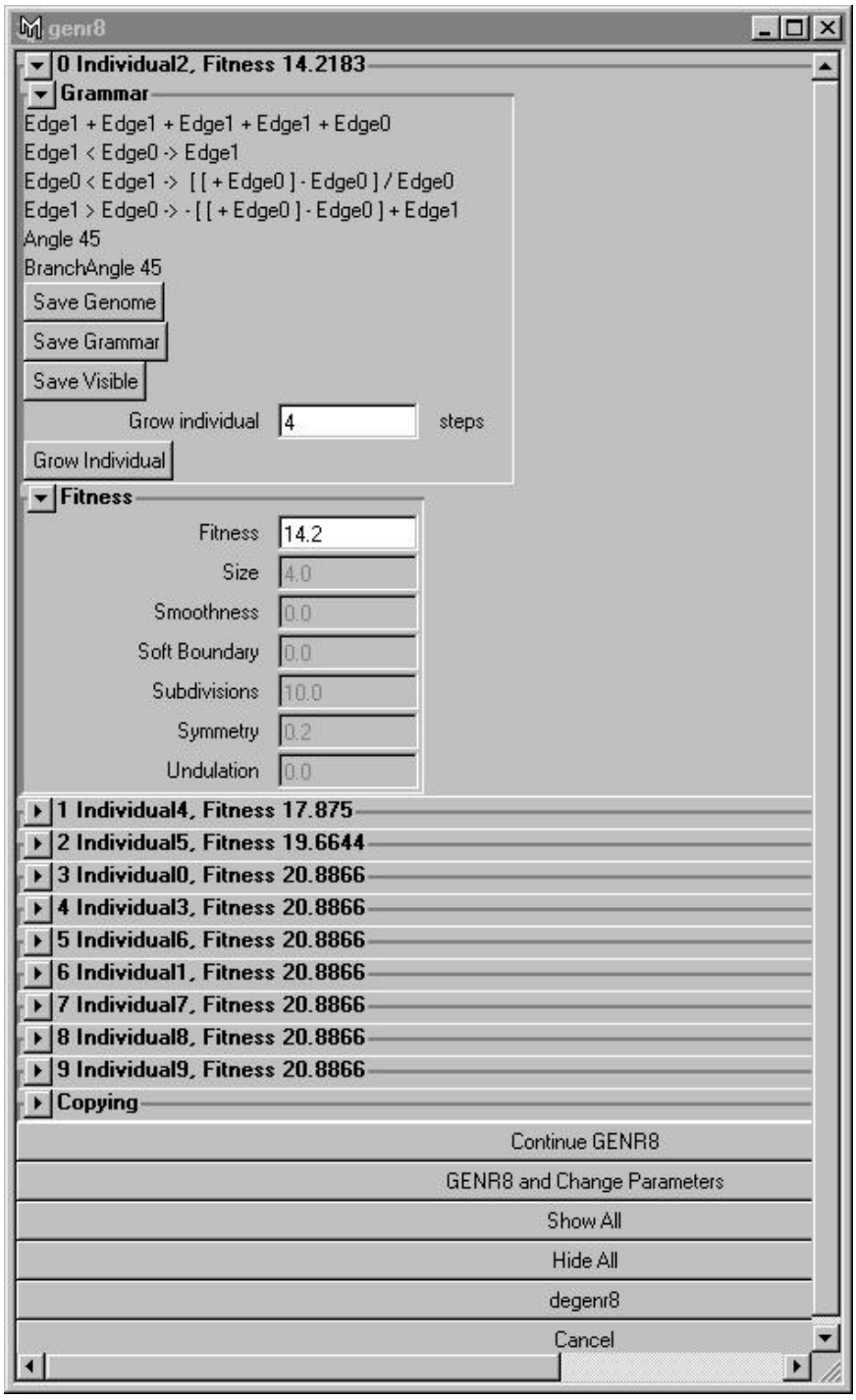

Figure 4-3: A screenshot of the output window. The visibility of the individuals is turned on and off as the menus are expanded or collapsed.

the EA at any time, for instance, the user might spot an interesting design. Next, the user should be able to intervene by changing parameters (eg mutation rate) and the environment (eg add an attractor). Finally it should be possible to resume the process from where it was interrupted.

Interruption is straightforward to implement. The requirements are that the user should be continuously updated on the results of the design process and have some sort of control that interrupts the design foray. In GENR8, the designs are shown on the screen as they evolve. Relevant data and statistics are displayed in another window. By pressing the Esc-key, the design foray is interrupted.

Intervention is trickier to deal with. Ideally it should not only be possible to change the parameters and the environment before resuming. We would also like to be able to modify the phenotype and have those changes mapped back to the genotype. We have not been able to implement this in GENR8 due to the complexity of the growth model because HEMLS are extremely complex to invert. However, we have been able to invert the first mapping, from genotype to grammar. This process, termed regn8, is described in the next section. When a design foray is ended (or interrupted) the user may change any parameters or set up a new environment before continuing, with the old population. It is also possible to make copies of an individual and insert those into the population, indirectly nudging the evolutionary process in the desired direction. Moreover, one can load a saved population from a file if one wishes to introduce specific genetic material into the population.

The requirement for resumption is that we should be able to restart the process with updated parameters. It is interrelated with intervention and if intervention is cleverly implemented, resumption is very easy to implement. Thus it is intervention that is the trickiest part of IIR.

4.2.1 regn8

Although it would be desirable to allow the user to change the phenotype and have those changes mapped back to the genotype, it has not been implemented in GENR8. The reason for this is the difficulty of inverting HEMLS, ie deducing what grammar

created a given surface. However, it is possible to invert the first mapping in GENR8, from genotype to grammar.

Thus a user can construct a grammar by hand or modify an evolved grammar and give that as input to regn8. Regn8 then inverts the mapping and outputs a genome. This user-defined genotype can be inserted into the population as the starting point for further design forays. Since the grammars are quite hard to understand, this feature may be of limited use to a designer with limited knowledge of computer science.

Since we know the production rules for expanding the BNF, we can use that knowledge to reverse the process. As the grammar is expanded, a tree-structure is built up where the leaves constitute the resulting grammar. If the multiple (eg { `<Segment>` }) and optional (eg `[ <Sync> ]`) terminals and non-terminals are used restrictedly, it is possible to reconstruct the entire tree just from the grammar (ie the leaves). The reversible BNF has been chosen so that this is possible and we can construct one of the genomes that will give us the input grammar.

4.3 Design evaluation

A key issue for EAs is the fitness evaluation, it determines which individuals will be reproduced in the next generation. For creative design this is obviously a very hard task, since it involves aesthetic criterion rather than an objective goal function.

The most common approach is to determine fitness either by using a mathematical function to evaluate the design (and thus we are once again back to optimization) or fitness is directly determined by the user. Since it is obviously very hard to capture the nature of the outcome of a creative design process in a mathematical function, this approach has some severe limitations. This approach is more useful when trying to optimize an existing structure or recombining existing elements.

The latter approach is called interactive evolutionary computation [26] [8] [27] [30] [17] and it deviates from traditional AI-techniques in one important aspect; instead of trying to model the behaviour of the human user, it tries to incorporate the user into the system. The main advantage with IEC is that it does not constrain the creative

process and there is no need to create a fitness function.

Traditional IEC often runs into problems with human fatigue [30] [17]. The user has to evaluate the entire population between each generation, which means that the user must do a lot of work. This limits the population size and the number of generations in practical use. To circumvent this problem, IEC approaches tend to focus on getting faster convergence, using enhanced interfaces to trying to predict the user's preferences.

In GENR8, the user can set the fitness values for the population by hand as is standard in IEC. However, that is not how the tool is intended to be used primarily. GENR8 instead allows the user to express his or her preferences by setting the parameters for the fitness function (described below). Another more indirect way to direct the evolution is to change the environment. Altering the environment will make the grammars produce different surfaces. Since the selection and fitness evaluation is based on the surfaces, this will affect the breeding.

4.3.1 Fitness function

The concept of a fitness function with multiple components serves the designer well. It is easy to emphasize the features that one is most interested in and it gives the system some of the human ability to be responsive to several criteria at the same time. The parameters are independent and can be used to express multi-level, non-linear and possibly conflicting goals of a designer. Recall, the weight and parameters of any criteria can be changed at any time during a design foray.

For each criterion there is a weight associated that controls the relative importance of that criterion when calculating fitness. The user sets a target value for each criterion, and deviations from that value are penalized.

Size

Size is a measure of the extent of the surface in the x and the y direction. Other ways to control the size of the surface is by adjusting the scale and the starting length of

the seed or to use a fixed perimeter and a user defined seed.

Smoothness

Figure 4-4: A rugged surface. The view is from the side.

The smoothness criterion has two components, one local and one global. The global smoothness, undulation, simply measures the difference between the minimum and the maximum z values. The local component calculates the difference in z values for each vertex and its neighbours.

Soft boundaries

This is, in fact, a fourth wall-behaviour and thus it is only useful in an environment where boundaries are present. If this wall behaviour is used, the surface can grow through the boundary, but it incurs a fitness penalty as it does so.

Subdivisions

This is a measure of how subdivided a surface is. Subdivisions are created when branches are merged. The criterion is calculated as the ratio between the number of

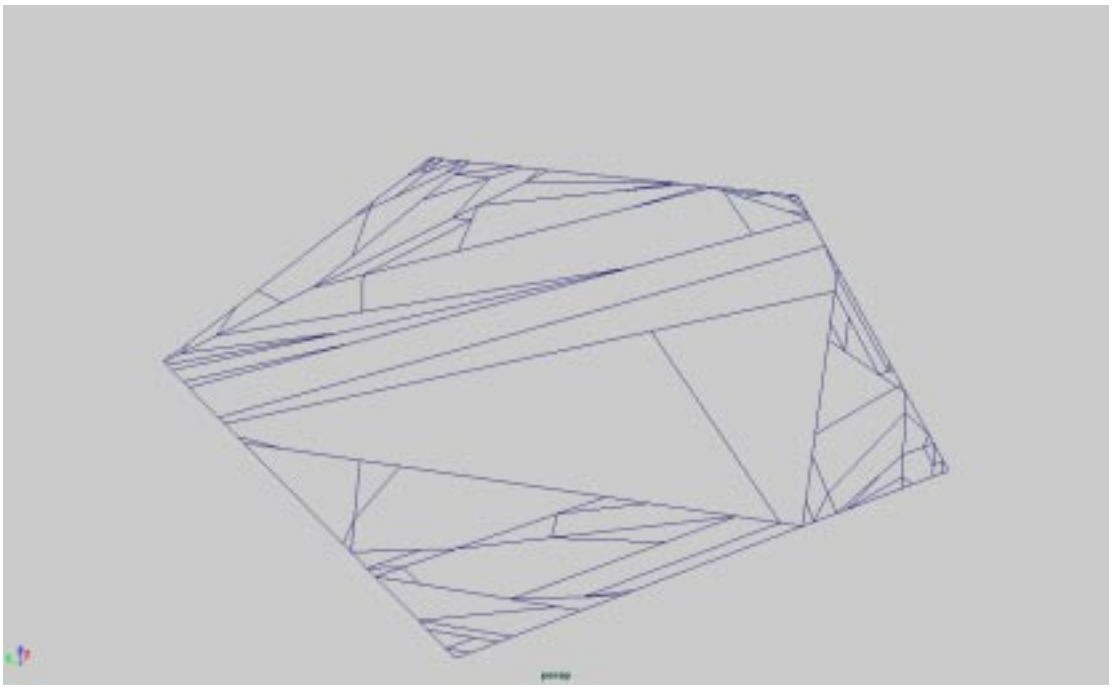

Figure 4-5: A surface with many subdivisions.

vertices and edges.

Symmetry

Symmetry measures how the vertices are distributed with respect to the x and y axis. If a completely symmetric surface is desired, it is better to use the symmetric BNF; it always produces symmetric surfaces. This criterion only penalizes deviations from the ideal, it does not prevent them from being created.

Example

Figure 4-6 shows the best individual in a (randomized) starting population of size 50. When the parameters of the fitness function have their default values. If we then change the parameters to increase the number of subdivisions, after 5 generations, we get the result shown in Figure 4-5. If we wish to then make the surface more rugged rather than flat, we change the smoothness criteria. The result can be seen in Figure 4-4.

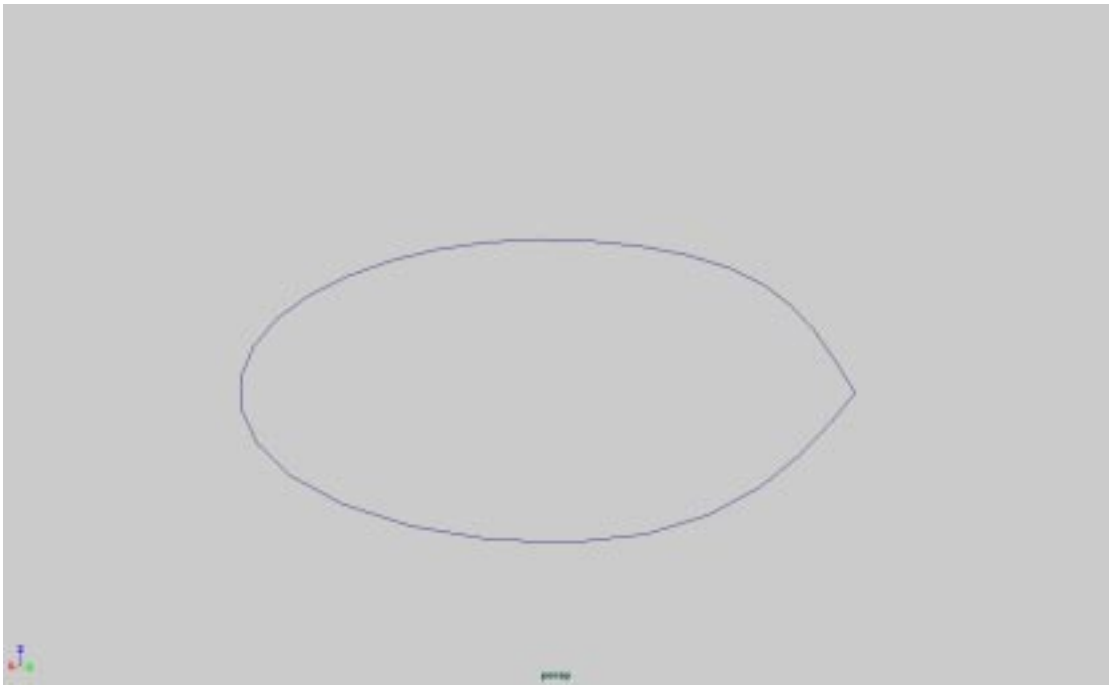

Figure 4-6: A randomized starting individual. Since no branches have merged, there are no subdivisions.

Chapter 5

Future Work

5.1 Solids

GENR8 creates surfaces, but we would like to extend the tool so that we can grow solid objects as well. In the next incarnation we could use a solid modeler instead of Maya (it can only model surfaces). Solid modeling permits one to analyze the structural and material properties of the designs and incorporate this in the fitness evaluation. This could, for instance, be done using some finite element method.

An open question at this point is how to modify HEMLS so that they can be used to generate solids. There are three major issues that must be addressed; grammar, growth and interpretation. We need an entirely new grammar to represent a solid object, the current one can only produce surfaces. Then we have to consider the growth model; will it still be adequate for solids? Finally we must find a suitable graphical representation. Will turtle graphics still be a good way to draw the design? It is quite possible that we end up with something that does not bear much resemblance to L-systems whatsoever.

5.2 3D-Visualization

GENR8 creates surfaces in 3D space and it would certainly be interesting to be able to view and evaluate the surfaces in a 3D environment rather than on a 2D

monitor. Thus we would like to extend GENR8 through the use of some virtual reality technology.

5.3 Learning User Behaviour

An interesting feature would be a machine learning system that observes the users' behaviour and deduces the users' preferences. This would lead to even more efficient and personalized fitness evaluation. There are two possible strategies for how this could be implemented. One way is to have a set of parameters for each user that are updated as the tool is used. The other strategy is to have the same set of parameters updated by all (or a group of) users. This could lead to an interesting (and possibly frustrating) mix of preferences.

Chapter 6

Conclusions

6.1 Computer science

6.1.1 ALife

In this thesis we have extended the map L-systems model to make them work in 3D. We have also incorporated concepts from ordinary L-systems to make it a more powerful and expressive model. These features are context sensitivity, time-variation, stochasticity and environmental influence. The environment is especially interesting and important, because it makes the growth reactive.

6.1.2 Evolutionary computation

EAs are very useful for optimization problems and have been used in that field for decades. We are trying to use them for generative purposes, a much harder task because it involves aesthetic aspects.

In GENR8 we have implemented IIR, to give the user greater control of the evolutionary process. The user does not have to await the final output, he or she is free to intervene at any time, giving the user greater control of the creative process.

We have exploited the grammatical evolution EA in a field that the inventors most likely did not expect it to be used. A part of this work was to give a BNF specification of the grammar describing HEMLS.

GENR8 has been seamlessly integrated into Maya and for someone who is familiar with Maya, it is intuitive to use. The output of the system are ordinary Maya objects and the user can use any other tool in Maya to continue working on the surface that was produced by GENR8.

One does not need an expert knowledge of EAs in order to use GENR8, the GUI allows the user to express his or her preferences without any knowledge of what is going on 'behind the monitor'. The user does not have to assign a fitness value to each surface, this is done by a fitness function with multiple parameters. The parameters express different aspects and qualities of a surface. This greatly reduces human fatigue.

6.1.3 Combining EC and HEMLS

As we mentioned in Chapter 1, this thesis is in the field of emergent design. One of the main concepts in emergent design is to combine EC and ALife. By doing this, we can use the searching capabilities of EC and the combinatorial and generative aspects of ALife. Thus we are able to search a large universe of designs.

6.2 Architecture

We have developed a new tool that is much more cooperative than ordinary CAD-tools; which is rare in current architect tools. GENR8 is creative and it can help the designer by suggesting new concepts. Ordinary computers sit on the architects deskspace, but do not usually provide more than basic CAD tasks. The computer is certainly powerful enough to perform creative tasks. However, this requires new tools such as GENR8.

The main notion behind GENR8 is emergence; a concept that is well suited for architectural design. Emergence is an interesting phenomenon and it allows us to construct designs in a bottom-up fashion. Thus we can create designs that consists of primitive elements that combine to form intricate structures.

Appendix A

BNF Specifications

In the following sections we give the specifications of the BNFs as they are implemented in GENR8.

A.1 Default

In Appendix B there is an example of how the BNF is used to map a genotype to a grammar.

```
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 }
```

A.2 Reversible

We only needed a minor modification to the default BNF to make it reversible.

```
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 }
```

A.3 Symmetric

The strategy behind this BNF is to start out with a symmetric seed and then make sure that all operations preserve that quality.

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

T = { +, -, &, ^, \, /, ~, [, ], <, >, ->, Edge, Angle, Sync, EdgeX,
      EdgeY, EdgeZ, BranchAngle }
```

```
S = { <L-System> }

P = {

<L-System> ::= <Axiom> <RewriteRule> { <RewriteRule> } Angle 45
               [ Sync ] BranchAngle BranchAngleValue

<Axiom> ::= <EdgeX> + <EdgeY> + <EdgeX> + <EdgeY> |
            <EdgeX> + <EdgeY> + <EdgeX> + <EdgeY> + <EdgeX> + <EdgeY> |
            <EdgeX> + <EdgeY> + <EdgeZ> + <EdgeY> + <EdgeX> + <EdgeZ> |
            <EdgeX> + <EdgeY> + <EdgeX> + <EdgeY> + <EdgeX> + <EdgeY>
            <EdgeX> + <EdgeY>

<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>

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

A.4 Probabilistic

The probabilistic BNF do not use the `Weight` terminal, if the terminal is abscent, GENR8 gives each successor an equal weight by default.

```
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> }

<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 }
```

Appendix B

Example of how a genotype is mapped to a grammar

We start with a genome, an array of integers:

212, 187, 632, 832, 800, 122, 517, 338, 197, 39, 878, 185, 954, 863, 660, 276, 909, 321, 545, 240, 670, 231, 292, 158, 494, 999, 658, 844, 316, 710, 362, 27, 194, 144, 171, 243, 260, 414, 337, 790, 309, 344, 456, 30, 134, 13, 774, 162, 911, 222

This array is going to be used to expand the L-system grammar by choosing appropriate production rules. We start out with the start symbol, `S` and use the production rules for the default BNF (Appendix A.1). `<L-System>` has only one production rule, so the gene, 212, is irrelevant as the non-terminal is expanded to

```
<Axiom> <RewriteRule> { <RewriteRule> }
```

Next the `<Axiom>` is expanded, again there is only one rule, so we get the same result regardless of the gene:

```
<Edge> "+" <Edge> "+" <Edge> { "+" <Edge> }
<RewriteRule> { <RewriteRule> }
```

In fact there is a special end-symbol which allows GENR8 to keep track of where the rules end, but we do not have to worry about that now. For the edges we must

determine what type they are going to have. Type = (number-of-types+1) Mod gene. Thus we have a decreasing probability of introducing new types. From the beginning there are two types, so we have 632 Mod 3 = 2.

```
Edge2 + <Edge> "+" <Edge> { "+" <Edge> }
<RewriteRule> { <RewriteRule> }
```

We have two more edges and the genes are 832 and 800 which gives us 832 Mod 4 = 0 and 800 Mod 4 = 0.

```
Edge2 + Edge0 + Edge0 { "+" <Edge> }
<RewriteRule> { <RewriteRule> }
```

The next symbol, , tells us that we are going to have 0 or more of whatever is between the brackets. We determine the exact amount in a way that is similar to what we did with the edge types. To see if we are going to have another <Edge>, we test for (gene Mod (occurrences + X)) > occurrences we add another <Edge>. Occurrences starts at 0 and is increased each time we add a new Edge. X is an integer >= 2 that can be used to adjust the probabilities, in our example, X is 2. We have 122 Mod 2 = 0, which means that there will be no more symbols for the seed.

Next we let the 517 expand the <RewriteRule>.

```
Edge2 + Edge0 + Edge0
<Predecessor> -> <Successor>
{ <RewriteRule> }
```

For the <Predecessor> we have four different production rules, and the gene is 338 Mod 4 = 2 which means that we should take the third production rule.

```
Edge2 + Edge0 + Edge0
<Edge> ">" <Edge> -> <Successor>
{ <RewriteRule> }
```

For the predecessors we do not want to introduce new edge types (this could lead to dead rules) so instead of number-of-types+1 we use number-of-types, which gives us 197 Mod 3 = 2 and 39 Mod 3 = 0.

```
Edge2 + Edge0 + Edge0
Edge2 > Edge0 -> <Successor>
{ <RewriteRule> }
```

In the next step, 878 is used to expand the `<Successor>`

```
Edge2 + Edge0 + Edge0
Edge2 > Edge0 -> <Modifier> <Edge>
{ <RewriteRule> }
```

The next gene, 185 Mod $9 = 5$ gives us the next expansion.

```
Edge2 + Edge0 + Edge0
Edge2 > Edge0 -> "^" <Modifier> "&" <Edge>
{ <RewriteRule> }
```

Once again we expand a `<Modifier>` with 954 Mod $9 = 0$.

```
Edge2 + Edge0 + Edge0
Edge2 > Edge0 -> "^" { <Edge> } "&" <Edge>
{ <RewriteRule> }
```

Now we have the brackets again and the next gene (again X=2), 863 Mod 2 tells us that we should have an `Edge`. The next gene 660 Mod $4 = 0$ sets the type. We test to see if we should have another `Edge`, but 276 Mod $3 = 0$ means that we should stop adding edges.

```
Edge2 + Edge0 + Edge0
Edge2 > Edge0 -> ^ Edge0 & <Edge>
{ <RewriteRule> }
```

Going on we expand the `<Edge>` with 909 Mod $4 = 1$. Next we test if we are going to add more `<RewriteRule>` (X=5), with the gene 321 Mod $5 = 1$, continuing, we have 545 Mod $5 = 0$, which means that we should only add one more `<RewriteRule>`.

```
Edge2 + Edge0 + Edge0
Edge2 > Edge0 -> ^ Edge0 & Edge1
<RewriteRule>
```

It is left as an exercise for the reader to verify that the `<RewriteRule>` is expanded to.

```
Edge2 + Edge0 + Edge0
Edge2 > Edge0 -> ^ Edge0 & Edge1
Edge0 > Edge1 -> [ [ + + Edge0 ] - - Edge0 ] Edge0
```

Now we see that we do not have any productions for `Edge1`, which means that it is a 'dead' `Edge`, once we create something with type one, there will be no interesting developments for that segment. To get around this problem, there is a repair mechanism that makes sure that there is at least one production rule for each segment (although the context sensitivity may make the production rules useless anyway). Analyzing the above set of rewrite rules, we see that we need to add a production rule where `Edge1` is the predecessor. During the repair phase, we are not allowed to increase the number of edge types. After a few expansions we get:

```
Edge2 + Edge0 + Edge0
Edge2 > Edge0 -> ^ Edge0 & Edge1
Edge0 > Edge1 -> [ [ + + Edge0 ] - Edge2 ] Edge0
Edge1 -> Edge0
```

Finally, we are going to set the value for the parameters, using the genome. First we set the angle to ((gene Mod 3)+1)·15, giving us ((337 Mod 3)+1)·15 = 30. Then we find out if we are going to have synchronous growth by testing 790 Mod 2 = 0 (which means that we are going to have asynchronous growth). The `BranchAngle` is determined by ((gene Mod 5)+1)·15, in our case 75. We are now finished and after having used 41 genes, we have grammar that we can use to generate a scaffold.

```
Edge2 + Edge0 + Edge0
Edge2 > Edge0 -> ^ Edge0 & Edge1
```

```
Edge0 > Edge1 -> [ [ + + Edge0 ] - Edge2 ] Edge0
Edge1 -> Edge0
Angle 30
BranchAngle 75
```

If you type the above into a txt-file, you can use GENR8 to find out what it looks like. The result may vary depending on the environment, so it is hard to draw any immediate conclusions (although I think that it will be a very boring structure that unable to subdivide in an interesting way).

Bibliography

[1] Peter Bentley, editor. *Evolutionary Design by Computers*. Morgan Kaufmann, May 1999.

[2] Peter Bentley. From coffee tables to hospitals: Generic evolutionary design. In Peter Bentley, editor, *Evolutionary Design by Computers*, chapter 18. Morgan Kaufmann, May 1999.

[3] Peter Bentley. An introduction to evolutionary design by computers. In Peter Bentley, editor, *Evolutionary Design by Computers*, chapter 1. Morgan Kaufmann, May 1999.

[4] 2001. http://cui.unige.ch/db-research/Enseignement/analyseinfo/AboutBNF.html.

[5] T. Broughton, Paul Coates, and Helen Jackson. Exploring 3d design worlds using lindenmayer systems and genetic programming. In Peter Bentley, editor, *Evolutionary Design by Computers*, chapter 14. Morgan Kaufmann, May 1999.

[6] Roger Curry. On the evolution of parametric l-systems. http://pharos.cpsc.ucalgary.ca/Dienst/UI/2.0/Describe/ncstrl.ucalgary_cs/1999-644-07.

[7] Emergent design group, MIT, 2001. http://web.mit.edu/arch/edg/.

[8] W. Hsu and B. Liu. Conceptual design: issues and challenges. *Computer-Aided Design 32*, pages 849–850, 2000.

[9] Christian Jacob. Genetic l-system programming:breeding and evolving artificial flowers with *mathematica*. In *First International Mathematica Symposium*, pages 215–222, Southampton, UK, 1995. http://www2.informatik.uni-erlangen.de/IMMD-II/Persons/jacob/Publications/.

[10] Gabriella Kókai, Zoltán Tóth, and Róbert Ványi. Modelling blood vessels of the eye with parametric l-systems using evolutionary algorithms. In W. Horn et al., editor, *AIMDM'99, LNAI 1620*, pages 433–442. Springer-Verlag, 1999.

[11] Ladislav Kovács. Simultaneous effects of the environment and the tropism to the growth of a tree. In *Proceedings of the 4th Central European Seminar on Computer Graphics for students (CESCG 2000)*, 2000.

[12] Cambrian Labs, 2000. http://www.cambrianart.com.

[13] Maria Lantin and F. David Fracchia. Generalized context-sensitive cell systems. In *Presented at the First International Workshop in Information Processing in Cells And Tissues*, September 1995.

[14] Lauren Lapre. http://www.xs4all.nl/ ljlapre/.

[15] J Lluch, MJ Vicent, R Vivó, and R Quirós. Green: A tool for modeling natural elements. In *The 8th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media*, pages 52–59, 2000. http://www.dsic.upv.es/users/sig/investigacion/papers.html.

[16] M Marcotty and H Ledgard. *The World of Programming Languages*. Springer-Verlag, Berlin, Germany, 1986.

[17] Hiroaki Nishino, Hideyuki Takagi, Sung-Bae Cho, and Kouichi Utsumiya. A 3d modeling system for creative design. In *The 15th Int'l Conf. on Information Networking (ICOIN-15)*, pages 479–486, Beppu, Japan, January 2001.

[18] Una-May O'Reilly, Markus Kangas, and Peter Testa. Moss: Morphogenetic surface structure, a nature inspired tool. In submission to Parallel Proble Solving from Nature - 2000.

[19] Una-May O'Reilly, Peter Testa, Simon Greenwold, and Martin Hemberg. Agency-gp: Agent-based genetic programming for surface design. Submitted as a late-breaking paper to GECCO 2001.

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

[21] Craig Reynolds. Boids, 1986. http://www.red3d.com/cwr/boids/.

[22] Mike Rosenman and John Gero. Evolving designs by generating useful complex gene structures. In Peter Bentley, editor, *Evolutionary Design by Computers*, chapter 15. Morgan Kaufmann, May 1999.

[23] Conor Ryan, JJ Collins, and Michael O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In *Lecture Notes in Computer Science 1391. First European Workshop on Genetic Programming*, 1998.

[24] Conor Ryan and Michael O'Neill. Grammatical evolution: A steady state approach. In *Second International Workshop on Frontiers in Evolutionary Algorithms*, pages 419–423, 1998.

[25] Conor Ryan, Michael O'Neill, and JJ Collins. Grammatical evolution: Solving trigonometric identities. In *In proceedings of Mendel 1998: 4th International Mendel Conference on Genetic Algorithms, Optimisation Problems, Fuzzy Logic, Neural Networks, Rough Set.*, pages 111–119, Brno, Czech Republic, June 1998.

[26] Tomoya Sato and Masafumi Hagiwara. Idset: Interactive design system using evolutionary techniques. *Computer-Aided Design 33*, pages 367–377, 2001.

[27] C.H. Séquin. Cad and the arts. *Computer-Aided Design 33*, pages 345–348, 2001.

[28] Karl Sims. Evolving three-dimensional morphology and behaviour. In Peter Bentley, editor, *Evolutionary Design by Computers*, chapter 13. Morgan Kaufmann, May 1999.

[29] Celestino Soddu and Enrica Colabella. Argenia - generative art, 2001. http://www.celestinosoddu.com.

[30] Hideyuki Takagi. Interactive evolutionary computation: Fusion of the capabilities of ec optimization and human evaluation. Paper draft for proceeding of the IEEE that will appear in 2001 summer, 2001.

[31] 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. http://www.aimit.edu/people/unamay/papers.html.