

# Automatic Structure Generation using Genetic Programming and Fractal Geometry

by

*Steve Raphael Bergen*

A thesis submitted to the  
School of Graduate Studies  
in partial fulfilment of the  
requirements for the degree of  
Master of Science

Department of Computer Science  
Brock University, St. Catharines, Ontario

© *November, 2011*

## Abstract

Three dimensional model design is a well-known and studied field, with numerous real-world applications. However, the manual construction of these models can often be time-consuming to the average user, despite the advantages offered through computational advances. This thesis presents an approach to the design of 3D structures using evolutionary computation and L-systems, which involves the automated production of such designs using a strict set of fitness functions. These functions focus on the geometric properties of the models produced, as well as their quantifiable aesthetic value - a topic which has not been widely investigated with respect to 3D models. New extensions to existing aesthetic measures are discussed and implemented in the presented system in order to produce designs which are visually pleasing. The system itself facilitates the construction of models requiring minimal user initialization and no user-based feedback throughout the evolutionary cycle. The genetic programming evolved models are shown to satisfy multiple criteria, conveying a relationship between their assigned aesthetic value and their perceived aesthetic value. Exploration into the applicability and effectiveness of a multi-objective approach to the problem is also presented, with a focus on both performance and visual results. Although subjective, these results offer insight into future applications and study in the field of computational aesthetics and automated structure design.

## Acknowledgements

This work would not have been possible if not for the constant support and help from my family, friends, students, staff and mentors. I would like to specifically acknowledge the following people/organizations for their contributions:

- Brian Ross, for his endless barrage of ideas, suggestions, comments and support. A greater mentor there is not
- My parents, Lesley and Steve, for their constant support and understanding for my work and choices, despite a lack of understanding for what my work actually is
- Elicia and my brothers, Rob, Jim, Colin and Nick for similar support
- My friends, for giving me a place other than school to escape to, and my classmates in the CSC, for giving me a place on campus to hide
- Cale Fairchild, for his patience in dealing with someone who consistently requires his aid but refuses to learn Unix
- Brock University and the Computer Science Department, for the use of their resources, funding and facility

*“You! You’re the same. No matter where you go, there you are. It’s always the same old you. Let me suggest that you take a vacation from yourself. I know it sounds wild. It is the latest thing in travel. We call it the Ego Trip.”*

— Bob McClane

# Contents

<b>Abstract</b>	<b>ii</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>List of Tables</b>	<b>vii</b>
<b>List of Figures</b>	<b>viii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Goals . . . . .	3
1.1.1 Applicability Evolutionary Computation to Problem . . . . .	3
1.1.2 Extension of Existing Aesthetic Measures . . . . .	3
1.1.3 Implement Improved L-System Encoding for Genetic Programming . . . . .	4
1.2 Thesis Structure . . . . .	4
<b>2 Background Information</b>	<b>5</b>
2.1 Shape Grammars . . . . .	5
2.2 Lindenmayer Systems . . . . .	6
2.2.1 Validity of an L-system . . . . .	9
2.3 Voxels . . . . .	9
2.4 Evolutionary Computation and Genetic Programming . . . . .	10
2.4.1 Generational Algorithm . . . . .	10
2.4.2 Chromosome Representation . . . . .	10
2.4.3 Initialization . . . . .	11
2.4.4 Solution Evaluation . . . . .	12
2.4.5 Selection . . . . .	12
2.4.6 Reproduction – Crossover and Mutation . . . . .	12
2.4.7 Elitism . . . . .	14
2.5 Multi-objective Genetic Programming . . . . .	14

2.5.1	Pareto Ranking . . . . .	14
2.5.2	Rank-Sum . . . . .	15
<b>3</b>	<b>Literature Review</b>	<b>16</b>
3.1	Art and Evolution . . . . .	16
3.2	Research in Aesthetics . . . . .	18
<b>4</b>	<b>Tree Encoding and Model Generation Process</b>	<b>23</b>
4.1	System Loop . . . . .	24
4.2	Genetic Programming and Tree Encoding in System . . . . .	25
4.2.1	L-system Encoding Requirements . . . . .	25
4.2.2	Function Set . . . . .	27
4.2.3	Walkthrough Decoding of Sample Chromosome . . . . .	31
4.3	L-systems Used and their Respective Alphabets . . . . .	32
4.3.1	Voxel-space Drawing Alphabets . . . . .	33
4.3.2	Plant-based Alphabet . . . . .	33
4.3.3	L-system Alphabet . . . . .	33
4.4	Model Conversion Process . . . . .	34
4.4.1	Voxel Distance Calculation . . . . .	35
4.4.2	Marching Cubes Algorithm . . . . .	35
4.4.3	Post-processing Steps . . . . .	35
<b>5</b>	<b>Model Evaluation</b>	<b>38</b>
5.1	Model Constraint Functions . . . . .	39
5.1.1	Volume . . . . .	39
5.1.2	Dimension . . . . .	39
5.1.3	Surface Area . . . . .	40
5.1.4	Unique Surface Normals . . . . .	40
5.2	Aesthetics-based Functions . . . . .	40
5.2.1	Distribution-based Functions . . . . .	40
5.2.2	L-system Complexity . . . . .	43
5.2.3	Symmetry . . . . .	44
5.3	Fitness Targets and Analysis of Pre-Existing Models . . . . .	45
<b>6</b>	<b>Setup of Experiments and L-system Improvements</b>	<b>52</b>
6.1	Outline . . . . .	52
6.2	Experiment Parameters . . . . .	52
6.3	Analysis of L-system Encoding Improvements . . . . .	54

6.3.1	Introduction and Setup . . . . .	54
6.3.2	Results . . . . .	55
6.3.3	Conclusions . . . . .	57
<b>7</b>	<b>Single-objective Experiments</b>	<b>60</b>
7.1	Introduction . . . . .	60
7.2	Results . . . . .	60
7.3	Conclusion . . . . .	62
<b>8</b>	<b>Multiobjective Optimization Strategy</b>	<b>68</b>
8.1	Introduction . . . . .	68
8.2	Comparing Summed Rank and Pareto . . . . .	69
8.3	Results . . . . .	70
8.4	Examination of Visual Results . . . . .	73
8.5	Conclusions . . . . .	76
<b>9</b>	<b>Multi-objective with Entropy and DFN</b>	<b>77</b>
9.1	Introduction . . . . .	77
9.2	Results . . . . .	78
9.3	Conclusions . . . . .	84
<b>10</b>	<b>Miscellaneous Runs</b>	<b>86</b>
10.1	Introduction . . . . .	86
10.2	New DFN Target . . . . .	87
10.3	Organic Forms . . . . .	90
10.4	City Layouts . . . . .	92
10.5	More Results and Conclusions . . . . .	95
<b>11</b>	<b>Human-centered Study of Aesthetic Measures</b>	<b>98</b>
11.1	Introduction . . . . .	98
11.2	Results . . . . .	100
11.3	Conclusions and Discussion . . . . .	101
<b>12</b>	<b>Conclusions and Future Work</b>	<b>103</b>
12.1	Conclusions . . . . .	103
12.2	Future Work . . . . .	104
	<b>Bibliography</b>	<b>106</b>
<b>A</b>	<b>Miscellaneous Analysis and Results</b>	<b>111</b>

# List of Tables

2.1	Example production rules for an expression . . . . .	6
4.1	GP types used in the language . . . . .	28
4.2	Function set for the GP Language . . . . .	28
4.3	L-system Alphabet . . . . .	34
5.1	Fitness Functions and their appropriate L-System grammars . . . . .	39
5.2	Correlation between several aesthetic-based fitness functions, calculated using approximately 200 pre-existing models . . . . .	51
6.1	GP And L-System Parameters . . . . .	53
6.2	Results of encoding experiment from first and final generation of GP . . . . .	56
8.1	Best and average fitnesses across all runs for the MO experiment using DFN, complexity and symmetry as aesthetic measurements, as well as standard deviation. (1) uses summed rank, (2) uses Pareto, (3) uses random search. Best results are in boldface.	75
A.1	Two-tailed T-test statistics comparing the first generation results of both encodings, over 10 runs. One test is applied for unused production rules and one for complexity.	111
A.2	Two-tailed T-test statistics comparing the first generation results of both encodings, over 10 runs. One test is applied for the percentage of invalid individuals and one for unchanged starting strings. . . . .	111
A.3	Two-tailed T-test statistics comparing the average fitnesses for Pareto and Summed Rank from the final generation, for each fitness function. . . . .	112
A.4	Raw results from survey participants. A value of 0 is an incorrect guess, and a value of 1 is a correct guess. The totals for each model test is shown in the bottom row, and the totals for each participant is shown in the last column . . . . .	113

# List of Figures

2.1	Three stages of a plant-based L-system, using an increasing number of iterations. . .	8
2.2	Examples of voxel art, with single voxel selected . . . . .	9
2.3	Basic generational GP algorithm . . . . .	11
2.4	Example GP chromosome representation and translation . . . . .	12
2.5	Demonstration of subtree crossover with two example chromosomes. . . . .	13
4.1	System processing loop . . . . .	24
4.2	General GP tree representation . . . . .	29
4.3	Sample chromosome with lists and ERCs evaluated, and final L-system produced . .	32
5.1	Two examples of DFN curves and respective actual curves . . . . .	42
5.2	Two examples of $1/f$ curves and respective actual curves . . . . .	43
5.3	2D image division for symmetry calculation . . . . .	45
5.4	Distribution of DFN and $1/f$ fitness scores calculated for approximately 200 pre-existing models . . . . .	47
5.5	Distribution of entropy and symmetry values calculated for approximately 200 pre-existing models . . . . .	48
5.6	Distribution of mean and standard deviation values calculated for approximately 200 pre-existing models . . . . .	49
5.7	Samples taken from the set of pre-existing models, as well as their scores in six categories	50
6.1	Percentage of unused production rules from population of L-systems per generation, between two L-system encodings . . . . .	58
6.2	Average L-system complexity of population per generation, between two L-system encodings . . . . .	58
6.3	Percentage of invalid models from population per generation, between two L-system encodings . . . . .	59



6.4	Percentage of unchanged starting strings from population of L-systems per generation, between two L-system encodings . . . . .	59
7.1	Single-objective runs of DFN (target 0) and 1/f noise (target 0), averaged over 10 runs	63
7.2	Single-objective runs of surface area (target 52500) and dimension (target dimension [40,120,80]), averaged over 10 runs . . . . .	63
7.3	Single-objective runs of entropy (maximize) and symmetry (target 1), averaged over 10 runs . . . . .	64
7.4	Single-objective runs of unique normals (maximize) and volume (target 4500), averaged over 10 runs . . . . .	64
7.5	Single-objective runs of mean (target 0.025) and standard deviation (target 0.25), averaged over 10 runs . . . . .	65
7.6	Single-objective runs of complexity (maximize), averaged over 10 runs . . . . .	65
7.7	Rendering of models with highest fitness for the model constraint fitness functions and aesthetic fitness functions . . . . .	66
7.8	Rendering of models with highest fitness for the distribution-based fitness functions .	67
8.1	Average per-generation population DFN scores for summed rank and Pareto, with target shown . . . . .	71
8.2	Average per-generation population Complexity scores for summed rank and Pareto, with target shown . . . . .	71
8.3	Average per-generation population Symmetry scores for summed rank and Pareto, with target shown . . . . .	72
8.4	Range bars for fitness functions, showing min, max, average and quartiles 1 and 3. Columns represent (1) summed rank, (2) summed rank without duplicates, (3) Pareto, (4) Pareto without duplicates, (5) random search, (6) random search without duplicates.	73
8.5	Sample models chosen from the top ten individuals of various runs. Shown with L-system and fitnesses (DFN, complexity and symmetry) . . . . .	74
8.6	Sample models chosen from a set of rank 0 individuals using Pareto ranking. Shown with fitnesses (DFN, complexity and symmetry) . . . . .	75
9.1	Per-generation results of high DFN versus high entropy fitness targets . . . . .	79
9.2	Per-generation results of low DFN versus low entropy fitness targets . . . . .	79
9.3	Per-generation results of high DFN versus low entropy fitness targets . . . . .	80
9.4	Per-generation results of low DFN versus high entropy fitness targets . . . . .	80
9.5	Distribution of final population for 10 runs, of high DFN versus high entropy fitness targets . . . . .	81

9.6	Distribution of final population for 10 runs, of low DFN versus low entropy fitness targets . . . . .	81
9.7	Distribution of final population for 10 runs, of high DFN versus low entropy fitness targets . . . . .	82
9.8	Distribution of final population for 10 runs, of low DFN versus high entropy fitness targets . . . . .	82
9.9	Sample models from each of the four tests, with corresponding fitnesses (DFN / entropy)	83
10.1	Rendering of low-DFN models using no texturing or smoothing techniques . . . . .	88
10.2	Rendering of low-DFN models using texturing and no smoothing techniques . . . . .	88
10.3	Rendering of low-DFN models using smoothing techniques and no texturing . . . . .	89
10.4	Rendering of low-DFN models using texturing and smoothing techniques . . . . .	89
10.5	Hand-picked results of two separate runs using a DFN target of 3.5, and rendered with textures and smoothing techniques in Blender . . . . .	90
10.6	Hand-picked results of two separate runs aiming to produce organic forms, and rendered with textures and smoothing techniques in Blender . . . . .	91
10.7	Renderings of two models evolved to resemble city layouts . . . . .	93
10.8	More renderings of two models evolved to resemble city layouts . . . . .	94
10.9	Renderings of models using smoothing and texturing, each evolved using separate parameters . . . . .	96
10.10A	high-quality rendering of an ice-like sculpture . . . . .	97
11.1	Example model pair test used in the survey . . . . .	100
11.2	(Left) Histogram displaying number of correct choices, out of 34, for all 20 tests. Horizontal line shows mid-point at 17. (Right) Same as left, except with non-dominating model pairs removed. . . . .	101
A.1	Fitnesses, targets and models for tests 1-5 used in human-oriented survey . . . . .	114
A.2	Fitnesses, targets and models for tests 6-10 used in human-oriented survey . . . . .	115
A.3	Fitnesses, targets and models for tests 11-15 used in human-oriented survey . . . . .	116
A.4	Fitnesses, targets and models for tests 16-20 used in human-oriented survey . . . . .	117

# Chapter 1

## Introduction

The world is filled with three dimensional structures. Cities are virtually packed with architectural wonders – ranging from small and simple to enormous and complex – designed by some phantom architect with an intent to produce something beautiful yet practical. Homes are full of objects of varying size, form and purpose, and even nature itself beckons people in droves every day to bear witness to the marvelous forms it offers. In fact, people are so used to the presence of these forms that they rarely take the time to notice the meticulous design steps required to produce such artifacts. Each individual form takes time and experience to produce – a job left to skilled architects and designers. In producing their works, they must of course take into account two factors – purpose and aesthetic value. Of course, aesthetic value is a highly subjective notion.

A perfect example of this design process exists in architectural design. A modern home is not simply built on a whim. It must first be constructed mentally by a designer, painstakingly drafted onto paper, while taking every flaw and minor imperfection into consideration for the final product. The success of this product relies on its functionality, how it will react to the natural physics of the world, how it satisfies its own purpose and so on. Its aesthetic appeal is also paramount to its success, as people tend to take notice of the beauty of each structure and room as they enter. It is because of these facts that the transition from a rough draft to a final product is a long process, requiring the expertise of many parties. An architect is not always an artist.

As computational resources such as speed and memory become cheaper and more widely available, the computer is now useful for simplifying the most difficult and painstaking of tasks. It has more recently allowed designers to visualize their works in manipulatable environments, which has had quite an impact on the design community. Programs like Maya, Corel Draw and Blender facilitate this purpose and are used both commercially and academically. The use of these applications is not without problems, as the process of generating useful 3D structures or models can be painstaking and long despite the efforts made by the computer to simplify it. At the time of

writing of this thesis, there is very little in the field of automated structure design. Even rarer is the application that can generate aesthetic forms automatically, without the need for user guidance. Such an application would indeed be useful to the design community as a tool for inspiration and design exploration.

This thesis introduces a means of producing 3D models automatically, using evolutionary computation. The approach aims to relieve the task of manual 3D modelling, which can be difficult and time-consuming. It also presents an interesting research topic, involving computational aesthetics applied to 3D models. The models that are generated must fulfill certain aesthetic and geometric requirements imposed by the user, which enable the production of more specific, constrained forms. For example, a user may wish to generate small, complex models as a possible inspiration for chandelier design, and may do so by tweaking the various fitness function targets and constraints offered by the system.

In order to produce such models, the system employs a fractal-based drawing technique which uses the concepts behind L-systems to generate self-repeating forms. The L-systems discussed generate an evaluation string which in turn can be parsed using several different drawing grammars to generate a model in a 3D voxel environment. The models are evolved towards a target fitness, where the fitness function(s) used quantifiably reflects the model's geometric and aesthetic properties. The geometric measures include dimensional boundaries, volume, surface area and the number of unique surface normals, where some aesthetic measures focus on complexity, symmetry and model distribution data. The targets that are chosen for each function are problem-specific, and so will differ for each experiment.

For aesthetic fitness functions, this thesis will explore several existing quantifiable aesthetic measures which are applied to 2D image analysis, and investigate their applicability and extension to a 3D problem domain. The majority of 2D aesthetic measurements measure image properties, such as color changes across the image and form measurements. Their 3D extensions will measure properties attributed to 3D models such as surface shape, structure and size. Certain properties are shared between 2D images and 3D models as well, such as symmetry and complexity.

The proposed approach has many real-world applications and contributions. The models generated can be used as an inspirational tool for designers in any field, producing a diverse population of potential candidates which in turn can be manipulated and altered by a designer. The extensions made to the aesthetic measurements from 2D to 3D may offer insight into future research possibilities, as the field of computational aesthetics is still in its early stages, especially in the 3D modeling domain. The discussed system could also be extended in the future to produce more specific forms, which in turn could be used to automatically generate dynamic environments for movies, video games and animations. The problem introduced by automating the production of aesthetic 3D forms investigated in this thesis is a challenging evolutionary design problem, which offers insight into many future applications, fields and studies.

## 1.1 Goals

### 1.1.1 Applicability Evolutionary Computation to Problem

A general goal is to examine the effectiveness and applicability of an evolutionary computation technique to the design and production of aesthetic three dimensional forms. In order to justify the use of genetic programming for this problem domain, a variety of issues are considered:

- The effectiveness of the genetic programming technique in producing models which satisfy each fitness function individually, using a single-objective approach.
- Different combinations of fitness functions – both aesthetic and geometric – through a multi-objective approach, and a comparison of different multi-objective strategies such as Pareto and Summed Rank.
- The effectiveness of an evolutionary computation approach in producing a diverse population of aesthetic models, useful for design inspiration.

This problem is of interest to many parties, and provides a challenging evolutionary design problem.

### 1.1.2 Extension of Existing Aesthetic Measures

There are many existing aesthetic measures which can quantifiably rank images aesthetically. Although the notion of aesthetic value is subjective, these measures have offered insight into well-known theories of general human aesthetic interest, encompassing ideas of symmetry, complexity, color, shape and form. These theories can be extended and implemented towards an approximate numeric calculation of a form's aesthetic appeal, as seen in existing measures of symmetry, color distribution and complexity in 2D images. Although rare, there are a few examples of systems which have already attempted to use some of these existing measures as fitness functions for evolutionary design problems. The majority of these systems rely heavily on user-guided evolution, however, which is slow and inefficient. This thesis attempts to investigate further into the application of quantifiable aesthetic measures as fitness functions for evolutionary design. More specifically, this thesis:

- Explores the background of computational aesthetics, examining the foundational theories and ideas behind the field and their potential application to 3D evolutionary design.
- Explores the possible extensions of many existing aesthetic measures from 2D images to 3D models through actual implementation and the examination of visual results.
- Examines the success and aesthetic value of models generated using these extensions by various means, including statistical analysis and user feedback.

### 1.1.3 Implement Improved L-System Encoding for Genetic Programming

This thesis examines an existing L-system encoding for genetic programming chromosomes, and offers improvements to areas of the encoding which result in the production of faulty production rules and evaluation strings. This will involve the introduction of new definitions for L-system correctness, completeness and validity. The thesis will present these new improvements to the encoding which will enforce certain constraints on the L-systems produced, as well as speculate upon potential future improvements.

## 1.2 Thesis Structure

This thesis is organized as follows. Chapter 2 provides necessary background information regarding shape grammars and L-systems, and their ability to produce 3D models. It also provides information regarding evolutionary computation, genetic programming and their extensions and uses. Chapter 3 introduces a literature review of the work of others who have used evolutionary computation for design problems and investigated computational aesthetics. The system details, such as the system programming loop, the L-system encoding and decoding process and the model conversion process, are found in Chapter 4. This chapter also presents the L-system alphabets and genetic programming function sets used in this thesis. Chapter 5 introduces the fitness functions used for experiments, as well as information behind each function's purpose and calculation.

Chapters 6 to 11 introduce the five major experiments investigated in this thesis, with each focusing on a different goal. In addition, Chapter 6 introduces the parameter set used for the evolutionary processes in these experiments. Chapter 12 provides a discussion of the results found in this research, and evaluates the effectiveness of the proposed system in achieving its goal. It also outlines potential future research in the field of computational aesthetics and evolutionary design, and possible improvements to the system.

## Chapter 2

# Background Information

This chapter introduces the necessary background required for the complete understanding of the research topic presented. Such topics include shape grammars and L-systems, which are closely related, as well as preliminary information regarding evolutionary computation and its extensions. New measurements of validity and completeness for an L-system are also presented here.

### 2.1 Shape Grammars

A shape grammar is a type of formal grammar that can be used to generate complex geometric shapes or forms using a predefined set of shape manipulation and construction rules, also known as *production rules* [37, 39]. These rules contain a set of shape manipulation, transformation, movement, placement and construction rules, which can be used in conjunction with one another to generate a wide variety of forms. A formal grammar contains a set of rules that can be used to generate a string in a formal language, composed of symbols and sub-strings from a pre-defined alphabet. As an example, a formal grammar can be used to produce a syntactically and semantically correct program from a programming language. For example, the program

$$\text{int } i = 2 * 2$$

follows a simple grammar that makes it syntactically and semantically correct. In this case, we define a variable with a type, then assign a value to that type. This value can be a product of any other rules that return an expression as a value, such as arithmetic operations. In this case, we are using multiplication with two integer values. From the viewpoint of a programmer this is correct, but in order for this to be correct in the context of the language it is written in, it must obey the rules of its production grammar. A simple grammar for the previous expression can be seen in Table 2.1.

The format seen in the example is known as Backus-Naur (BNF) form. With BNF, a grammar is

Table 2.1: Example production rules for an expression

$\langle \textit{statement} \rangle$	$::= \langle \textit{type} \rangle \langle \textit{ident} \rangle = \langle \textit{expr} \rangle$	$\langle \textit{expr} \rangle$	$::= \langle \textit{val} \rangle \langle \textit{op} \rangle \langle \textit{val} \rangle$
$\langle \textit{type} \rangle$	$::= \textit{int} \mid \textit{double} \mid \textit{float}$	$\langle \textit{op} \rangle$	$::= * \mid / \mid + \mid -$
$\langle \textit{ident} \rangle$	$::= \textit{some string}$	$\langle \textit{val} \rangle$	$::= \textit{some real number}$

defined using terminals, non-terminals, a start symbol and a set of production rules. In our sample grammar, the start symbol is  $\langle \textit{statement} \rangle$ , which exists to define what the program is built from, also called an initiator. Non-terminals point to other rules and terminals, such as  $\langle \textit{expr} \rangle$ . Terminals are rules that are no longer evaluated in the grammar – such as  $+$  – and are necessary in building the final program expression. The production rules set defines all the rules encompassing the non-terminals and terminals.

This concept can also be applied to shapes instead of mathematics and programming languages. With the binary operators defined in our previous grammar, each has an explicit purpose. For example, the addition function will add two expressions. This can also apply to other functions as well, such as scaling an image, changing a color or rotating a shape. These are typical functions used in shape grammars. For example,  $\textit{circle} + \textit{circle}$  in a grammar might combine two circles, where each circle is defined separately in the grammar. Another example might be  $\textit{scale}(\textit{circle}, \textit{int})$  which scales a supplied circle by the amount supplied in  $\textit{int}$ .

Typically, with shape grammars, instead of a line of code as an initiator, a shape or form is used. The production rules can then be used to carry out transformations and additions to that shape, creating a brand new shape. In the case of 3D forms, a fully complete grammar that could create any conceivable form would be extremely complex and difficult to manipulate or understand, which is why most shape grammars are made to create specific types of simpler shapes.

## 2.2 Lindenmayer Systems

Lindenmayer Systems – or L-systems [44, 45] – are quite similar to shape grammars. They also define a set of production rules to generate forms, though the intent behind them differs. In an L-system, the goal is to generate self-similar fractal forms. More specifically, the system defines rules for producing a single form, and any alterations to the rules themselves will produce variants of that form.

In this thesis, *D0L-systems* are used (deterministic with no context), as they are the simplest form of L-systems [22]. A D0L-system is defined as  $G = (\Sigma, \omega, P)$ .  $\Sigma$  is the alphabet of the language, where  $\Sigma = \{s_1, s_2, s_3, \dots, s_n\}$  and each  $s_i$  is a symbol within the language. The symbol  $\omega$  is called an *axiom*, and is defined of the set  $\Sigma^*$ . The axiom is also more commonly known as the starting string or initiator.  $P$  defines a mapping  $P : \Sigma \rightarrow \Sigma^*$ , where for all  $s_i \in \Sigma$ , there exists a  $s \rightarrow P(s)$ .



This means that for every symbol, there is exactly one direct mapping to a production rule <sup>1</sup>. Each production rule is of the form  $LHS \rightarrow RHS$  where LHS references a single left-hand side symbol in  $\Sigma$  and RHS contains an ordered sub-expression from  $\Sigma^*$ , which is found in the right-hand side of the production rule. The axiom is altered by the production rules and replaced for each iteration of  $P$ . This sequence of iterations can be defined as

$$\omega^0 \xrightarrow{P^0} \omega^1 \xrightarrow{P^1} \omega^2 \xrightarrow{P^2} \dots \xrightarrow{P^{n-1}} \omega^n \quad (2.1)$$

Each  $P^i$  represents the  $i^{th}$  iteration of  $P$ , and each  $\omega^i$  represents the sub-expression produced from the most recent iteration on  $P$ . Prior to the first iteration,  $\omega^i = \omega$  or simply equals the starting string, and  $\omega^n$  equals the final evaluation string of the L-system. This final string can be used to produce the shape or form intended by systematically parsing the symbols in the string, mapping each symbol to a predefined drawing function. Each drawing function is used to alter a *canvas*, which is the drawing environment. The entire language of the L-system is described by  $L(G) = \{P^i(\omega), i \geq 0\}$ .

As previously mentioned, the evaluation string produced by  $L(G)$  consists of an ordered set of symbols that can be parsed in order to generate a form. One of the most common examples are L-systems used to define plants. These systems use *Turtle Graphics* commands such as Forward (F), Turn (+ and -), and cursor position functions (push '[' and pop ']'). An example of such a system can be seen in Figure 2.1, where each subsequent drawing represents a further iteration on the L-system from the previous. In this example, the starting string used is F, and there is only one production rule  $F \rightarrow FF[+FF][-FF]$ . A 2-iteration run would look like

- **Iteration 0** : Resulting string is starting string F
- **Iteration 1** : F is replaced with  $FF[+FF][-FF]$
- **Iteration 2** : Each occurrence of F in (1) is replaced with  $FF[+FF][-FF]$ , resulting in  $FF[+FF][-FF]FF[+FF][-FF][+FF[+FF][-FF]FF[+FF][-FF]][-FF[+FF][-FF]FF[+FF][-FF]]$

As this example shows, the resulting image grows rapidly in complexity for each iteration, despite the simplicity of the alphabet and rules themselves<sup>2</sup>. The symbols in the alphabet are represented with simple visual commands and can be rendered as a string of single-character commands (such as F and + in Figure 2.1). In addition to those symbols that alter the image in some way, other commands

<sup>1</sup>A D0L-system is context-free as it has only one production rule for each symbol. L-systems that allow more than one production rule per symbol are known as stochastic L-systems. In a stochastic L-system, when a symbol maps to multiple production rules, only one rule is chosen from the set per generation. This is done with some probability or chosen with some cyclic order.

<sup>2</sup>It is important to note that while the images drawn by an L-system grow in complexity for each iteration of the L-system, they in fact grow physically as well. In the example, the character  $F$  results in drawing a line of length 10 on the canvas. When we replace it with a larger string of several  $F$ 's, we can expect that this replacement will cause the resulting image to grow with each iteration. This process is directly analogous to a growing tree.

can be used which are only meant for replacement. As an extension of the previous example, the starting string could be replaced with  $A$ , and a new rule  $A \rightarrow FAF$  could be introduced. In this context, the symbol  $A$  has no visual effect on the image, but is instead replaced at each iteration. The combination of these replacement symbols and drawing symbols (such as  $F$ ) are known as the *variables* of the system. It is common to find only variables on the left-hand side of production rules, and this reasoning is explained later in this thesis.



Figure 2.1: Three stages of a plant-based L-system, using an increasing number of iterations.

Once again referring to the sample Turtle Graphics drawing commands, each occurrence of  $F$  results in drawing a line of some length  $X$ , and each occurrence of  $+$  and  $-$  results in altering the current angle by some value  $\alpha$ . These values are constant in this type of system, unless altered by global modifiers. However, some L-systems allow certain symbols to take arguments as parameters, also known as *parametric L-systems*. In a parametric L-system, symbols are of the form  $s = \{s(p_1, p_2, \dots, p_n)\}$ , where  $p_i$  is the  $i^{\text{th}}$  parameter for that particular symbol. For example, instead of  $F$  relying on a global variable  $X$ , its length can be passed to it as a parameter  $F(10)$ . With parametric L-systems, the evaluation string tends to grow more quickly with the addition of brackets and numbers.

L-systems are of great interest as they have been shown to produce some of the most common natural and man-made structures, a fact that has also been associated with fractals [14]. Both fractals and L-systems have been said to generate more aesthetically pleasing forms as they increase in complexity, and thus possess a form of aesthetic attraction [23]. In fact, fractal forms are very common in architectural designs [38]. This concept provides an interesting question; can structures composed of fractal geometries provide insight into the automatic generation of aesthetic structures? The sheer difficulty in manually designing an L-system to produce a specific form is problematic,

and serves to justify the inclusion of an automated process, created to aid in L-system construction with minimal human supervision.

### 2.2.1 Validity of an L-system

Two new definitions regarding L-systems were created specifically for the purposes of this thesis – *completeness* and *validity*. The *completeness* of an L-system describes how well the system follows the definitions for a D0L-system defined in this section. The *validity* of an L-system describes how functional a system is over many iterations. A valid L-system must have :

1. a non-empty starting string,
2. at least one production rule, with at least one rule used per iteration, altering  $\omega^i$
3. each '[' symbol paired with a ']' symbol, with at least one variable between each set, and
4. the LHS variable of each production rule found in the RHS of at least one other production rule or starting string  $\omega$ .

## 2.3 Voxels

A voxel (volumetric pixel) is a single data element in a 3-dimensional grid of elements, commonly used for the visualization and analysis of medical and scientific data [36]. The concept of a voxel is directly analogous to a pixel in an image – as a pixel is represented as a square data element in a 2-dimensional grid, a voxel is represented as a cube. Typically, a voxel is in one of two states – on or off – and is only made visible in its 'on' state. When rendered, voxel-art produces box-like structures, composed entirely of cubes (see Figure 2.2). These forms are simple to generate and quick to render, as voxels that are hidden from view by other voxels are easy to locate. In addition, calculations such as model symmetry, volume, surface area and dimension are exceedingly simple.

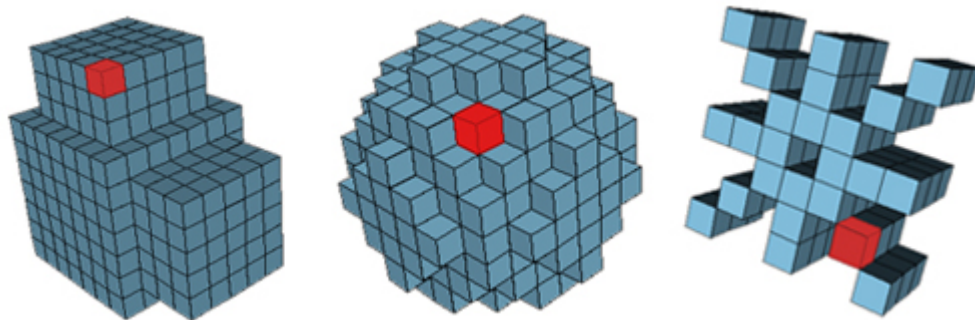


Figure 2.2: Examples of voxel art, with single voxel selected

In order to draw in voxel-space, the system need only 'turn on' necessary voxels in the space. A 3D pen is used in this respect, drawing on the 3-dimensional voxel canvas in the same way a pen draws pixels in a 2D graphic editor. Aspects such as the pen size and orientation can be altered as well. The canvas size for this research project was pre-set to specific dimensions of 128 x 128 x 128, now referred to as the voxels' *bounding box*. This was chosen as such in order to limit the possible size of models created this way (2,097,152 possible voxels must be processed). This size is also sufficient enough to allow enough detail to show in the final rendered model, while still focusing on the general shape and form as a whole. Each voxel represents a single cubed unit of measure.

## 2.4 Evolutionary Computation and Genetic Programming

Evolutionary Computation (EC) is a sub-field of artificial intelligence, which utilizes the foundations of biological evolution in solving difficult problems [12]. There are several extensions and techniques in EC, but the research present in this thesis focuses primarily on Genetic Programming (GP). GP itself uses the basic principles of Darwinian Evolution to evolve a population of individuals based upon each individual's fitness within that population [26, 42, 43]. Simply stated, an individual that is more fit will typically have a higher chance to reproduce – as will its mate – and therefore pass on its genetic material to future generations. This concept is more commonly known as *survival of the fittest*.

### 2.4.1 Generational Algorithm

Each stage of the evolutionary process in GP consists of a number of steps, grouped together in what is called a single *generation*. Before the generational algorithm begins, individuals are randomly generated and inserted into the initial population, and their fitnesses are calculated. Once this is done, individuals are selected in couples from the population based upon their fitnesses, which were computed at the end of the previous generation. At this point, each couple exchanges genetic material, producing offspring who will carry the genetics of the parents to the next generation. Once a new population has been created, their fitnesses are evaluated and the process loops for a determined period of time. This entire process is outlined in Figure 2.3, and each step and its relevance is discussed in further sections.

### 2.4.2 Chromosome Representation

In GP, a population is evolved in stages, and each individual in the population is a program. More specifically, each individual in the population is a representation of a single problem solution, where each solution is represented in a program tree structure called a *chromosome*. Each non-leaf node in the tree – including its root – takes the form of an operator, and each operator's subtrees will evaluate

- 
1. Initialize random or seeded population of GP trees
  2. Pre-processing (optional)
  3. Evaluate initial population using selected fitness function(s)
  4. From 1 to *max\_generations* loop
    - (a) Employ elitism, if appropriate
    - (b) Select 2 individuals from population using chosen selection method
    - (c) Perform crossover on the couple with probability  $P_c$ , producing two offspring
    - (d) Mutate offspring with probability  $P_m$
    - (e) Repeat steps (a – c) until new population is full
    - (f) Replace old population with new population, and evaluate fitnesses
  5. Post-processing (optional)
- 

Figure 2.3: Basic generational GP algorithm

to its operands (arguments). The leaf nodes are terminals, and are typically numbers or functions with no arguments. In strongly-typed GP, each operator requires specific return types for each of its argument subtrees. For example, if the operator at a node is '*AddInteger*', it will require two integers as operands, and return an integer itself after it and its subtrees have been evaluated. Thus, only operators that return integers – or integer terminals – can be used as operands for *AddInteger*. In order to evaluate an individual, the program generated by that individual's program tree is constructed using depth-first search, and then executed to some meaningful end. This decoding step is referred to as genotype-to-phenotype mapping, where the genotype is the GP program tree, and the phenotype is the function or program produced by parsing the tree. An example of a simple GP chromosome representation and evaluation can be seen in Figure 2.4. The list of all possible operands and operators that a GP can use in its chromosome representation and construction is called the *function set*, and is determined prior to the evolutionary cycle of the GP.

### 2.4.3 Initialization

The initial population is created randomly prior to the first generation, consisting of randomly generated tree structures. Parameters are set in place to control the size and shape of the trees produced, in order to ensure that they do not exceed memory limitations. In addition, a typical GP tree grows exponentially with increasing depth, and will therefore take longer to both decode and execute. In some situations, a seeded population is used to guide the evolutionary process by introducing more fit individuals into the initial population. Once the population is created, their fitnesses are calculated during the evaluation phase described in the next section.

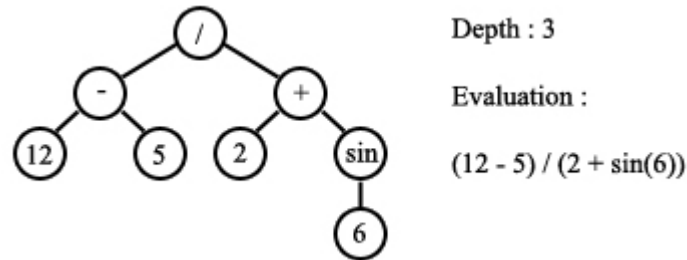


Figure 2.4: Example GP chromosome representation and translation

#### 2.4.4 Solution Evaluation

At the end of each generation – and during the initialization phase – the entire population is evaluated using *fitness functions*. A fitness function is a means of quantitative measurement of an individual’s success within its population with respect to a specific problem. In GP, a fitness function maps a chromosome to a numerical value, representing its ability to solve the current problem at hand. The score returned by the fitness function for each chromosome would be assigned accordingly as a chromosome’s *fitness*, and individuals with higher fitnesses will have a higher chance of successful reproduction. In order for an individual to be evaluated, its tree must first be interpreted. It is important to note that although GP aims to produce an optimal solution, this is usually rare and depends greatly on the difficulty of the problem. In many cases, a near-optimal solution is acceptable. The ultimate goal of the GP run is to converge a population of randomly generated individuals to a more refined population of improved, near-optimal solutions.

#### 2.4.5 Selection

Selection is the process of determining which individuals are granted the opportunity to reproduce. This is accomplished by extracting individuals from the population via probability based upon their fitnesses. The chosen individuals are compared to one another, and those of the highest fitness are given the chance to reproduce. It is generally undesirable to have only the most fit individuals take part in reproduction, as this causes early sub-optimal convergence in the population. Therefore, most selection techniques offer individuals with lower fitness a fighting chance to reproduce as well on occasion, in order to keep the population diverse.

#### 2.4.6 Reproduction – Crossover and Mutation

The reproduction stage consists of two parts – crossover and mutation. Both of these evolutionary techniques are used to move through the *solution space*, which is the set of all possible solutions

that can be generated using the GP function set. GP function sets that are more complex tend to have larger solution spaces, as there are significantly more combinations of the functions for each chromosome. In addition, trees that are allowed to grow large – containing a large number of nodes – will also increase the size of the solution space. Thus, a smaller – yet sufficient – function set is preferred.

Crossover involves taking the two parent chromosomes and exchanging large portions of genetic material, producing offspring in the process. In GP, a common crossover technique is known as *subtree crossover*, which is demonstrated in Figure 2.5. In subtree crossover, two nodes are chosen – one in each parent chromosome – that have the same return type. These two nodes need not be equivalent. The nodes and their subtrees are exchanged between the parents, producing two completely new, yet similar, individuals. Crossover helps to move through the search space quickly, jumping between largely different solutions.

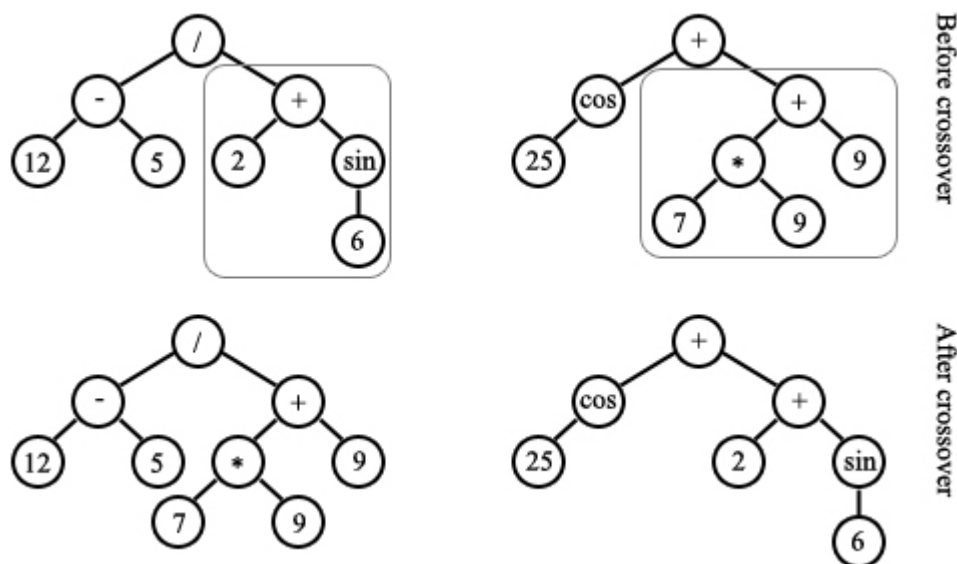


Figure 2.5: Demonstration of subtree crossover with two example chromosomes.

Mutation involves taking the two resulting offspring and altering their trees slightly. This step is meant to introduce new genetic material into the gene pool of the population, preventing convergence and offering new solutions. A common mutation technique in GP is *subtree mutation*, which involves choosing a single node in an individual, and generating a whole new subtree in its place. This is done in a similar manner as the initialization phase. Mutation, unlike crossover, moves through the search space slowly, and is used to ‘fine-tune’ solutions.

### 2.4.7 Elitism

Elitism is used to save the most fit individual from previous generations, in order to preserve its genetic material and increase the chances it will reproduce again. At the end of each generation, prior to the replacement of the previous population, the individual(s) with the highest fitness from that population are set aside and swapped into the new population. This is done by typically replacing the worst individual of the new population. When copying the most fit individual, no modification is done to the chromosome.

## 2.5 Multi-objective Genetic Programming

In EC, a multi-objective optimization (MO) problem requires the optimization of multiple features simultaneously, as opposed to the single-objective approach. Each feature has its own target value, though it is possible for one feature to affect the score of another. The problem introduced by using MO evolution is that a simple ranking method will not work as it does with single-objective evolution – it is not always necessarily clear which individual is better than the other.

### 2.5.1 Pareto Ranking

The most common ranking method for MO evolution is Pareto ranking, which uses the notion of *domination* to discern one individual's rank from another [12]. An individual  $A$  is said to dominate  $B$  if it is superior in at least one feature score, and at least equivalent in all other features. If  $V$  denotes a feature vector  $\vec{V} = (v_1, \dots, v_k)$  and  $A$  and  $B$  are similarly defined, then

$$A \text{ dominates } B \text{ iff } \exists i : a_i < b_i \wedge \forall i : a_i \leq b_i \quad (2.2)$$

This applies to a minimization problem, where each  $v_i$  represents the error between the target and the actual value for a single feature test  $i$ . The population is ranked using the following strategy. All initial individuals that are undominated are assigned a rank of 1, at which point they are *removed* from the current population. Then all the individuals from the current population that are undominated are assigned a rank of 2 and removed. This process continues until all individuals are ranked. The ranks assigned to each individual are then assigned as fitness scores used during the evolutionary search. This strategy works best with low-dimensional problems, and tends to produce outliers which excel in one feature test but fail in all others. Outliers are generally unwanted, as an optimal-scoring individual in MO will excel in all fitness categories.



### 2.5.2 Rank-Sum

Rank-sum is another scoring strategy used in MO problems, which can be applied to problems with higher dimensionality [4]. Consider a search problem with feature vector  $\vec{V} = (f_1, \dots, f_k)$ . For each feature  $f_i$ , the fitness scores of every individual within the population are ranked according to their position with respect to others in the population. Each ordered rank  $r_i$  is assigned to a rank vector  $\vec{R} = (r_1, \dots, r_k)$  for that individual. An optimal score within the population has a rank vector of  $r_i = 1$  for  $1 \leq i \leq k$ . Once the rank vectors of each individual in the population have been calculated, a single weighted summed rank score is assigned to each individual. This value is calculated as  $fit = \sum_i w_i r_i$ , where each weight  $w_i$  is assigned by the user. These weights are entirely optional and default to a value of 1 if not used. This strategy works well for creating a more diverse population of individuals which are better in most fitness categories, and for removing the appearance of high-ranking outliers common with pareto ranking.

## Chapter 3

# Literature Review

This chapter introduces the previous work by other authors in the field of evolutionary computation and its application to image and form design. Existing aesthetic measures, many of which are extended and explained further in this thesis, are introduced here, as well as some of the earliest contributions to computational aesthetics.

### 3.1 Art and Evolution

Evolutionary computation has been used to produce designs and artwork for a variety of purposes – from artwork generated from user-guided evolution to architectural planning and form generation [56]. The works seen in [3, 27, 58] explore the possibilities of using EC to generate a wide variety of images through 2D texture and fractal formulae and other well-known methods. Various implementations of systems which focus on the generation of 3D abstract forms can also be seen in [27], which rely solely on interactive evolution. This lack of a quantitative fitness function is a severe drawback when the amount of time to render a single 3D form and the time it takes to manually assign fitness scores to an entire population is considered.

EC has recently been applied to the field of architectural design, to aid in the design process of structural layouts, architectural construction and organization [25]. Due to the number of factors to be considered when designing a building, the problem of constructing a building using EC is extremely difficult, especially when the practicality and usefulness of the end result is considered. The work in [25] divides these factors into three categories – topology, shape and size optimization. This introduces many potential fitness functions to the problem of general architectural design, such as size, shape, weight, organizational elements and aesthetic appeal. Watanabe used light distribution on window surfaces as a fitness function to evolve the placement of buildings in a city block, the methodology of which is used in actual city design [64]. The IGDT tool described in [60]

was used with a GA to evolve a population of architectural trusses based on a variety of qualitative and quantitative fitness functions, including economic and physical practicality. The aesthetic value of the truss was left to the designer's judgement, guiding the evolution. The central focus of the evolution was the avoidance of the optimal, unaesthetic solution, which was already known and considered flat and uninteresting. The development of a final solution led to the discovery of several new solutions meant to inspire the designer, and the tool existed as a true design aid.

Coia explored the usage of GP and shape grammars to generate buildings according to a user-specified form criteria [8]. The shape grammar used was implemented through the *CityEngine* system, which uses non-recursive procedural modeling techniques to render high-detailed buildings and city layouts [41]. Each individual in the population was scored according to the criteria imposed by the user, which included target form dimensions, maximizing the number of unique surface normals and 2D form-fitting. Garces-perez *et al.* used GP to solve common facility layout design problems, focusing on room placement within the bounds of a supplied manufacturing facility [11]. Their fitness functions focused on the practicality and appropriateness of the design produced, rather than through interactive evolution. A similar approach using GA and a grammar-based floor plan design for residential houses was implemented by Rosenman in [48], where a variety of fitness functions were used to measure the feasibility of the design, such as minimizing room perimeter to area ratio and fulfilling zone requirements. In addition, an interactive element was implemented, allowing a user to assign a separate score reflecting an individual floor plan's aesthetic appeal.

Although EC is not applied, Lipp *et al.* introduced an interactive grammar-editing tool for architectural design which attempted to address some of the same issues EC addresses – namely the difficulty in manually creating and editing a shape grammar [29]. Although this method removed the need for direct grammar editing and focused solely on visual editing and direct feedback, it was completely interactive which may be considered undesirable due to time constraints – especially for very complex grammars. Terzidis explored the applications of procedurally-generated architectural designs through the use of algorithmically-controlled grammars to produce anything from commercial buildings to organic forms [57].

EC has also been used to reproduce and generate models and images of natural phenomena, such as plant and animal lifeforms. In 1991, Karl Sims explored the possibilities of using EC to generate 3D plants using a GA and the concepts behind various plant-generation algorithms [53]. The chromosome of his GA encoded the various parameters required by these algorithms and interactive selection guided the evolutionary process. His work also delved into the generation of 2D images through EC using an expression-based approach, which successfully resulted in complex and interesting images. A GP-approach was used by Watanabe in [64] to evolve the production rules of an L-system designed to produce 3D plants using a Turtle Graphics drawing criteria. As a fitness function, the distribution of light over the leaves of the L-system-generated plants was calculated in an attempt to generate plants which replicated the actual growth patterns of those on

Earth. The evolution of plants has also been implemented and extended in other various works using L-systems, such as the artificially-created garden in [21] and in other applications [37]. Sims used EC to evolve the structure and movements of 3D virtual creatures, simulating their movement and progress in order to assign fitnesses to each individual [54]. Individuals were stored in directed graphs and evolved using a GA. Hornby used parametric L-systems and GP for a similar goal, measuring distance travelled from the creatures' center of mass as a fitness function [18]. The appeal of their work rested on the idea that the individuals in the population would *learn* to move, as natural lifeforms do over time.

Others focused on using EC to produce landscapes, such as Walsh *et al.* who used a GA to evolve a set of parameters for a fractal-based terrain generation API in order to generate lush, aesthetic terrains [63]. Their fitness function was based on concepts derived from Birkhoff's aesthetic measure of order and complexity, measuring the difference between an image's size and its Kolmogorov complexity to represent order, and the image's size to represent complexity. The Kolmogorov complexity was approximated using JPEG compression, and the image used was a snapshot of the terrain. The results of the experiment were analyzed via online survey. The GENR8 system – a Maya plugin created by Hemberg *et al.* – uses EC in evolving 3D surfaces [16]. Each surface is constructed using an organic growth model similar to that of plants, which is implemented in practice using the concepts behind Map L-systems. The GENR8 system evolves a grammar for the Map L-system using interactive evolution, which in turn generates an organic surface intended to aid designers.

EC has been used to generate general objects as well, for design inspiration and potential physical implementation. Hornby used an Age-Layered Population Structure (ALPS) to evolve a series of tables using a complexity metric as a fitness function [19]. His metric is described later in this section. Pang *et al.* introduced an interactive evolutionary technique for modeling 3D fractals, using Hausdorff dimension – which is calculated using the box-counting method – as a fitness function and Iterated Functions Systems to generate the fractal art [40]. The fractals are generated using a GA, rendered onto a 3D coordinate-space using voxels and can be edited interactively at key points during the evolution. The system was used to generate jewelry and light patterns.

## 3.2 Research in Aesthetics

Architectural design is one of the primary applications of the generation of 3D forms. Interactive evolution – though time-consuming – is beneficial when the sheer complexity of aesthetic form is considered. The difficulties in quantifiably justifying the aesthetics of a form are reasonably so due to the subjectivity of the concept of aesthetic appeal. This can be seen when surveying the aesthetic diversity of any city scape or rural town. Even so, many have explored the visible complexities and common qualities of many aesthetic structures [52, 51]. Symmetry, color usage, structure orientation and other factors have been suggested to contribute to a structure's aesthetic appeal.

Fractal patterns – from small facades to the foundations of the building itself – can also be seen in the simplest structures. In fact, many believe that the implementation of fractals in architectural design is a staple to the natural being of humans, who are drawn to the organic fractal forms in nature [9, 23].

The Golden Ratio is one of the oldest and most commonly-known aesthetic measures. Two quantities are said to be in the golden ratio if the ratio of the sum of the quantities to the larger of the two quantities is equivalent to the ratio of the larger to the smaller. Artists, sculptors and architects since the Renaissance have attempted to proportion their work to approximate this concept. Birkhoff later conceptualized a quantitative measurement for aesthetics in his measurement of vases [7]. In 1928, he based the human perception of aesthetics on two criteria, order and complexity. Order  $O$  related to the geometrical relationships within an object – such as symmetry – and complexity  $C$  related to the visual stimulus of the object in relation to the level of attention each detail requires. He believed that complexity negatively impacts upon the aesthetics of an object, and his overall aesthetic measure was  $M = O/C$ . He applied this concept to a class of ancient Chinese vases, measuring the perpendicular, tangent, vertical and horizontal order of specialized points along the outline of a particular vase, and complexity was measured as the number of these points. His preliminary findings inspired future work in the field of computational aesthetics.

The symmetry of an object is another well-known aesthetic measure. A large number of man-made objects, as well as of those in nature, are symmetrical in shape and form. The measurement of symmetry is simpler for a 2D image, and a variety of ways to calculate it have been proposed. Lipson *et al.* defined a symmetrical body as “when it can be divided into parts that are related to each other in certain ways. The operation of transferring one part to the position of a symmetrically related part is termed a symmetry operation, the result of which is to leave the final state of the body indistinguishable from its original state” [30]. These operations – in 2D – are mirror, rotation and glide. In 3D, new operations are added – inversion centers, screw axis and mirror planes – which make the problem of calculating symmetry much more difficult. Gunlu *et al.* produced a 2D symmetry calculation using DCT coefficients [13]. They tested this calculation on a series of 2D images of human faces. Kazhdan *et al.* produced a measure of symmetry for 3D models called a Reflective Symmetry Descriptor (RSD), which represents a measure of the reflective symmetry of the model for all planes through its center of mass [24]. The model is converted to a voxelized version and its reflection is computed, at which point 3D planes are passed through the model’s center of mass. The differences between the original form and its reflection are calculated and a spherical function which describes the model’s symmetries is generated. This method is able to ignore most noise in the model, which takes the form of small details and textures. The reflective symmetry measure used in his RSD is also commonly used to calculate symmetry in a 2D image.

Entropy is a measure of the uncertainty that is associated with a random variable [2]. This can

be calculated using Shannon entropy, which is calculated using

$$H(X) = - \sum_{x \in X} p(x) \log p(x) \quad (3.1)$$

where  $X$  is a distribution of random variables. The level of uncertainty is directly associated with the value of  $H(X)$ . Rigau *et al.* observed the similarities between Birkhoff's aesthetic measure when using Shannon entropy to approximate order, and Kolmogorov complexity to approximate complexity [47]. Although subjective, the entropy of an object may be directly related to its visual complexity, and therefore aesthetic appeal.

Kolmogorov complexity is a direct measure of the computational resources that are needed to correctly represent an object [28], which is also commonly known as the Algorithmic Information Content (AIC). More specifically, it defines the minimal number of resources that a data set can be represented with. One of the primary issues with this measure is its incomputability – there is no formal definition for the function of complexity described. Many approximations exist, which are tailored specifically to each applicable problem. For example, JPEG compression has been used to approximate the Kolmogorov complexity of images, using highest-quality compression techniques [63]. Repeated patterns, whether in images or written messages, are considered simple and this is reflected directly in the compression-method for calculating complexity.

Hornby examined a measure of structure and organization by combining his proposed measures of modularity (M), reuse (R) and hierarchy (H), which could be used as a measure of complexity [19]. He compared his metrics to complexity metrics that are well known, such as AIC, grammar size, tree complexity, and number of build symbols in a grammar. He proposed a final measure of complexity, which could be calculated as a measure of structure and organization

$$SO = \sqrt{M^2 + R^2 + H^2} \text{ or } SO = M + R + H \quad (3.2)$$

Machado *et al.* proposed that an image's aesthetic appeal is based on a relationship between the complexity of an image and the difficulty in processing an image [33]. The image's visual complexity (IC) is based on the concept that humans favor unpredictability in images, while the image's processing difficulty – or complexity – (PC) is based on the human mind's ability to easily process simpler images, causing preference for these images subconsciously. Machado uses a fractal image in comparison, which can be generated using a simple mathematical model, yet appears increasingly complex for each level of detail. An image is then considered aesthetic if IC is high, while PC is low, and the aesthetic measure is then computed as

$$M(I) = IC(I)/PC(I) \quad (3.3)$$

for an image  $I$ . The IC value was then estimated as the amount of effort expended in compressing a

2D image, using a ratio of the visual difference between the compressed and uncompressed images, and the difference in file sizes of the compressed and uncompressed images. The PC value was estimated using a means of fractal image compression known as the Box-counting method, which is considered another measure of complexity commonly used for calculating the fractal dimension of a self-similar fractal image. This method involves splitting an image into a grid of increasing resolution in areas where change in the color of surrounding pixels occurs. The more an image is divided, the higher its fractal dimension.

The fractal dimension of an image is in itself a complexity measure, measuring the scaling between patterns at different magnifications. Larger values are more complex, while smaller values are visually simpler. Spehar et al investigated the relationships between the human preference of images with varying fractal dimension [55]. Images were chosen from natural, mathematical and human-made fractals, and it was discovered that images in the fractal dimension range of 1.3 to 1.5 were preferred over others. For Machado's work, a value of 1.35 was used as a target score for the PC value.

Exploration has been done in finding the similarities between classical music and fractal geometry [17, 20]. Hsu *et al.* examined the frequency of incidence of the note intervals of many classical songs, examining their similarities on a log-log plot. Interestingly, the plots are similar to those generated by distributions exhibiting the properties of  $1/f$  noise.  $1/f$  noise – or pink noise – refers to a distribution of signals whose power spectral density is inversely proportional to the frequency. The noise is an intermediate between white noise ( $1/f^0$ ) and red noise ( $1/f^2$ ). Pink noise is of the form  $S(f) = 1/f^\alpha$  where  $0 < \alpha < 2$ . An exceptional amount of research has been done with respect to  $1/f$  noise, beginning with work by Voss and Clarke in 1976 with their research examining  $1/f$  noise seen in voltages across semiconductors [61]. It has been found to be common in natural phenomena such as earthquakes and unnatural such as music [34]. The presence of  $1/f$  noise is often considered to be a universal phenomena, which makes it of great interest for research potential. Its applicability to aesthetics has been focused thus far on human cognition, speech and music.

A model of aesthetics was proposed by Ralph which measures the distribution of color gradient in a 2D image, and fits it to a normal distribution [46]. The value produced is known as the Deviation from Normality (DFN). To accomplish this, the color gradient is first computed across the image – for each RGB color channel – then the mean and standard deviation of the data is calculated. Using this information, the distribution can be estimated and constructed as a histogram, where it is then compared to the actual histogram generated from the gradients of the image. The DFN value is calculated as

$$DFN = 1000 \sum p_i \log\left(\frac{p_i}{q_i}\right) \quad (3.4)$$

where  $p_i$  is the observed probability in bin  $i$  of the histogram, and  $q_i$  is the expected probability assuming a normal distribution, using the data calculated above. Using this formula, a DFN of 0 indicates a perfect fit to the normal distribution curve. This measure has been used in previous

research in the evolution and production of images and image filters [6, 35, 49]. These previous works used GP principles to evolve a set of images and filters encoded as formulae in the GP chromosome, using the DFN as a fitness function, as well as the mean and standard deviations of the distribution. Color palette matching was also used to guide in the evolution of images whose color palette fit within the color quantized bins of a source image's color histogram. Previous research done by Ralph showed that common values of  $DFN = 0$ ,  $\mu = 3.3$  and  $\sigma^2 = 0.75$  were commonly found in Impressionist masterpieces.

The majority of the aesthetic measures available at the time of writing of this thesis are primarily applicable to 2D images. These functions tend to fall into one of two categories – distribution- and complexity-based [15]. Despite the wide variety of these aesthetic measures – both in 2D and 3D – the research is very subjective and still in its early stages of life, opening many possibilities for future research.



## Chapter 4

# Tree Encoding and Model Generation Process

This chapter explains in detail the most crucial elements of the system. To ensure the full understanding of the system's entire process by the reader, the following sections were put together, explaining the reasoning and solutions behind each implementation decision. There were many goals that were considered prior to the creation of this system, which influenced its design. The major goals include:

- The system must be able to produce 3D models, which in turn are generated from the evaluation string of an L-system.
- The L-system of a model must be encodable within a GP chromosome, and must be general enough to allow the construction of any possible L-system, yet constrained enough to reduce the number of invalid L-systems generated.
- The 3D models must permit accurate geometric analysis.
- The system itself must run efficiently, taking runtime into consideration.

The entire system runs through JNetic, an EC-based software meant to ease the manual workload of the user [5]. JNetic used a GA to evolve a population of vector images, using direct color distance matching between an individual and a target image as a fitness function. This system was extended heavily for this thesis and as a side-effect, the complexity of the system grew substantially, especially regarding the system loop. The GA was swapped out and replaced with a GP powered by Sean Luke's ECJ, a Java-based EC package [32]. The ECJ system offered extensive control to the programmer, giving full access to the function set and GP parameters. For the purposes of the research present in this thesis, the JNetic system was further modified to incorporate new extensions to ECJ's GP

back end, as well as new interface elements required for the manipulation of the models generated during each run. The entire system, its extensions and the ECJ back end is written in Java and *Java Swing*, with the 3D modeling handled by Java 3D using *Java View*.

## 4.1 System Loop

A significant amount of processing is done in the system during the course of a single GP run, as well as pre- and post-processing of the models, files and parameters used to guide and influence the evolutionary process. This process is largely step-by-step, and is explained generally in Figure 4.1. This chapter describes the majority of these steps in greater detail, but a general walkthrough of the evolutionary algorithm and model conversion process is described here.

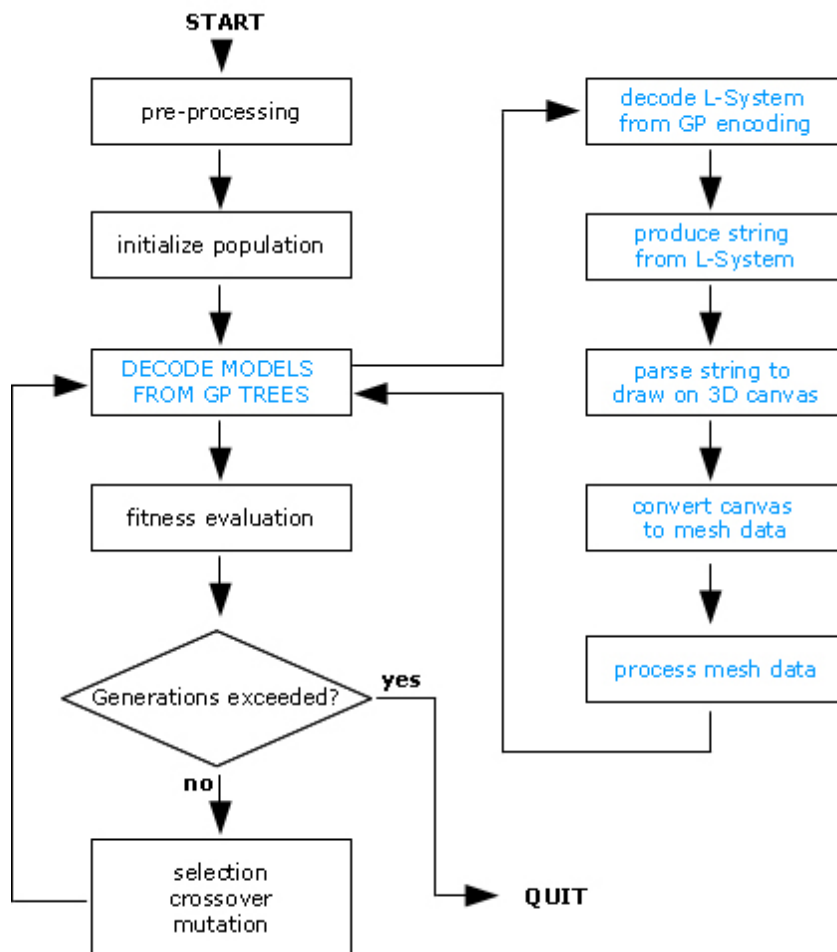


Figure 4.1: System processing loop

First, before any model processing or evolution takes place, parameter files are generated containing the GP parameters required by ECJ in the pre-processing stage. In addition, all L-system parameters and constraints are calculated and stored, such as the range of iterations and L-system alphabet. Once this step is complete, the initial population is generated randomly using a standard GP initialization mechanism. At this point, the GP loop begins and executes until a user-specified number of generations elapses.

Before the population can be evaluated, individuals must be converted into models from their GP tree representations during the decoding phase. This is done for each individual in the population. The GP tree uses a grammar-like encoding which stores a complete L-system definition, including the number of iterations, the starting string and a set of production rules. Once the L-system is decoded, it is processed for the number of generations decoded from the tree, producing an evaluation string of symbols from the L-system alphabet. This string is used to generate a model in 3D, by parsing it iteratively one symbol at a time and mapping each symbol to a pre-defined Turtle drawing function. The model produced from this parsing exists as a voxel volume, which is further processed into a 3D mesh consisting of a set of vertices, edges and face data. Final processing is done on this data and assigned to the respective chromosome that produced it, signaling the GP process that the chromosome is ready for fitness evaluation.

The fitness evaluation stage processes each chromosome individually, assigning a score for each fitness function chosen by the user for the current GP run. Once this has been done, each chromosome is ranked within the population, and the standard GP evolutionary operators are executed, producing a new population. The GP loop then continues back at the decoding phase. Although each step required for the decoding process seems simple in detail, the amount of processing behind each step is significant and tends to grow rapidly with increases in L-system complexity. Each of the steps in the decoding process is described in greater detail in the following sections, as well as the intentions and reasoning behind each one.

## 4.2 Genetic Programming and Tree Encoding in System

### 4.2.1 L-system Encoding Requirements

The choice of the GP function set used to encode an L-system definition within a chromosome is paramount to the success of this thesis, and is influenced by a number of factors. More specifically, a complete L-system encoding must follow all aspects of the DOL-system definition, which includes the starting string  $\omega$ , number of iterations, and a complete set of production rules. This concept was explained previously in Section 2.2, introducing the concepts of completeness and validity. The encoding should ensure the validity of produced L-systems – meaning that all production rules must map from symbols in  $\Sigma$  to existing substrings within  $\Sigma^*$ , and each symbol with a production rule

mapping must be present in at least one other production rule's RHS or in  $\omega$  in order to guarantee its use. This also implies that  $\omega \neq \omega^1 \neq \omega^2 \neq \dots \neq \omega^n$ . In order to guarantee the increase in structural complexity common in fractals for increasing iterations, the definition  $|\omega^i| < |\omega^j| \forall (i < j)$  should be followed, where  $|\omega^i|$  is the magnitude or length of the string  $\omega^i$ .

There is a fault in this definition, however, as L-system structural complexity does not always translate into output string complexity. For example, the strings  $s^a = \{F + + + + F\}$  and  $s^b = \{FFF + FFF\}$  are both of magnitude seven, but  $s^a$  only physically alters the image twice using  $F$ , where  $s^b$  alters it six times. This side-effect is directly linked to the production rules of the L-system. To further this example, the production rule  $F \rightarrow F + + + +$  has no effect on the complexity of the previous string, since  $\nu(\text{LHS}) = \nu(\text{RHS})$ , where  $\nu(s)$  is the number of variables present in a string  $s$ . Therefore, the production rule  $FF \rightarrow F+$  *always reduces* the complexity of the previous iteration's evaluation string.

The definition is now extended to  $\nu(\omega) < \nu(\omega^1) < \nu(\omega^2) < \dots < \nu(\omega^n)$  and for each production rule,  $\nu(\text{LHS}) < \nu(\text{RHS})$ . This will ensure that the complexity of  $\omega^i$  increases for each iteration with respect to the final rendered result. These definitions impose a decidedly necessary constraint on the type of L-systems generated, which in turn limit the generality of possible results. In addition to these general rules, a few more specific rules are outlined here.

### Push and Pop Symbols

The push and pop symbols ('[' and ']') are used to store the current global state of the system and push it to the top of a FIFO (first-in first-out) stack. With respect to a Turtle Graphics environment – as is used in this thesis and explained in Section 4.4 – this stores the current cursor coordinates (x, y, z), and global variable values (sizes and orientations). In order to ensure the validity of an L-system, each '[' must have a matching ']', and so the number of '['s and ']'s in a string must be equal. This property applies to the starting string  $\omega$ , each production rule, and each subsequent  $\omega^i$ . Since the push/pop symbols store and restore global states, they are only useful if the canvas is altered after the initial push and before its paired pop symbol. For example, since the string  $\{F[+ + +]F\}$  does not alter the image between the push/pop symbols, it is equivalent to  $\{FF\}$ . Therefore, there should be *at least* one variable between [ and ].

### Use of Variables in Production Rules

As mentioned in Section 2.2, every symbol in  $\Sigma$  has a mapping in  $P$ , but some only map to themselves:  $s \rightarrow P(s)$  where  $P(s) = s$ . These symbols are typically non-variables that do not alter the final image. To ensure that the system is context-free, variables may only be associated with a single production rule mapping (LHS). Variables that map to  $\Sigma^*$  must also be present in the RHS of at least one production rule, otherwise their mappings are never used. Finally, there must be at least one  $s \rightarrow P(s)$  where  $s \in \omega$ . If not,  $\omega$  will never be altered by the production rules of the L-system.

### L-system String Limitations

An L-system's evaluation string grows exponentially with each iteration, even with simple alphabets and production rules. In order to decrease the time necessary to parse an L-system's final evaluation string, limitations should be made on how fast a string can grow and by how much. This can be done by limiting the size of the RHS of each production rule, as well as the size of  $\omega$ . In the GP tree, this can be accomplished by limiting the size the tree can grow, or taking a substring of all strings ( $\omega$  and RHS) that exceed a preset capacity.

### 4.2.2 Function Set

When choosing a function set for the GP language, there are a number of factors that must be considered. First, the complexity of the set dictates the size of the solution space, where a more complex function set denotes a larger solution space to search. A simpler function set reduces the size of the search space, but severely limits the GP system's capacity to produce a wider variety of meaningful solutions. When designing the function set, it is also important to reduce the amount of undesirable influence the set has over the GP's evolution. Choosing simpler functions may result in a failure of the GP's ability to produce a meaningful solution, while choosing a wide variety of functions may be unnecessary to the solution. Thus, an understanding of the problem is an asset. It is also important to avoid the introduction of functions that overly bias the search, such as offering too much information to the GP.

Jacob *et al.*[22] produced a method of DOL-system encoding for GP which addressed many of the issues described in the previous section. Minor extensions to this encoding were implemented in the system which enforce the definitions of L-system validity and completeness. The GP types used in this thesis for L-system encoding can be seen in Table 4.1 and the function set can be seen in Table 4.2. A visual representation of the general GP tree can be seen in Figure 4.2.

The encoding presented in this thesis and Jacob's are virtually identical except for a few major differences. In Jacob's encoding, stacks are used to represent expressions from the L-system alphabet for the RHS and starting string, which can grow to any size. The LHS of each production rule uses a single symbol from the alphabet, instead of choosing only from variables. In addition, there are no restrictions on variable use for the RHS of production rules in his encoding, as any combination of symbols can be used. This leads to the problem of unused production rules. In order to encourage the production of valid L-systems, Jacob uses custom GP operators which favor valid L-systems and production rules during crossover and mutation. These differences cause Jacob's encoding to produce more invalid L-systems during the GP initialization, which is an issue addressed in the new encoding.

Table 4.1: GP types used in the language

Representation	Description
<b>Atomic Types</b>	
NIL	Null identifier, required by ECJ
N	Integer representing the number of iterations
T	Tree root type, containing starting string, learning rules and iterations
L	List of characters (string)
C	Single character
R	Learning rule of the L-system
<b>Subtypes of L</b>	
$L_{list}$	A general ordered list of symbols in the L-system alphabet
$L_{start}$	A list of symbols, used for the initiator
$L_{RHS}$	A list of symbols, present on the right-hand side of a learning rule
<b>Subtypes of C</b>	
$C_{sym}$	A single symbol/variable existing in the L-system alphabet
$C_{var}$	A single variable existing in the L-system alphabet
<b>Subtypes of R</b>	
$R_{rule}$	Learning rule(s)

Table 4.2: Function set for the GP Language

Returns	Function	Description
T	lssystem(N, $L_{start}$ , $R_{rule}$ )	Returns a complete L-system
N	iteration( )	ERC representing iterations between 2 and $N$
$L_{start}$	startS( $L_{list}$ , $C_{var}$ , $L_{list}$ )	Starting string, containing $\geq 1$ variable
$L_{list}$	listT( )	Return list {'C'}
$L_{list}$	listT2( $C_{sym}$ )	Return list {symbol}
$L_{list}$	listBranch( $L_{list}$ , $L_{list}$ )	Symbol list composed of two other lists
$L_{list}$	listSBranch( $C_{sym}$ , $L_{list}$ )	Symbol list composed of symbol + list
$L_{list}$	pushPop( $L_{list}$ , $C_{var}$ , $L_{list}$ )	Symbol list enclosed in '[' ]'
$L_{RHS}$	rhs( $L_{list}$ , $C_{var}$ , $L_{list}$ , $C_{var}$ , $L_{list}$ )	RHS of a learning rule
$C_{sym}$	symbol( )	Single symbol chosen from SYM list as ERC
$C_{var}$	variable( )	Single variable chosen from VAR list as ERC
$R_{rule}$	ruleSingle( $C_{var}$ , $L_{RHS}$ )	Single learning rule
$R_{rule}$	ruleBranch( $C_{var}$ , $L_{RHS}$ , $R_{rule}$ )	Single learning rule with branch

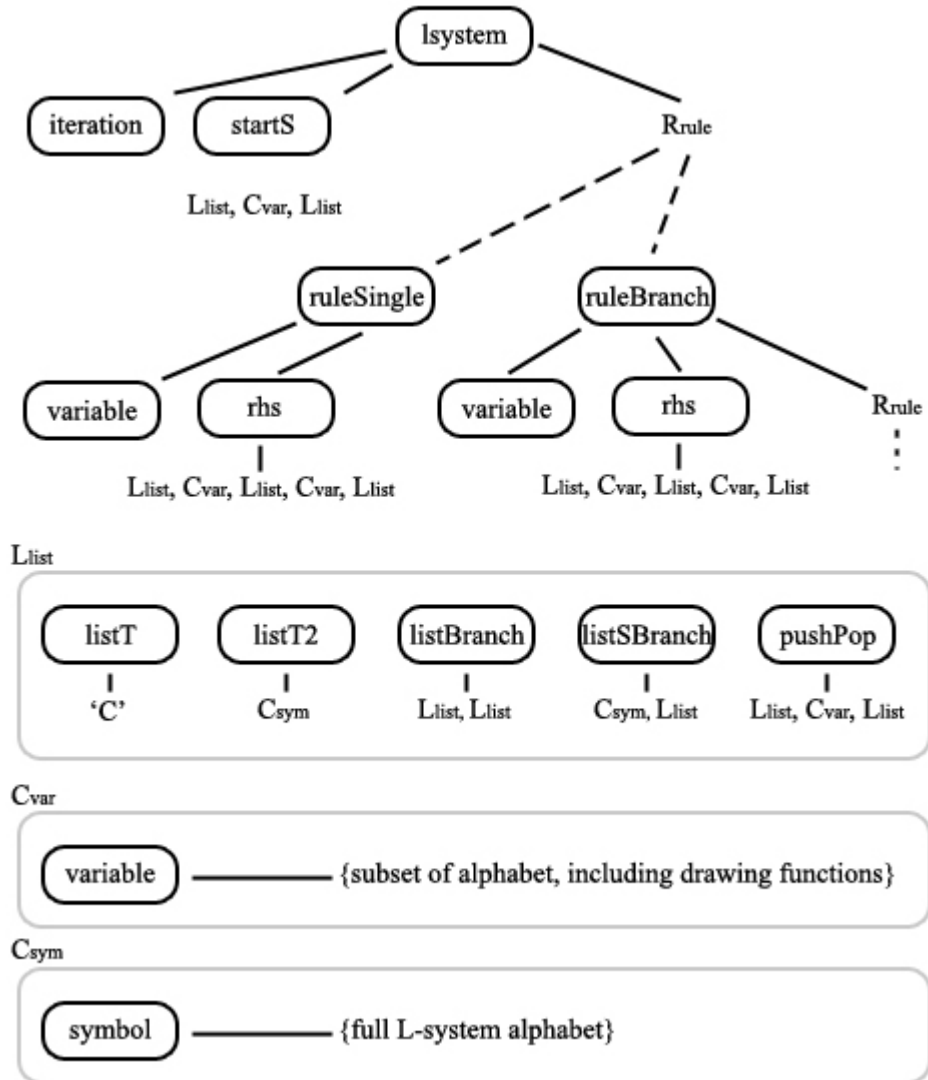


Figure 4.2: General GP tree representation

As with Jacob’s encoding, the root of the new encoding returns an L-system composed of a starting string and one or more production rules. In addition, the number of iterations is also included in the tree as an ERC within a user-specified range, as seen in the function *lsystem*. There are two functions that return the  $R_{rule}$  type – one that returns a single rule, and another that returns a rule followed by one or more learning rule(s). These allow the GP to evolve any number of learning rules, but enforces at least one.  $L_{list}$  types return a list of symbols and variables as a sub-expression, similar to the stack used in Jacob’s encoding. Functions that return C and N types are terminals which actually return an ERC integer within a certain range. The primary differences between the two encodings is not the function set itself, but in the constraints enforced by certain functions. These constraints, their benefits and the remaining function set are explained here, in the order they would be evaluated in the GP tree.

### Preprocessing

Before the GP begins, the L-system alphabet is split into two sub-lists – VAR and SYM. The SYM list contains all symbols from the alphabet except for *push* and *pop*, which are removed completely. The VAR list contains every variable that can be used in the LHS of a production rule. This includes any non-drawing variables such as A and B, and any drawing variables such as C, S and F. Two additional lists are maintained as well during the GP tree decoding phase – *USED* and *CURRENT*. The CURRENT list maintains a list of all the variables used in the starting string and RHS of any production rule which have not yet been used in the LHS of a rule. The USED list maintains a list of all variables used in the LHS of production rules. These lists are updated continually as the L-system is decoded, and are paramount in enforcing valid L-systems.

### Starting String

The starting string of the L-system is processed before the production rules. Since only variables are used in the LHS of production rules, the starting string must contain at least one variable in its expression. The *startS* function in the function set enforces this by requiring a  $C_{var}$  type between two symbol sub-expressions. Once the full expression is generated, each unique variable used in the expression is added to the CURRENT stack. The size of the starting expression is also constrained and cut off after a certain length is attained.

### Production rules

The *variable* function returns an ERC integer between 1 and  $|VAR|$ , which is used to return a variable from the VAR list. For the LHS of a production rule, this number is instead used to choose a variable from the CURRENT list, removing it from CURRENT and adding it to USED. This ensures that the current production rule is guaranteed to be used at least once. If the CURRENT list is empty, then no further production rules are processed. The RHS of a production rule only uses



one function, *rhs*. This function is similar to that of the starting string, but enforces the use of at least two variables instead of one. This guarantees that the complexity of the L-system's evaluation string will always increase with the use of each production rule. Once the RHS expression is built, it is processed and all variables that are in RHS but not in the USED and CURRENT lists are added to the CURRENT list, in order to aid further production rules that have not yet been processed. As with the starting string, the returned RHS expression is constrained in size.

### Push and Pop

The push and pop function *pushPop* uses the same parameters as the starting string function, but instead of returning the expression as-is, it instead returns the expression bounded by '[' and ']'. This was added to ensure that each '[' symbol was paired with a matching ']' symbol, and that every push and pop would bound at least one variable in order to make physical alterations to the final rendered form, prior to restoring the environment settings with ']'. If these symbols were simply left to the SYM list, there would be no constraint on their pairing, and many invalid L-systems would be produced containing them.

L-system validity decomposition is possible – albeit rare – in the new encoding. Crossover, for example, might swap two RHS expressions between trees, removing sub-expressions necessary for certain production rules. Mutation also has the ability to cause this effect, and so certain steps were taken in order to reduce the impact of the GP operators. Validity labeling of L-systems affects the fitness function values of an individual, reducing the chances that an invalid L-system can reproduce. The constraints of the function set described previously also reduce this occurrence dramatically. In fact, the only possibility to generate an invalid L-system during crossover and mutation is to completely remove all variables from the starting string, rendering all production rules useless. This, however, results in no drawing being done to the canvas, and therefore poor fitness values for that individual.

### 4.2.3 Walkthrough Decoding of Sample Chromosome

A sample chromosome can be seen in Figure 4.3, with the alphabet and variable lists shown. First, the SYM and VAR lists are created, containing {A,B,C,+,-,\*,/} and {A,B,C}, respectively. Starting from the root, the number of iterations is evaluated as 3. Next, the starting string is decoded. Since each ERC for variables and symbols refers to an index in the VAR and SYM lists, and the sub-expression is created using depth-first search, the resulting string is -CC+. The variable C is then added to the CURRENT list, prior to decoding the production rules.

There is only one production rule here. First, the LHS variable is evaluated. Though the ERC was generated between 1 and 3 (the length of the VAR list), modulus arithmetic is used between it and the length of the CURRENT list. Since the CURRENT list is only length 1, it will be 1,



### 4.3.1 Voxel-space Drawing Alphabets

Two of the alphabets used in the system use *voxel-space drawing*. The *Voxel-based Model System* (VMS) is the simplest form of voxel drawing of the two alphabets, closest to the Turtle-Graphics method described in the previous section. There are two 3D spaces that are used here – the *voxel space* and the *fractional space*. The voxel space is the 128-cubed volume described earlier, and the fractional space is a 128-cubed volume composed of floating-point measurements. In the voxel space, movement occurs on an integer level (voxel-to-voxel), where in the fractional space the pen can move in real measurements (fractions). As the L-system evaluation string is parsed, the pen is moved accordingly in the fractional space by the set amount. Once it has stopped, its coordinates are converted to voxel-space integer coordinates, and the voxels between the starting and ending points of the pen are turned 'on'. The two spaces are used in order to compute more realistic angles when moving through voxel-space, since movement in a voxel environment only consists of 90-degree angles. The pen initially starts at voxel (64, 64, 64). In the event of the pen moving out of the bounding box, it is either reset to this starting point, or the model is marked as 'invalid'.

The *Surface-based Model System* (SMS) is a variant of the VMS. Instead of moving freely in 3D voxel space, the pen is confined to a 128 x 128 canvas located at the base of the voxel bounding box. As the pen moves across the canvas, the lowest possible unmarked voxel at the pen's (x, y) coordinate is turned on. Therefore, as the pen moves over voxels that it has already marked, voxels higher on the z-dimension are turned on, building structures upwards and creating landscapes. This alphabet was created in order to produce 3D fractal textures. The pen initially starts at voxel (64, 64, 0), and returns to this point in the event that it moves out of bounds.

### 4.3.2 Plant-based Alphabet

The *Plant-based Model Language* (PML) is directly related to the L-system alphabets used by Lindemayer *et al.*[45]. Using simple Turtle Graphics drawing functions, lines are drawn in fractional space at different angles and lengths. This alphabet was added in order to test the capabilities of the system in producing fractal-like forms in addition to plants, such as the sierpinski triangle.

### 4.3.3 L-system Alphabet

Both voxel-based systems have similar alphabets, as can be seen in Table 4.3. The variables are used for drawing on the canvas, as well as for replacement in the L-system production rules. The majority of the alphabet is used to manipulate the state of the Turtle – such as angle and position – as well as the global state – such as angle modifiers and pen size. In addition, the voxel-based systems use more unique modifiers called *gravity wells*, which are explained in detail in Section 4.4.3. The column labeled PML in Table 4.3 shows whether certain symbols pertain to the PML as well, or are simply confined to use with the voxel-based systems. For all three systems, each symbol used

may not necessarily be treated in the same way. For example, in the PML the symbol F is used to draw a line, where the other two systems use F to move forward while drawing nothing.

Table 4.3: L-system Alphabet

Character	Description	PML
<b>Variables</b>		
A	Replacement character with no drawing potential	✓
B	Replacement character with no drawing potential	✓
F	Move forward, draw nothing	✓
C	Move forward, draw a cube of global width/height/depth	
S	Move forward, draw a sphere of global radius	
<b>State Manipulators</b>		
+ and -	Rotate position on the X-axis	✓
* and /	Rotate position on the Y-axis	✓
@ and &	Rotate position on the Z-axis	✓
[ and ]	Push/pop global coordinates and state	✓
W and w	Increase/decrease global width	
H and h	Increase/decrease global height	
D and d	Increase/decrease global depth	
U and u	Increase/decrease width/height/depth simultaneously	
< and >	Increase/decrease alteration scale	
X and x	Increase/decrease X-rotation angle	✓
Y and y	Increase/decrease Y-rotation angle	✓
Z and z	Increase/decrease Z-rotation angle	✓
<b>Other</b>		
G	Create gravity well at current position	
g	Create repulsion field at current position	

## 4.4 Model Conversion Process

Evaluation of the models is of utmost importance in this thesis. The fitness functions used – which are defined in Chapter 5 – each vary with respect to the data they need for computation. These functions, whether geometric or aesthetic, require extensive geometric information from the models in order to calculate their fitness scores. The models generated from the L-system in previous steps are composed entirely of voxels in a  $128^3$  volume. For a few of the fitness functions used in this thesis, this is acceptable, but many of them rely solely on the surface make-up of each model, measuring surface normals, areas and distributions. In voxel space, each surface is composed of four vertices forming a square, where each square is exactly the same area as all others on the surface. In addition, since each voxel is a cube of the same orientation in 3D, there are only six possible surfaces and therefore six possible normals –  $(1,0,0)$ ,  $(-1,0,0)$ ,  $(0,1,0)$ ,  $(0,-1,0)$ ,  $(0,0,1)$  and  $(0,0,-1)$ . In order to solve this issue, a surface is built over the voxel volume, generating a triangulated mesh

capable of additional normals, surface areas and edge lengths. This is done using the Marching Cubes Algorithm, outlined in Section 4.4.2. Once the algorithm has generated a mesh, additional post-processing is done to the model in order to prepare it for fitness evaluation, which is outlined in Section 4.4.3.

#### 4.4.1 Voxel Distance Calculation

Prior to further processing, the voxel-volume is subjugated to distance calculation at each voxel. This step involves going through every 'enabled' voxel and finding how deep it is within the model. This is easily computed by calculating how far it is from the first occurrence of a 'disabled' voxel from its position, radiating outward. Although potentially time-consuming, this step is necessary for the Marching Cubes Algorithm in order to aid in producing a larger set of unique surface normals.

#### 4.4.2 Marching Cubes Algorithm

The Marching Cubes Algorithm was proposed by Lorensen *et al.* and is used to create a triangular mesh that approximates an iso-surface [31]. The algorithm takes in a volume composed of iso-values – which are equivalent to the voxel-values in voxel space – and generates a surface that intersects the outer-most voxels. It takes into account the actual iso-values as well in order to more accurately represent the surface, increasing the number of potential surface normals. The algorithm consists of two steps – finding the triangulated surface from a given iso-value, and calculating the normals to the surfaces for each vertex in each triangle.

In the voxel-volume, voxels are processed eight at a time, grouped into cubes consisting of a single vertex. The algorithm uses the iso-values of each voxel in each cube to determine how the surface intersects the cube. A surface intersects a cube only if the iso-value of the vertex exceeds or equals the value of the surface to be constructed. Vertices outside the surface are disabled, while those inside are enabled. Once all cubes are processed, the list of vertices (ordered) is added to the mesh, which can be generated by connecting each vertex to its successor in the list. This produces the mesh that is used for further processing. A more detailed description of this algorithm can be found in [31].

#### 4.4.3 Post-processing Steps

The Marching Cubes algorithm produces a complete list of vertices and edges representing the approximated voxel surface, where the vertices are sorted based on their  $(x, y, z)$  values. This sorting is necessary for the success of the distribution-based fitness functions described in Chapter 5. In order to prepare this information for the fitness evaluation step, additional post-processing must be done which provides detailed mesh information and decreases the processing time of the evaluation step.

### Model Validity Testing

With the advances to the GP L-system encoding and its function set, it is rare for any generated L-system to produce an invalid model. A model is defined as invalid if it:

1. produces no vertices after the finalization of the Marching Cubes algorithm, usually as a result of sparse voxels,
2. consistently draws out of the bounds of the voxel space, or
3. exceeds the given time limit for processing the evaluation string and producing the finalized mesh.

In the event that one or more of these cases arise, a model is flagged as invalid. An invalid model is not subjected to fitness evaluation and is given the lowest possible fitness as a result. This decreases the likelihood that the individual that generated the model will reproduce during the reproduction phase of the GP.

### Gravity Wells

Gravity wells are vertex-modifiers in the L-system alphabet – denoted with 'G' and 'g' – and are inspired from the work of Hemberg and O'Reilly in the *GENR8* system [16]. They were introduced to the alphabet as a way of introducing new surface normals to the surface generated from the Marching Cubes algorithm, as well as to aid in producing more fluid and natural forms. There are two types; gravity wells (g) – which pull vertices inwards by a force  $F_g$  – and repulsion fields (G) – which push vertices away by a force  $F_G$ . In the alphabet, gravity wells are merely placed at the current coordinate of the turtle in fractional space, as opposed to being placed at a specified voxel. During the processing of the evaluation string, all gravity wells and their position and force are added to a list, which is processed during the post-processing stages.

The effect of the gravity wells is calculated during the post-processing stages. The calculations themselves are simple – for a list of vertices  $V$  and gravity wells  $G$ , for every  $v \in V$ ,

$$v = v + \sum_{i=1}^n f(v, g_i) \text{ where } g \in G \text{ and } f(v, g) = \frac{\overrightarrow{(v, g^p)}}{d(v, g^p) * 10/F_g} \quad (4.1)$$

In the above equation,  $g^p$  is the position of gravity well  $g$ ,  $d(v, g^p)$  is the distance from vertices  $v$  and  $g^p$  and  $\overrightarrow{(v, g^p)}$  is the vector between the two vertices.  $F_g$  is the force or magnitude of a gravity well, and is determined prior to the start of a run. The equation simply adds the effects of all gravity wells to each vertex, altering its position. Each gravity well's effect on a vertex decreases with increasing distance, and increases with increasing magnitude. Using this formula, gravity wells that are stacked can accumulate their effects, sometimes cancelling one another out or amplifying

their magnitude substantially. Gravity wells have the capacity to push or pull vertices outside the voxel space's boundaries, though at this point the effect is encouraged instead of punished.

### **Mesh Details**

The final post-processing step involves calculating and ordering the mesh data in a way that is useful for fitness evaluation. Vertices are grouped according to adjacency, which is useful for distribution calculations. Mesh volume, surface area, complexity, unique normals and dimensions are also calculated here and stored for later use. This final data, the resulting mesh, and the pruned L-system are stored in the GP chromosome for fitness evaluation, where it can be accessed and saved externally. This data has the capacity to be re-loaded into the JNetic system for viewing, as well as for exporting into raw mesh data.

## Chapter 5

# Model Evaluation

Once a model has been manufactured from the L-System generated from a GP-tree, various descriptive elements are calculated which can be used to accurately describe the physical characteristics of the model. These elements are more specifically used in the fitness evaluation of the model, which occurs at the end of each generation of the GP loop.

A variety of fitness functions were chosen in order to extensively experiment on their impact on the evolution of aesthetic models. Several of the functions are created to specifically focus on a model's aesthetics – such as the DFN and Complexity measurements – while others are reserved to set physical constraints on the models themselves. It was expected that, alone, many of these functions would produce uninteresting results. For example, if we simply chose to use the *Dimension* constraint, GP has the potential to produce any possible model that sits within that function's 3D rectangular bounds. Therefore, a multi-objective approach was chosen to group several of these functions together, in order to reduce the number of possible models that fit the descriptors. By doing this, it is also expected that more aesthetically-pleasing models will be produced.

In total, there are eleven different fitness functions, split into two groups: Model Constraint Functions and Distribution-based Functions. From this suite of fitness functions, only a small handful of these are chosen for each run, as having too many active fitness functions increases the search space substantially, resulting in longer run-times and decreased performance. It is also worth noting that many of the functions do not work well together, such as each DFN measurement and their  $1/f$  Noise counterpart when using the same distribution measurement. Not every fitness function is suitable for each L-System language either, as many require descriptor data not present in the resulting models of certain grammars. This is especially present in the Plant-drawing System. Each function and the alphabets that can use them can be seen in Table 5.1.



Table 5.1: Fitness Functions and their appropriate L-System grammars

Function	Voxel Draw	Voxel Surface	Plant Draw
Volume	✓	✓	
Dimension	✓	✓	✓
Surface Area	✓	✓	✓
Unique Normals	✓	✓	✓
Complexity	✓	✓	✓
Deviation from Normal	✓	✓	
1/ $f$ Noise	✓	✓	
Entropy	✓	✓	
Mean	✓	✓	
Standard Deviation	✓	✓	
Symmetry	✓	✓	✓

## 5.1 Model Constraint Functions

The fitness functions listed in this section are used to constrain the physical properties of a model, in order to limit the target search space when combined with the fitness functions associated with aesthetics. These functions are also employed to limit the physical size of a model in order to comply with Java memory restrictions.

### 5.1.1 Volume

Due to the fact that the models are generated from a voxel-based form, *Volume* is a simple calculation. Each active voxel 'block' in the mesh is attributed the value of one cubed unit, and the volume is calculated as the sum of all active units. Since the Marching Cubes algorithm gives an approximation of a 3D form over a voxel surface, the volume fitness function is also an approximation, though it is fairly accurate. Heavy usage of the *gravity wells* in the L-System grammars reduce the accuracy of this function.

### 5.1.2 Dimension

The *Dimension* fitness function measures the difference in the width, height and depth of a model's bounding box from a bounding box specified by the user. As previously mentioned, the voxel space is a 128 x 128 x 128 cube, and so it may seem logical to assume that the dimension of the model is limited to these bounds. This is incorrect, however, as the gravity wells have the ability to increase the size of a model on all axis by floating point values, which also results in a change of measurement from integers (voxels) to floating point values.

### 5.1.3 Surface Area

A model's *Surface Area* is measured as the sum of all face areas on the model. Each face consists of three vertices, and so the area is simply calculated using the area of a triangle in 3D. Through the use of the Marching Cubes algorithm, it is impossible for completely hidden faces to exist in the model, though some faces have the capacity to 'meld' together when gravity wells are used, causing minor fluctuations in the accuracy of this function.

### 5.1.4 Unique Surface Normals

Every face in the polygonal model has a face normal  $(x, y, z)$  representing the direction it is facing. The face normal – also called a surface normal – extends in a direction perpendicular to the face's surface, and is used in the calculation of lighting for 3D models. The *Unique Surface Normals* fitness function calculates the number of unique surface normals found in a single model, where a larger number of unique normals represents a higher fitness value. Since the values are floating point, two normals are considered equivalent if their vectors are identical within four decimal places, for each  $x$ ,  $y$  and  $z$ .

## 5.2 Aesthetics-based Functions

The fitness functions described in this section are associated with aesthetics, most of which have been implemented in other work (see Chapter 3). They are added in an attempt to increase the visual appeal of the models generated by the GP.

### 5.2.1 Distribution-based Functions

A distribution-based fitness function is one which measures the distribution of some specified model data over the entire model. The actual measurement of the data is specific to each fitness function. There are two chosen distributions capable of measurement by these functions, both of which measure differences across a model's surface. The first is a measure of the signed difference between adjacent face normals, ranging from zero degrees (no change) to 360 degrees. The second is a measure of the signed difference between adjacent face areas. When measuring the differences between the face data of two adjacent faces, the order is important. This means that if the difference between the face normals of faces A and B is 56 degrees, then the difference between B and A is -56 degrees. Due to this fact, faces are sorted based on vertex data prior to fitness evaluation, in order to ensure consistency between model measurements.

### Mean and Standard Deviation

The *Mean* and *Standard Deviation* of a distribution are common calculations, and exist as two separate fitness functions in this thesis. They can be applied to any distribution and any distribution-based function. When calculating fitness, the current mean or standard deviation is compared to a user-specified value, returning the absolute difference between the two as a fitness value.

### Deviation from Normal

The *Deviation from Normal* (DFN) function measures the difference between the histogram generated by the distribution's raw data and the estimated normal curve generated by the same data. The generation of the histogram is simple: the input data is separated into approximate bins and tallied, where each bin is indexed in an array. The normal curve is generated by first calculating the mean and standard deviation of the distribution, then calculating the expected probabilities for each bin value  $x$  from min to max, using a specified increment. The probabilities are calculated using

$$curve(x) = \frac{1}{STD * pDensity((x - mean)/STD)} \quad \text{and} \quad pDensity(x) = \frac{1}{\sqrt{2 * PI}} * e^{-\frac{x^2}{2}}$$

Using the probabilities generated using the above function, a second histogram is created storing the expected frequency of each bin value, which is calculated using

$$expected(x) = curve(x) * \frac{1}{dif} * total \quad \text{where} \quad dif = \frac{1}{inc}$$

where *inc* is the increment between bin values, and *total* is the size of the input data set. Once the two histograms have been created, the absolute difference between them is calculated at each bin index, summed and set to the range of 0 to 10. A DFN of 0 is a perfect match to the normal curve, and a DFN of 10 is the worst possible case. Figure 5.1 shows an example of a good and bad fit to the estimated normal curves of two models. The complexity of this function depends greatly on the size of the data set generated by the model, and so depends on the number of vertices and edges in the model. It is expected that reaching a target DFN of zero will be rare with more complex models.

### 1/f Noise

The 1/f Noise function measures the difference in the slope of the line of closest fit to the distribution data and the 1/f curve for a specific beta value. To do this, a histogram is generated for the distribution data in the same manner as for the DFN, and a line of best fit is computed for the logarithm of this data, which converts the shape of the line closer to that of the 1/f curve. The absolute difference between the two slopes is then calculated in order to find the differences between the two lines. Since the line of best fit can be a heavy approximation of a potentially sparse data set,

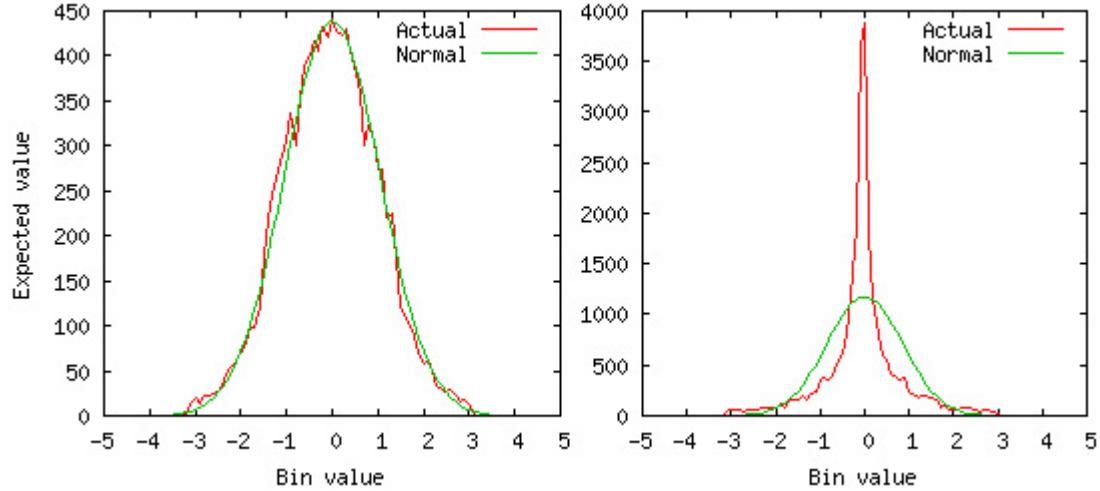


Figure 5.1: Two examples of DFN curves and respective actual curves

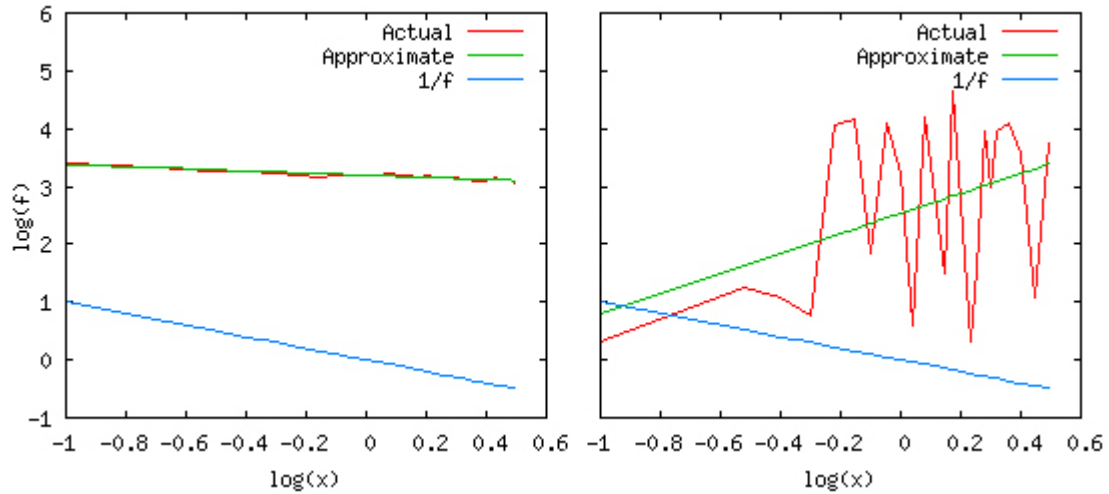
a punishment factor is considered as well, which measures the average absolute difference between the actual data set and the line of best fit, at each  $x$  value for a specific increment. The punishment value is then added to the difference in slope. Ideally, the difference between the two slopes is zero and the punishment value is zero, which means that a higher  $1/f$  score indicates a poorer fit to the  $1/f$  curve. For this measurement, the difference in the  $y$ -intercept of the two curves is not considered to be important, and so the two curves can be separated by virtually any distance with no impact on the fitness score. This decision was made due to the difficulty that was already seen in matching the  $1/f$  curve from the initial empirical study of the fitness function. Figure 5.2 shows an example of a good and bad fit to the  $1/f$  curve for two models. The left graph shows a data set that closely fits its line of best fit, which resulted in a low punishment score. Its slope is relatively close to that of the  $1/f$  curve. The graph on the right shows a poor fit, with a high punishment score and a large difference in slope.

### Entropy

The entropy fitness function uses the concepts behind Shannon's Entropy, which measures the level of uncertainty associated with the distribution data of a model. It is calculated using

$$Entropy(X) = - \sum_{x \in X} p(x) \log p(x) \quad (5.1)$$

where  $x$  is a value in the data set and  $p(x)$  is the probability of the value occurring in the data set. To calculate this, the data is sorted into bins within a histogram as with the DFN and  $1/f$  functions,

Figure 5.2: Two examples of  $1/f$  curves and respective actual curves

and the bin values are used in the calculation for probability instead of the actual values. A lower entropy indicates a low level of uncertainty, which typically results from a more simple model. A higher entropy indicates a high level of uncertainty, common with large, complex models.

### 5.2.2 L-system Complexity

The *L-system Complexity* measurement is inspired from the work of Komolgorov [28], the Box-counting Method [40], and Machado [33]. Although measurements for the complexity of fractals and L-systems exist for systems that generate 2D images, very little research has been done in extending these measurements to 3D. This is largely due to the increase in complexity of a 3D L-system, as well as the amount of data made available by a 3D model. Typical 2D complexity measurements have focused on the increasing amount of detail found in images at different zoom levels – as seen in the Box-counting method for fractal images – and the differences between the file sizes of compressed and uncompressed versions of an image – as seen with Kolmogorov complexity using JPEG compression. Others focus on the fractal-generating function itself, examining the simplicity of the function versus the complexity of the produced image – as seen in Machado’s work. The complexity measurement used in this thesis is based directly on these ideas.

This measurement – now referred to simply as *complexity* – is a measure of the capacity for growth of an L-system’s evaluation string  $\omega^i$  over many iterations. As mentioned earlier, an L-system’s complexity is directly related to the growth rate of  $\omega^i$  with respect to the number of

variables present. This measurement is simply

$$growth = \frac{\sum_{i=0}^{n-1} \nu(\omega^{i+1}) - \nu(\omega^i)}{n}$$

where  $\nu(\omega^i)$  is the number of variables present in  $\omega^i$ , and  $n$  is the iteration span. This equation gives the signed rate of growth of the L-system, but does not take into consideration the actual model generated from  $\omega^n$ . This is problematic, since it is possible for an evaluation string to continuously draw a single shape over itself, resulting in a simple model constructed from a complex L-system. For example, when the rule  $C \rightarrow CCCCC$  is applied to the starting string  $C$  multiple times, it results in a long string of  $C$ 's. When the pen inevitably goes out-of-bounds of the drawing space, drawing ceases, and the model's complexity suffers. To remedy this problem, the size of the resulting model is considered as well – more specifically, the surface area of the model. Since the surface area tends to overshadow the complexity measurement, the logarithm function is used to reduce the 'reward' given as the complexity and surface area grow. This also reduces the chance of a system becoming *too* complex. The final complexity function is then

$$complexity = \log(growth) + \log(surface\ area)$$

The complexity of a model is greatest when its surface area is large and its L-system growth is large and positive. One of the primary problems with this is the fact that when its target is a maximum value through evolution, most models that are generated will often be excessively complex, resulting in long rendering and processing times, and ultimately resulting in models which exceed the time limits imposed by the user. Due to this, the largest score that can be returned by this function during the GP evolution is directly related to the time limit, which must be considered prior to setting its target. Due to the nature of this function, it is best used when trying to achieve a large complexity target, as low targets are ultimately achieved by error-prone L-systems.

### 5.2.3 Symmetry

The *Symmetry* fitness function is used to measure the approximate physical symmetry of a model. This is done by measuring the distribution of vertices across the model along the three major axes, recording the separate symmetries of each axis and returning the highest value. The general idea behind this method is inspired from the visual symmetry of 2D images, where an image can be symmetrical along the x-axis even when both sides look nothing alike. The overall form and shape offer minor influence over the symmetry of the image – instead, the general presence of form across a central axis dictates symmetry. In order to measure this, the image is divided into boxes of equal size, as seen in Figure 5.3. The initial image (left) is divided once, resulting in the image in the

middle. To calculate the symmetry, each resulting column is compared to its mirror version, index-by-index, looking for a 'presence' of color. In this case the image is only divided once, and so the resulting calculation is not very accurate. In order to increase the accuracy, further divisions are done, as seen in the image on the right. The number of divisions should not be too large or too small – too many divisions increases accuracy, but succeeds only with perfectly symmetrical images, while too few divisions result in general symmetry calculations.

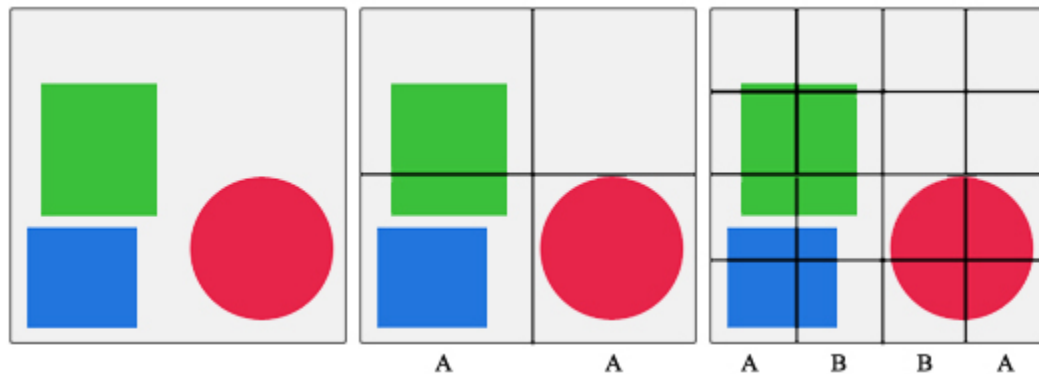


Figure 5.3: 2D image division for symmetry calculation

For this thesis, this idea was extended to three dimensions by dividing the model's bounding box into cubes. Instead of measuring the presence of color, the number of vertices present in each cube is compared to its mirrored cube, and the absolute difference is added to a global total. This total is divided by the total number of vertices in the model, and subtracted from one. Therefore, a symmetry of one results from a perfectly symmetrical model, while a symmetry of zero results from a perfectly asymmetrical one. The bounding box is divided into 20 x 20 x 20 cubes (8000).

### 5.3 Fitness Targets and Analysis of Pre-Existing Models

From the previous section, it is clear that there are many potential fitness functions that can be used. The amount of initial study required to find ideal target fitnesses for each function within this set would be tremendous, and so steps were taken to reduce this. For the model-constraint functions, the target fitnesses depend heavily on the problem presented to the GP. If larger, more complex models are preferred by the user, then the targets for the dimensions, surface area and unique normal fitness functions will likely be high. Relying on specific targets for these functions – although ideal for some problems which require accuracy, such as those associated with commercial designs – is not advised. The probability of the population converging on such a solution is low. The target values for these functions are merely heuristic, and should not be taken literally. Instead,

they should be interpreted as point ranges in the search. The models generated from this system are meant for design inspiration, not to produce real-world design plans. Therefore, there is no need to consistently use specific targets for these fitness functions. For example, if a large model is desired, then a surface area of 100,000 to 500,000 would be perfectly suitable.

For the other fitness functions – such as those based on aesthetic measures – such consistency is expected and even required to support the work in this thesis. For many of these functions, the following general assumptions and initial hypotheses are made: more aesthetic models are associated with low DFN scores, high entropy scores, high symmetry scores and low  $1/f$  scores. For complexity, Birkhoff speculated that a lower complexity is ideal for high aesthetic value [7]. Due to the nature of the complexity function presented in the previous section, a model with an ideal lowest-complexity score is small, simple and – in most cases – a single cube. The complexity function used in this thesis instead focuses on the L-system complexity, and its capacity to produce large, intricate models. The hypothesis is then extended for complexity, where an aesthetic model is one with a higher L-system complexity.

In order to justify the choices for the target fitnesses of the aesthetic-based functions, 200 pre-computed models were subjected to several of these functions, with the hopes that the majority of the models would fall within a certain target area for each function. The models themselves were chosen from Archive 3D, a website dedicated to the production and free distribution of 3D models [1]. Each model belonged to one of four categories – object, human, plant and polygon. The goal of the experiment was to locate potential *sweet spots* – fitness target areas which many of the models had in common. The discovery of any of these potential target areas would provide insight into the fitness targets for the experiments later in this thesis. The fitness functions chosen were DFN, entropy, symmetry and  $1/f$  noise. Complexity was left out due to its dependency on models constructed from L-systems. For the measurement used by the distribution-based functions, the difference between adjacent surface normals was chosen, as it would be focused on in the majority of the experiments. Each model was not subjected to a time-limit in its fitness evaluation, as others are during the GP runs in later experiments. The justification for choosing the models is based on the idea that each model was in essence created for an aesthetic purpose using 3D modeling software – to catch the eye of those viewing the model. Therefore they are aesthetic to at least two people – the author of this thesis and the creator of the model.

The distribution of the results can be seen in Figures 5.4 to 5.6, and a few examples can be seen in Figure 5.7. From these histograms, it can be seen that the majority of the models fall within 2.5 to 4.0 for DFN, 2.0 to 3.0 for  $1/f$  noise and 2.0 to 3.0 for entropy. For symmetry, the majority of the models fall between 0.5 or 1.0. For mean, the majority fall between 0.0 to 0.2, and for standard deviation the majority fall between 0.2 and 0.8, which is a large range and not too centralized. Although it was expected that many of the models would have lower DFN scores, higher entropy scores and lower  $1/f$  noise scores, the results have shown that a significant number of models have



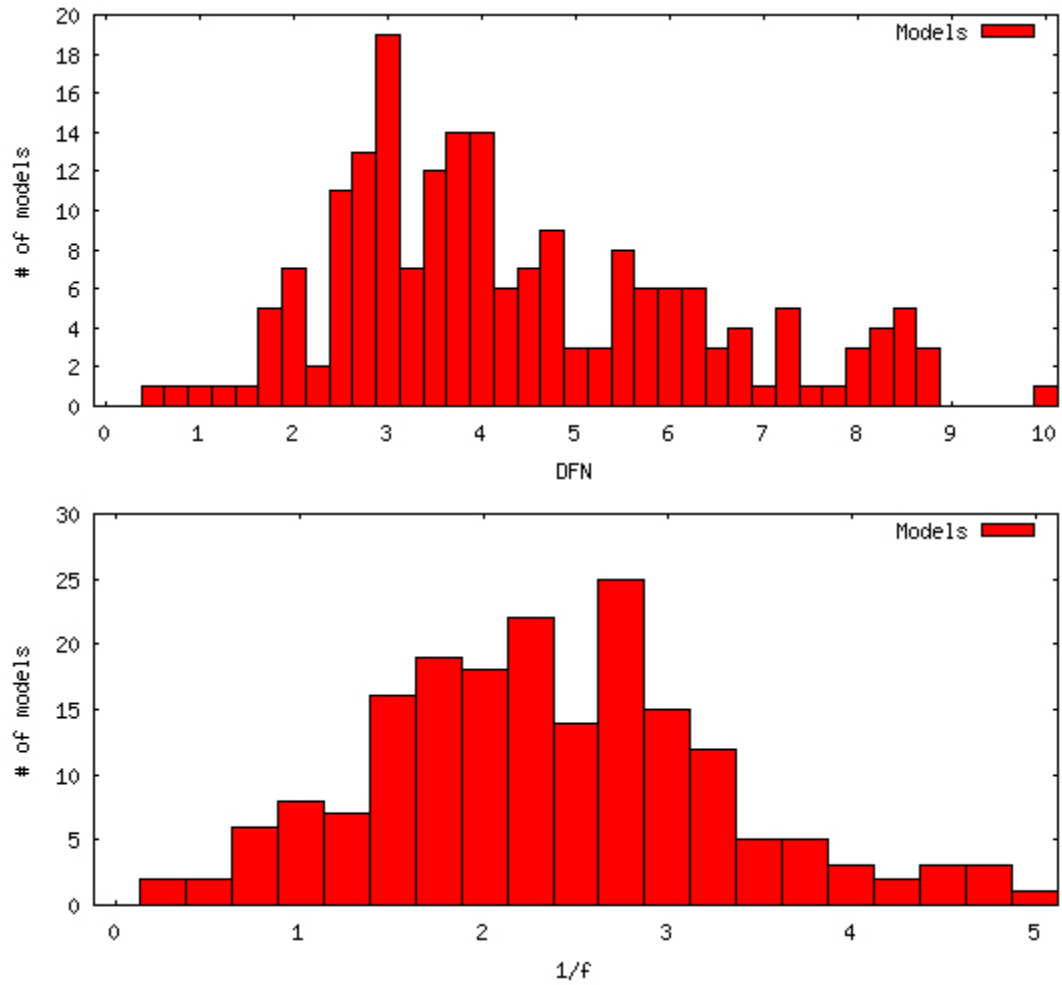


Figure 5.4: Distribution of DFN and  $1/f$  fitness scores calculated for approximately 200 pre-existing models

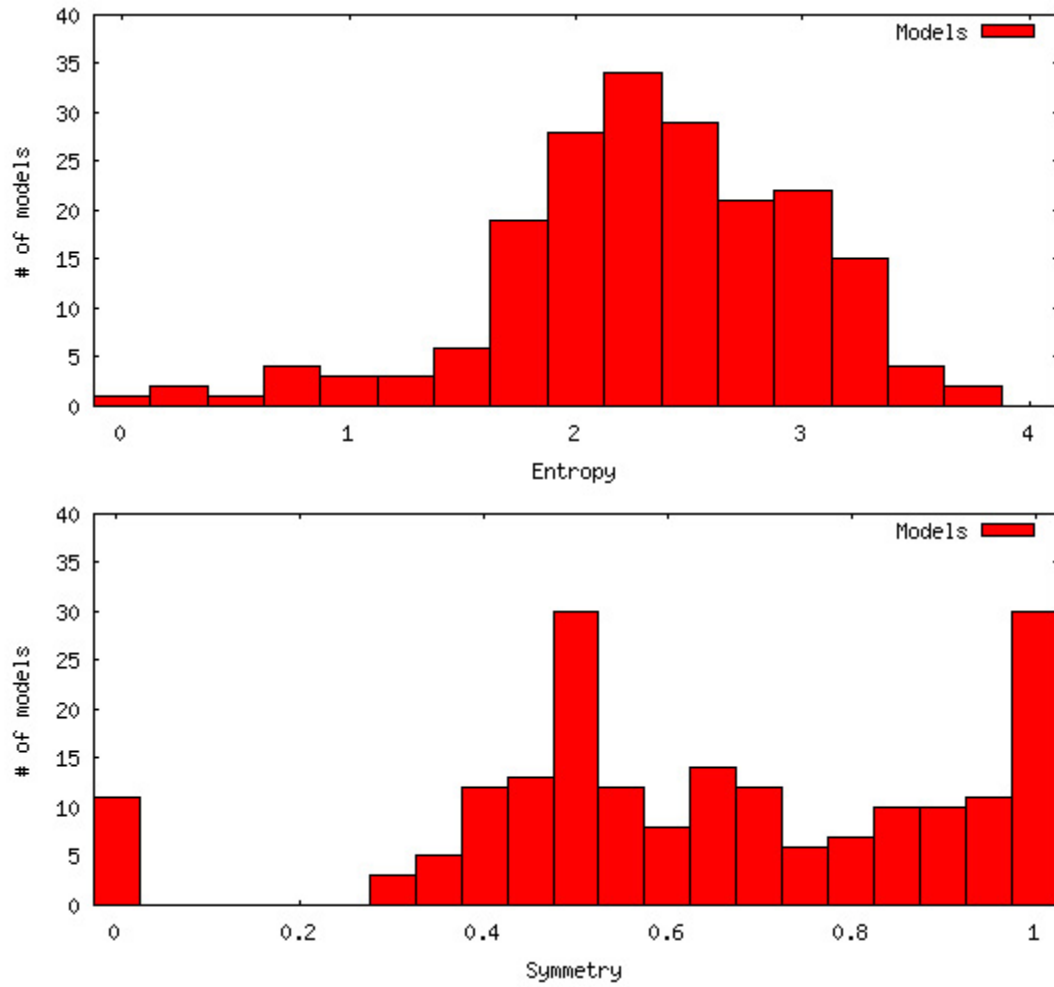


Figure 5.5: Distribution of entropy and symmetry values calculated for approximately 200 pre-existing models

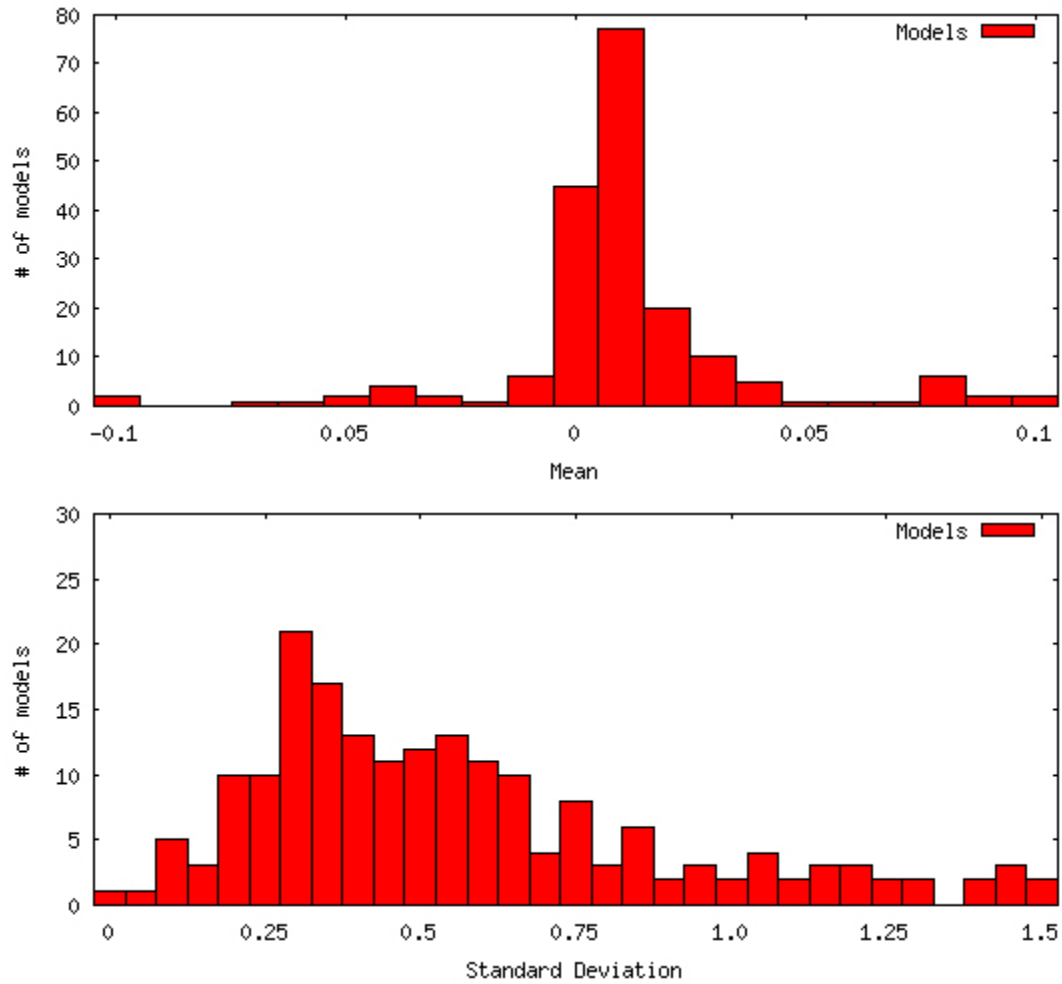


Figure 5.6: Distribution of mean and standard deviation values calculated for approximately 200 pre-existing models

scores relatively close to these targets, and far from the expected targets for less-aesthetic models. The distance between the observed sweet spot and the hypothesized sweet spot can be explained by the complexity of the models chosen for this experiment.

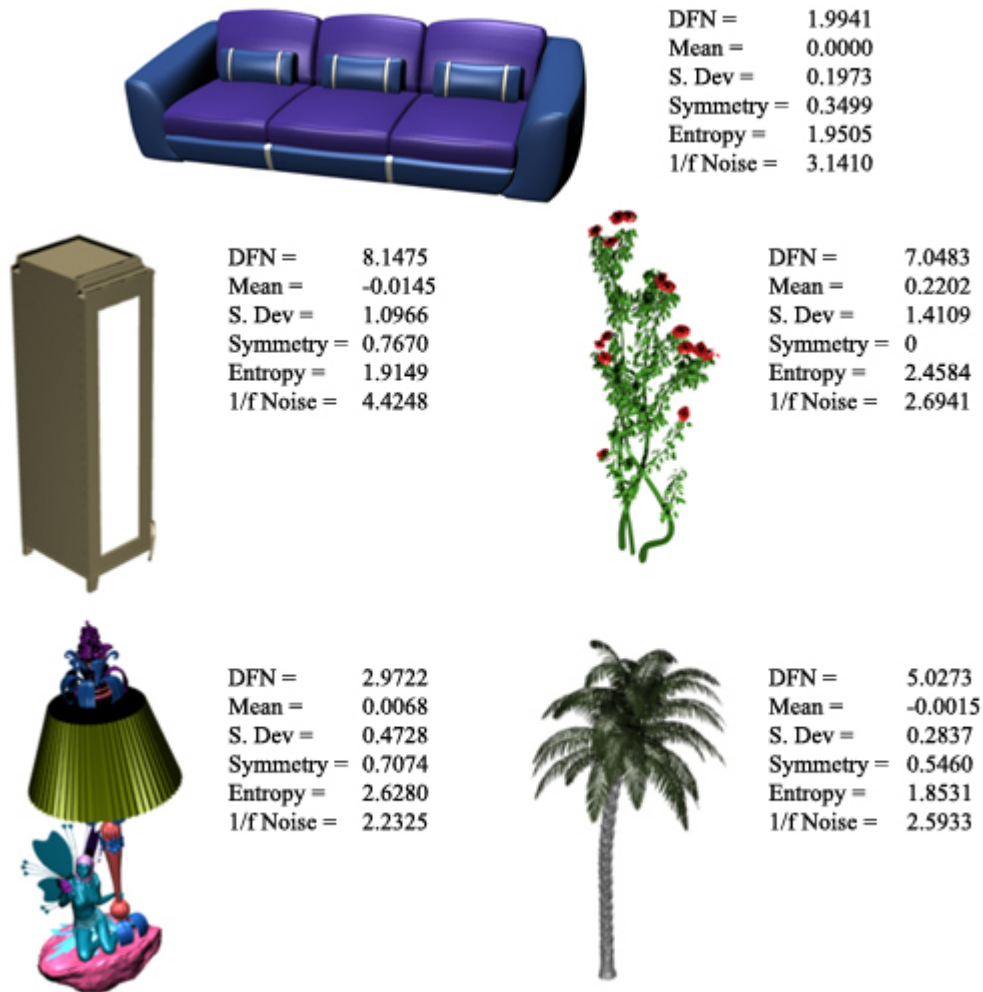


Figure 5.7: Samples taken from the set of pre-existing models, as well as their scores in six categories

The majority of the models chosen contained hundreds of thousands of vertices – creating a distribution that is extremely large and therefore difficult and unlikely to match to a normal curve,  $1/f$  curve or any other expected distribution. The same effect is observed for the symmetry calculations, which depend on the positional distribution of vertices. From this information, it is expected that very few models would garnish a low DFN, or high entropy. In fact, those that came close to these targets were the most simple models, which may further support Birkhoff's speculation regarding the aesthetic-complexity relationship.

It is rather difficult to manually assign or guess a model's score that is close to that model's actual score, for either of the distribution-based functions. The range of fitness scores seen from the models in Figure 5.7 are a perfect example of this. It is possible, however, to discern a model with a higher score from those with a lower score. The increasing divisions of a sphere, for example, show an increase in the DFN of that sphere, despite the increase in its complexity. A sphere with up to seven divisions garnishes DFN scores of 8.0833, 6.6052, 4.5105, 3.0078, 2.9139, 2.9093 and 2.9086 for each increasing division. At its most basic, it has a high DFN – common with simple, block-like models – and decreases its DFN as it increases the number of divisions. For each division, the difficulty in manually discerning the estimated DFN between the two spheres becomes increasingly difficult. The same concept applies to the other fitness functions. This idea supports the use of EC and automated evolution using these aesthetic measures, which removes this difficulty from the user.

As a final test, the Pearson correlation between the fitnesses of all models was calculated for the chosen fitness functions, the results of which can be seen in Table 5.2 [59]. Correlation coefficients closer to 1.0 or -1.0 indicate a high correlation between two fitness functions. From this table, the strongest correlations can be seen between the standard deviation and the DFN/entropy, and between entropy and the DFN. The correlation between the DFN/entropy and standard deviation is expected due to the relationships between each function's calculations. The correlation between DFN and entropy is far more interesting, and easily merits further study.

Table 5.2: Correlation between several aesthetic-based fitness functions, calculated using approximately 200 pre-existing models

	<i>DFN</i>	<i>Mean</i>	<i>Std. Dev.</i>	<i>Symmetry</i>	<i>Entropy</i>	<i>1/f</i>
DFN	1					
Mean	0.081644	1				
Std. Dev.	0.478077	0.249283	1			
Symmetry	-0.04697	-0.03974	-0.16465	1		
Entropy	-0.38027	-0.00526	0.424442	-0.10574	1	
<i>1/f</i>	-0.08336	-0.07625	-0.12265	0.076309	-0.18523	1

Although this experiment has shown that potential sweet spots exist, further study into this subject is still required to make any sound conclusions. In the future, a larger data set of models may be useful, hand-picked by multiple parties to introduce a decrease in bias. Exploration in the production of models which satisfy the hypothesized ideal fitness targets for these aesthetic functions is still of greater interest, though the use of the discovered sweet spot targets will be explored in later experiments. In addition, the possible correlation between entropy and the DFN will be investigated.

## Chapter 6

# Setup of Experiments and L-system Improvements

### 6.1 Outline

The experiments that follow in this chapter and later chapters are introduced as a means of supporting various claims and hypotheses presented. Each experiment investigates a different aspect of this research, and it outlined as follows. First, each experiment is introduced as well as the goals behind it and any initial hypotheses that can be made. The parameters of each experiment are also introduced, and reasoning behind their settings. The results are then introduced in various statistical and visual formats, and discussed with respect to the initial hypotheses and goals. Conclusions are finally made on the findings at the end of each chapter.

This chapter discusses the general experiment parameters used for the majority of experiments. It also introduces the first experiment, which compares the L-system encodings of Jacob and the encoding implemented for this thesis.

### 6.2 Experiment Parameters

The majority of the experiments were run on a cluster of computers in the Computer Science Department at Brock University. Half of the cluster are AMD Phenom II 1090T (6 core at 3.2Ghz) with 8GB of RAM running in double channel mode at 1600Mhz. The other half are Intel Core i7 920 (4 core hyper-threaded 2.66Ghz) with 12GB of RAM running in triple channel mode at 1333Mhz. All machines are running on CentOS 5.5 Linux. Initial empirical experiments, coding and research were done on a PC running Windows XP, with Intel Core 2 Q9400 (4 core at 2.66Ghz) with 3.25 GB of RAM.

The choice of parameters for the GP and L-System constraints are described in this section, as well as selected short explanations for a few of them. For the GP tree initialization, *Ramped half-and-half* is used, which combines the usage of the *Full* and *Grow* methods. The Full method chooses only non-terminals from the function set to generate the tree up to a specified maximum tree depth. Once each branch has reached the maximum depth, only terminals are chosen. This method aims to fill a GP tree as much as possible, using the maximum number of nodes possible. The Grow method is similar to the Full method, but instead chooses from both terminals and non-terminals in order to reach the maximum depth. By doing so, this goal is not always attained, resulting in a variety of smaller more erratic trees. The combination of the Grow and Full methods for Ramped half-and-half helps to generate a more diverse population. This method is also used when generating subtrees during mutation. Tournament selection is used as a selection mechanism. In this method, a number of individuals are chosen randomly from the population based on the *tournament size*  $T$ . From these individuals, the one with the highest fitness is chosen to reproduce. Since two parents are required to produce two offspring, two tournaments must take place for each set, one for each parent. Elitism was also used, where only one individual is saved per generation.

The remainder of the GP parameters can be found in Table 6.1. Each was chosen after extensive initial empirical study, including background research in genetic programming and the typical values of its parameters. A large population size was chosen to increase the diversity of the population, specifically at initialization. Tree sizes were especially limited due to the rate of increase of L-System complexity with increasing depth, which resulted in unnecessarily long evaluation string parsing times. The values chosen to limit the sizes of the RHS of production rules is directly related to this issue as well. Any alterations to these parameters for a particular experiment are explained in the experiment's introduction.

Table 6.1: GP And L-System Parameters

GP Parameter	Value	L-System Parameter	Value
<b>Generations</b>	60	<b>Rule RHS Max / Min</b>	2 / $\sim 10$
<b>Population Size</b>	500	<b>Iterations Max / Min</b>	6 / 3
<b>Crossover Rate</b>	90%	<b><math>\omega</math> Max / Min Length</b>	2 / $\sim 10$
<b>Maximum Crossover Depth</b>	10	<b>Alphabet</b>	C, A, B, +, -, *, /, @, &, w, W, d, D, h, H, [, ]
<b>Mutation Rate</b>	10%	<b>Variables</b>	C, A, B
<b>Maximum Mutation Depth</b>	17	<b><math>\omega_i</math> Parsing Time-out</b>	30 seconds
<b>Prob. of Terminals in Cross/Mut</b>	10%		
<b>Tournament Size</b>	3		
<b>Tree Grow Max / Min</b>	5 / 5		
<b>Tree Full Max / Min</b>	12 / 5		

Typically, thirty runs are used in order to show statistical significance in experiments. The experiments here are limited to ten runs, which resulted due to the average run-time for each

experiment. A typical run takes between one and five hours, though runs which use complexity or unique normals as fitness functions can potentially take much longer.

Finally, a diversity measure was implemented in order to increase the overall diversity of the final population after convergence. This measure was created by Flack for his work in a floor plan evolutionary design problem [10]. This measure works as follows. After the population is evaluated and fitness scores are assigned – or for MO, ranked using a MO strategy – the population is parsed for duplicates. In the event that a duplicate is found, one of the duplicates is given a raw penalty score, the value of which is chosen by the user. This penalty is subtracted from the fitness of the duplicate. A duplicate in this case is an individual which has the exact same evaluation string as another.

## 6.3 Analysis of L-system Encoding Improvements

### 6.3.1 Introduction and Setup

The goal of this experiment is to show the benefits of using an L-system encoding with strict constraints, and the ability to recover an invalid L-system. This is to be achieved by comparing this new encoding with one created by Jacob *et al.*[22], who produced a potential means of D0L-system encoding for GP. Although his encoding was meant for a parameterized D0L-system, little modifications were necessary to allow for a non-parameterized one. The primary outline of his encoding is described briefly as follows:

- The root of the GP tree returns a complete L-system composed of a starting string  $\omega$  and one or more production rules
- $\omega$  returns a stack expression, where a 'stack' is a sub-expression composed of symbols from the alphabet, and possibly other stacks
- Each learning rule has a LHS which is a single symbol from the alphabet
- Each learning rule has a RHS which is a stack expression
- GP operators such as crossover and mutation encourage the production of valid L-systems throughout the evolutionary process

Jacob's encoding can produce any hypothetical D0L-system, though the produced systems might not necessarily be valid by the definition of validity presented in this thesis. The primary difference between the encoding A and Jacob's is the use of the stack, which required a variable-length number of children for stack nodes. In order to emulate the use of the stack in encoding A, sub-expressions were instead generated using functions returning the  $L_{list}$  subtype, as seen in the function set definition in Table 4.2. In actuality, encoding A is very similar to encoding B – the only difference



between the function sets of the two encodings is that variables are never formally required in the LHS, RHS or  $\omega$  in encoding A.

There are three main problems with encoding A. First, symbols can be used in the LHS of multiple production rules, resulting in a non-deterministic L-system. Second, it is possible for a chosen LHS symbol for a production rule to not exist in the RHS of any other production rule or  $\omega$ , which results in unused production rules. Third, the differentiation between variables and non-variables is also not defined in this encoding, and so non-variables can be chosen for a production rule's LHS, which is not desirable. It is important to note at this point that the work in this thesis is not meant to scrutinize the work of Jacob *et al.*, but instead to offer an alternative GP tree encoding for D0L-systems with additional constraints. Although the GP operators in their encoding attempt to increase the number of valid L-systems during the course of evolution, it might instead be more desirable for the initial population to consist of more valid L-systems, which will be beneficial for evolution performance.

### 6.3.2 Results

For the experiment, two sets of ten runs were executed – one set for encoding A and B. Each run had identical parameters, alphabets and fitness function. As the focus of this experiment was not on the fitness function itself, but the validity and characteristics of the L-systems generated by the encodings, four new measurements were recorded on a generational basis, where each measurement is averaged over the population. The *percentage of unused production rules* measures the number of production rules never referenced when generating an evaluation string. The *L-system complexity* measures the average rate of variable growth of  $\omega^i$  in an L-system over each iteration. The *percentage of invalid models* measures the number of models that were flagged as invalid during model generation, which could happen for a variety of reasons relating to the L-system that generated it. The *percentage of unchanged starting strings* measures the number of  $\omega$  that are equivalent to the final evaluation string used to generate a model.

The fitness function used was the DFN with a target of 0, and the results can be seen in Figures 6.1, 6.2, 6.3, 6.4 and Table 6.2. The graphs display the average performance of each encoding for each measure discussed. The table shows the best performance and average performance of the ten runs for each encoding, at generation 0 and 59. For the percentage of invalid models, unused production rules and unchanged starting strings, a lower value is optimal, while a higher value is optimal for average complexity.

By examining Figure 6.1 and Table 6.2, the number of unused production rules in encoding A is staggering at generation 0, though it begins to even out over time. Even at its best, encoding B far surpasses encoding A, starting under 10% and staying below 1% for the duration of the run. This was expected, as the recovery process in encoding B ensures most rules are used at least once.

It is interesting to note that in the final generation of a few runs, encoding A did have a few lower percentages, even one at 0%. This result is similar to that of the percentage of unchanged starting strings (Figure 6.4), which makes sense considering production rules that are used will always change  $\omega$ . In encoding A, an average of over 75% of starting strings were unchanged.

Table 6.2: Results of encoding experiment from first and final generation of GP

Measurements	Average		Best	
	Encoding A	Encoding B	Encoding A	Encoding B
<b>Generation 0</b>				
Unused Rules (%)	83.7748	6.6604	80.199	5.5762
Ave Complexity	15.5184	828.3133	131.4929	1006.862
Invalid Models (%)	44.1555	10.1555	42.2	7.8
Unchanged $\omega$ (%)	77.0444	2.7555	71.8	2.2

Measurements	Average		Best	
	Encoding A	Encoding B	Encoding A	Encoding B
<b>Generation 59</b>				
Unused Rules (%)	22.2340	0.6646	0	0
Ave Complexity	13.5673	411.7079	43.0157	1566.143
Invalid Models (%)	1.6444	0.2444	0.2	0
Unchanged $\omega$ (%)	0.4222	0.1333	0	0

The results of the complexity measurement were also expected (Figure 6.2), as encoding B enforces variable usage on the RHS of production rules, where encoding A does not. A low DFN is naturally easier to attain with lower complexities, which explains the decrease in the complexity of encoding B over the run, but even so, encoding B still retains a significantly higher complexity in its population than that of encoding A. It was discovered during initial empirical study that this effect is much more desirable in order to produce more interesting models. Figure 6.3 shows the percentage of invalid models, and though encoding A starts out high at the beginning, it quickly decreases and sits under 5% with encoding A. Although this does not seem problematic, it has a dramatic effect on the diversity of the population in early generations. Since over 40% of the models generated by encoding A were flagged as invalid in generation 0, over 40% of the population is assigned the lowest fitness, leaving only 60% of the population to reproduce. This is one of the primary reasons a new encoding was considered in early testing.

The results of the first generation for both encodings were subjected to a two-tailed T-test, assuming unequal variances. This generation, as previously mentioned, is of greater importance than the final generation in supporting the use of Encoding B. The results of the T-tests – one executed for each measurement – can be found in Tables A.1 and A.2. Using an alpha value of 0.05, it can be concluded that Encoding B scored better than Encoding A on average for all measurements in the first generation with a 95% significance.

### 6.3.3 Conclusions

From the results of this experiment, it is obvious that encoding B outperforms encoding A in early generations, by generating more valid L-systems through the constraints in the function set. This being said, the new encoding always outperformed Jacob's on all accounts from generation 0 to 59. It is expected that the performance of Jacob's would have likely been improved had the evolutionary operators discussed in his paper been used. This of course would only have improved the performance of his encoding after the first generation. The enforced used of production rules and variables in the new encoding led to a dramatic difference in the complexity of generated L-systems between the two encodings, as well as the number of unused production rules and unchanged starting strings. This is highly beneficial, as it boosts evolutionary performance.

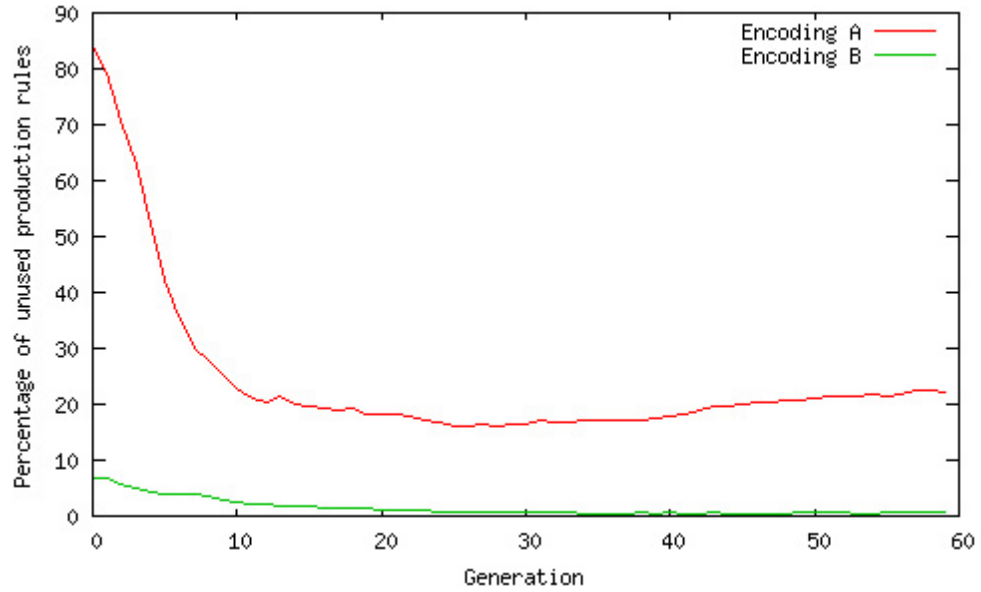


Figure 6.1: Percentage of unused production rules from population of L-systems per generation, between two L-system encodings

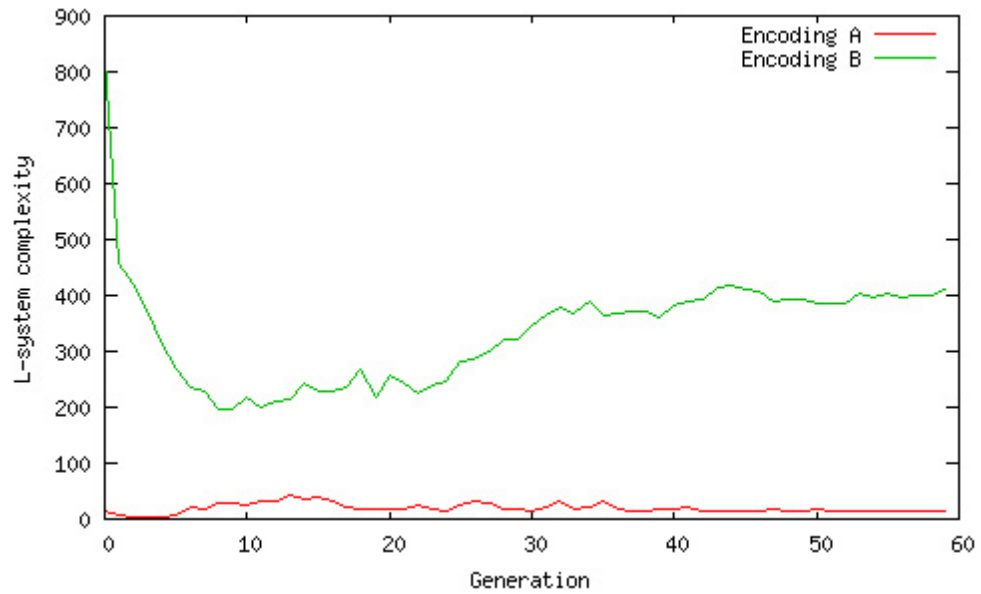


Figure 6.2: Average L-system complexity of population per generation, between two L-system encodings

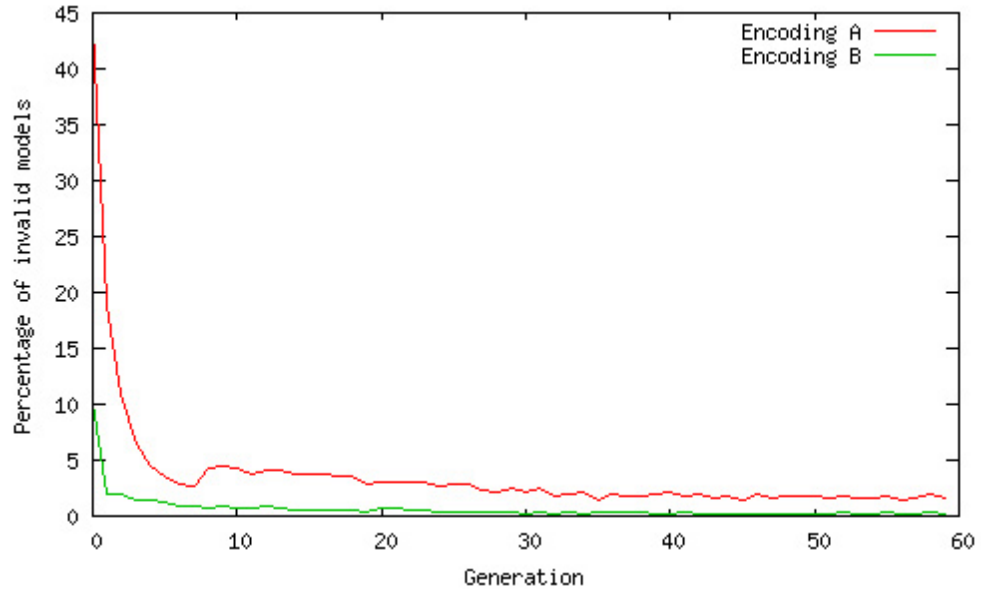


Figure 6.3: Percentage of invalid models from population per generation, between two L-system encodings

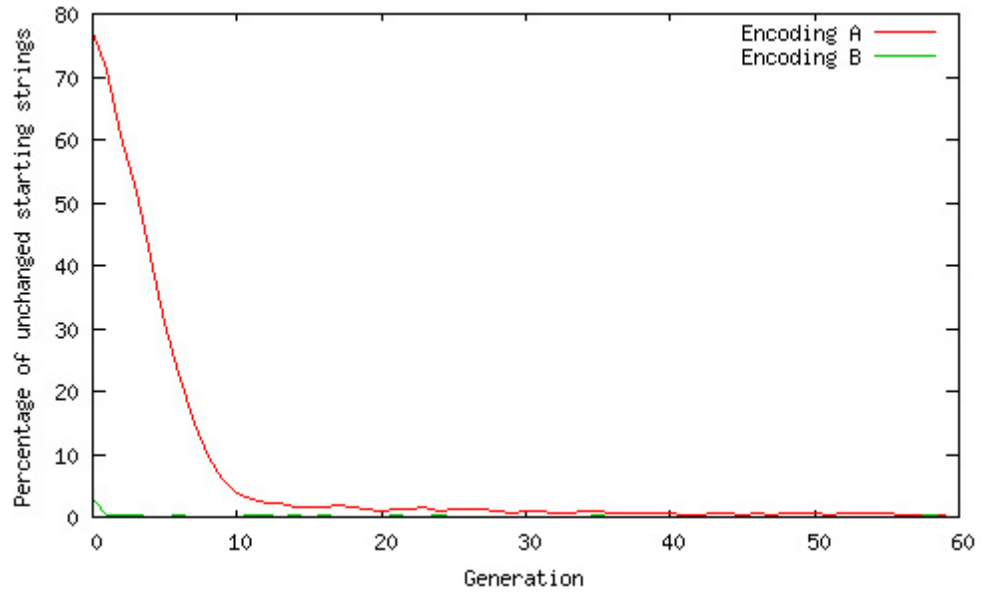


Figure 6.4: Percentage of unchanged starting strings from population of L-systems per generation, between two L-system encodings

## Chapter 7

# Single-objective Experiments

### 7.1 Introduction

The goal of this experiment is threefold – to demonstrate the effectiveness of GP for each fitness function individually, to show the success of the advanced encoding on simple problems, and to discuss the visual appeal of models generated using only a single fitness function. As mentioned in Chapter 5, there are eleven fitness functions available in the system. In addition, the distribution-based fitness functions each can measure two different distributions. For this experiment, only one of the distributions is used, measuring the difference between adjacent face normals in a model. This was chosen due to the similarity between the performances of the two distributions discovered during initial study of the distribution-based fitness functions. The parameters for this experiment are the same as those in Table 6.1.

### 7.2 Results

It was hypothesized that GP would succeed in evolving a population to satisfy each fitness function individually, by having the best of each generation approach the target fitness as well as having the population converge close to the best after some time. Sixty generations was chosen for the time span, as it was deemed sufficient enough to demonstrate convergence for all fitness functions. For a select few, however, thirty generations was used for reasons to be explained later in this section. The performance results for each fitness function can be seen in Figures 7.1 to 7.6. On the graphs, the best represents the average of the highest-scoring individuals of each run, for each generation. The average represents the average fitness value of the population of each run, for each generation. The targets for each fitness function are also shown. A good performance can be seen on graphs which have an average curve approaching the best curve, and a best curve approaching the target.

Ten runs were executed for each fitness function.

As seen in the graphs, GP excelled at evolving a population of models to fit the fitness criteria for each function individually. In most cases, the target value was reached by generation 60. The DFN and  $1/f$  noise functions are typically more difficult to reach a target value of zero, though any fitness less than one is considered acceptable. Conversely, maximizing entropy is equally difficult, and it is uncommon for an individual to have an Entropy of anything higher than 4.5. A few fitness functions – such as surface area, symmetry, volume, mean and standard deviation – are evidently far simpler for GP to reach the target than others, as GP most often generates an individual with the target fitness within the first few generations. Although this may seem as if these fitness functions are inappropriate as targets for GP, the results merely suggest that these functions instead may be more suitable for multi-objective problems, paired with other fitness functions. A symmetry of one, for example, is the most easily-achieved target of the suite of fitness functions available, but would likely produce interesting results when paired with other fitness functions, such as DFN.

The performance of the complexity function deserves a separate discussion, as it behaves slightly differently than the others. It was discovered during initial empirical study that maximizing the complexity function typically resulted in memory issues, which stemmed from the production of excessively complex L-systems in GP. To remedy this, a few measures were taken. First, the logarithm function was added to the complexity function to reduce the reward given for increasing complexity and surface area (see Chapter 5). During a run, the range of iterations could also be reduced to limit the capacity that each  $\omega^i$  could grow. Finally, reducing the generation span also seemed to reduce the number of memory issues. As the last two solutions impose restrictions on the L-systems produced, another more acceptable alternative is a multi-objective approach, pairing the complexity function with other fitness functions. In most cases, an increase in complexity is detrimental to the success of all other fitness functions – as other functions were found to succeed with lower complexities – and so this approach would hypothetically reduce the chances of the complexity function being maximized to the extent it would, if used on its own.

Another important issue to address is the large distance between the best and average curves at the time of convergence. In most cases, this is largely attributed to the difficulty of the fitness function, as seen in the DFN and  $1/f$  noise functions. In a few other cases, it is due to the number of invalid models generated, which have a dramatic effect on the population's overall fitness. The complexity and unique normals functions tend to produce large, complex models. Due to the restraint on the run-time of GP, many complex models often 'time-out' during the parsing and rendering stages, resulting in being assigned the worst possible fitness for each applicable fitness function. This affects the overall average population fitness significantly, and therefore results in the large distance between the two curves. This problem is largely unavoidable when these fitness functions are used individually, but as previously mentioned, a multi-objective approach is expected to reduce this effect.

Figures 7.7 and 7.8 show the rendered results of the highest scoring individual from all ten runs, for each fitness function. These images are meant to demonstrate the types of models created using each fitness function individually, while simultaneously showing the faults of each function as well. For each fitness function, the best individual and its fitness and L-system used to produce it are shown. In most cases, each fitness function typically produces models with similar characteristics – most of which excel with that particular function but fail with all others. A perfect example of this is the model generated using surface area, seen in Figure 7.7. While this model scored high for Surface Area, it would undoubtedly fail for symmetry, dimensions, unique normals, volume and any of the distribution-based functions. It may, however, succeed for the complexity function as its surface area is large and its L-system complexity is visibly so.

Many of the models are erratic and shapeless, which is likely due to the lack of other constraints on the models' shapes and size. This is apparent in the models produced using  $1/f$  noise and entropy. In other models, simplicity is chosen by GP, as is seen in symmetry, volume and complexity. The most surprising is the model generated by the complexity fitness function, which would be expected to be visibly complex. By examining the production rules of its L-system, it is obvious that the actual L-system complexity is high, but it can also be seen that the system basically draws a straight line to the bounds of the voxel-space. The high fitness value can be attributed to the enormous surface area, which was made possible by the gravity wells. These simple models are perfect examples of GP's capability to 'take shortcuts' in finding solutions.

### 7.3 Conclusion

From these results, it can be safely concluded that while GP has been shown to excel in evolving L-systems to satisfy the criteria for each fitness function individually, the visual results themselves are generally uninteresting. A multi-objective approach that combines these fitness functions in small subsets might be more preferable, and would potentially remedy some of the issues of certain fitness functions when used individually described in this section. More complex fitness criteria may also result in the production of more interesting models.



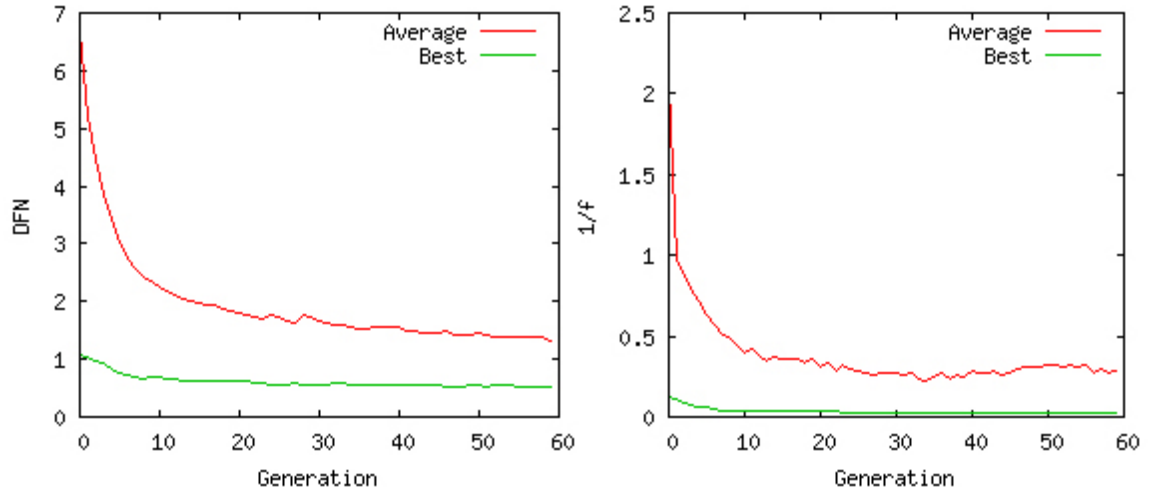


Figure 7.1: Single-objective runs of DFN (target 0) and  $1/f$  noise (target 0), averaged over 10 runs

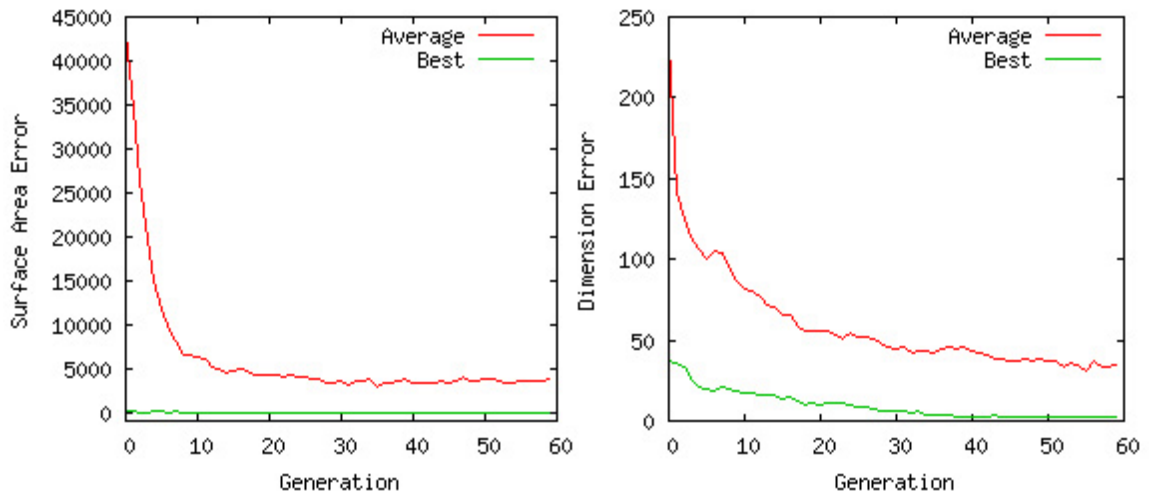


Figure 7.2: Single-objective runs of surface area (target 52500) and dimension (target dimension  $[40,120,80]$ ), averaged over 10 runs

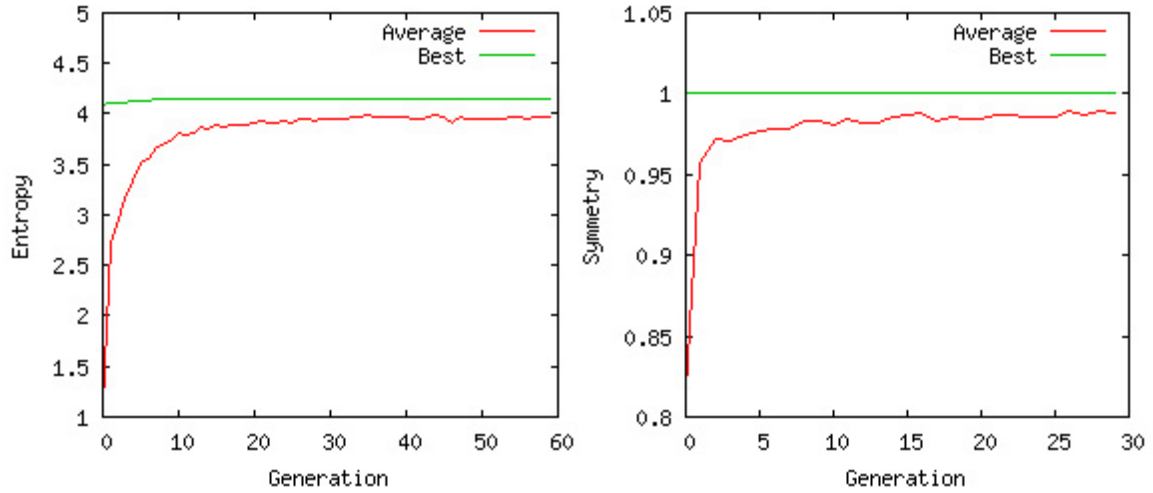


Figure 7.3: Single-objective runs of entropy (maximize) and symmetry (target 1), averaged over 10 runs

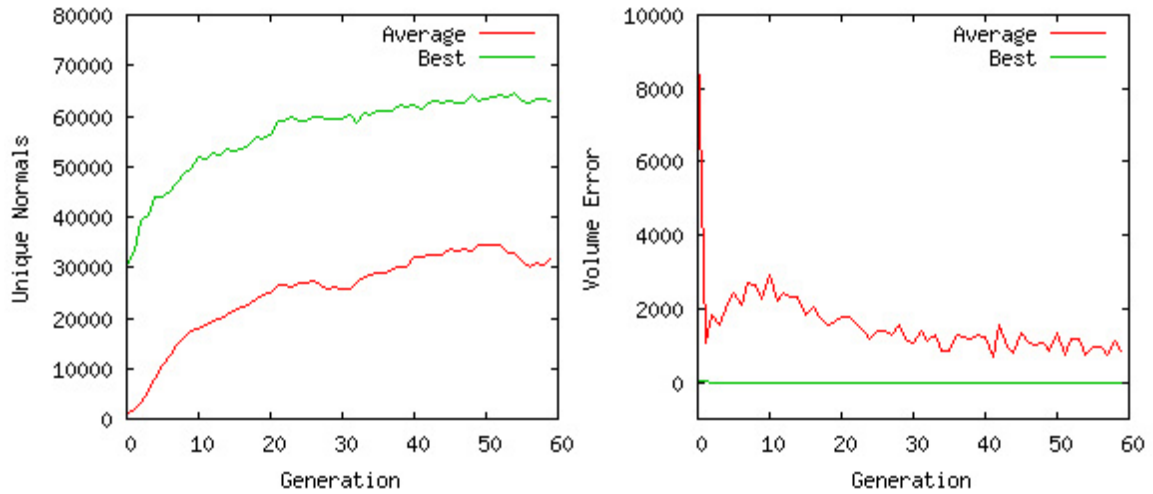


Figure 7.4: Single-objective runs of unique normals (maximize) and volume (target 4500), averaged over 10 runs

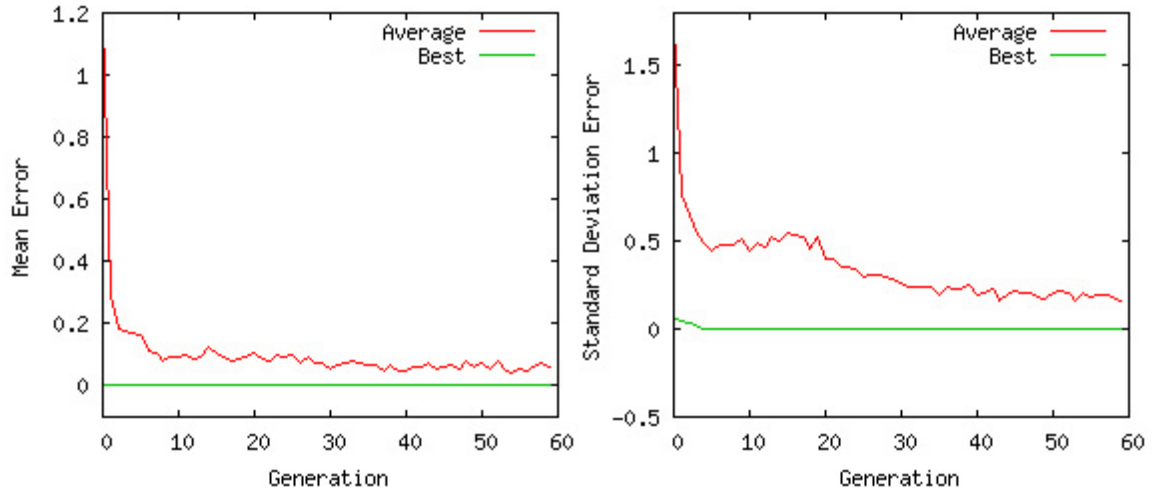


Figure 7.5: Single-objective runs of mean (target 0.025) and standard deviation (target 0.25), averaged over 10 runs

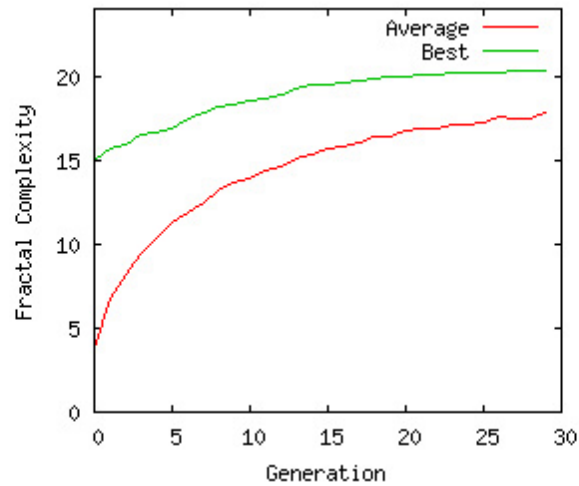
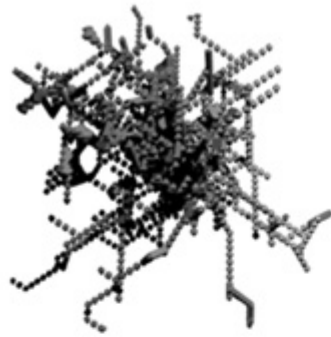


Figure 7.6: Single-objective runs of complexity (maximize), averaged over 10 runs



Surface Area : 52501

Iterations : 3  
 Start : &CCA[[[CC@]]]  
 Rules :  
 B > CB\*AA  
 A > @C@CC/C+C  
 C > \*CBCC-CA[C]



Symmetry: 1

Iterations : 3  
 Start : [//C]A@[+[+]  
 Rules :  
 A > [\*A\*\*A\*[CC]]  
 C > \*B[[CCC][C]]



Dimensions: 1 (error)

Iterations : 3  
 Start : A-CC[CCCC/  
 Rules :  
 A > CCCC[@C[[\*]]]  
 C > CA[+[+CC+C]]



Unique Normals: 73545

Iterations : 4  
 Start : +C[CC[CC+C]]  
 Rules :  
 B > CC+&CBC&  
 A > GC&A&CCCB&  
 C > D&CCD-&A+C



Volume: 4500

Iterations : 3  
 Start : \*B\*  
 Rules :  
 C > [BCBC/]D[[]]  
 B > CC/hCCC[CC]



Complexity: 20.6899

Iterations : 4  
 Start : CCCCCCCCCC  
 Rules :  
 B > CCCAC  
 A > [gACCgACCC]  
 C > CCAgCCCCC

Figure 7.7: Rendering of models with highest fitness for the model constraint fitness functions and aesthetic fitness functions

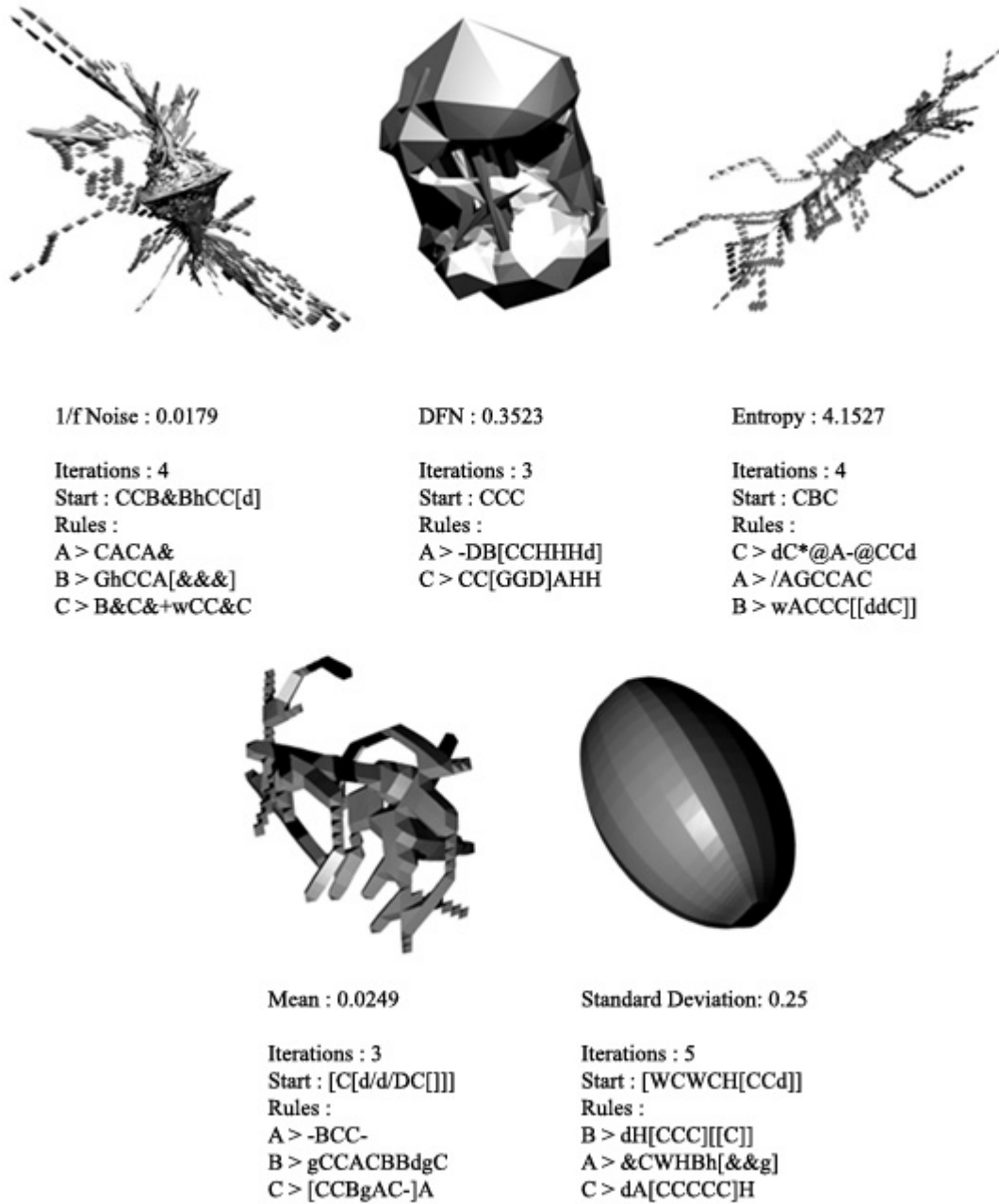


Figure 7.8: Rendering of models with highest fitness for the distribution-based fitness functions

## Chapter 8

# Multiobjective Optimization Strategy

### 8.1 Introduction

The previous experiment showed the results of a single-objective approach to the evolution of aesthetically pleasing models. Although the results proved that each fitness function was able to reach their target fitness over a suitable number of generations individually, it also showed that the use of a single fitness function most often resulted in the production of uninteresting models. A MO approach is expected to be more suitable to the problem, as combinations of the aesthetic measures and model constraint functions would undoubtedly contribute more together than individually.

The goal of this chapter is to examine the effectiveness of a MO approach to the problem, by examining both the visual results of the runs as well as the success of each fitness function's ability to reach their target in a MO environment. It is expected that this strategy will yield more interesting results than those of the single-objective runs, though this conclusion might be subjective. Each fitness function is also expected to reach their target fitness areas within the generation-span provided, assuming that the fitness functions chosen are independent of one another – the score of one is not dependent or correlated to the score of another. It was previously mentioned that it is typical for the GP to favor simpler models when using most aesthetic measures individually, and that these measures would benefit greatly from a pairing with a model constraint fitness function. A low DFN, for example, is easier for the GP to achieve when the model is composed of fewer faces. Therefore, in order to enforce the production of more complex models, the use of model constraint functions are suggested. In addition, other aesthetic measures grouped together might achieve similar effects.

## 8.2 Comparing Summed Rank and Pareto

A MO problem involves the use of multiple feature tests, which must be optimized together. The feature targets – or fitness targets – remain similar to those of single-objective evolution, though by using MO an attempt is made to reach all targets simultaneously. In order to assign a single fitness to an individual meant to be ranked within the population, different strategies are used.

Chapter 2 details the evaluation strategies behind Pareto and the summed rank methods, both of which are used in this experiment. Both methods have their advantages and disadvantages. Normalized summed rank is more commonly-used for problems with higher dimensionality, and most often results in a more diverse population make-up than that of Pareto when diversity strategies are used. Pareto ranking, on the other hand, often produces outliers, which are usually undesirable for fitness-driven evolution. These outliers are a direct result of Pareto’s notion of domination, in which some of the highest-ranked individuals remain as such as long as they are undominated by any other individual. These high-ranked individuals often boast strong scores for one or two features, while the remaining scores suffer. This effect is especially problematic for MO in this thesis, as all fitness functions are required to succeed simultaneously.

For this experiment, three fitness functions were chosen – DFN, complexity and symmetry. Although these three chosen fitness functions are all considered to be aesthetic measures, it has been observed that the complexity function tends to produce larger models when the target is high – indirectly suggesting it is also a model constraint function – which is expected to influence the ability of the DFN to reach lower target values. This should greatly increase the difficulty of the problem. The symmetry function was chosen due to its simplicity in achieving its target score of 1.0 in simpler models, which should result in a similar effect as the DFN function mentioned earlier. As with previous experiments, ten runs were executed using the parameter settings seen in Table 6.1. The target fitness for DFN was 0.0, symmetry was 1.0 and complexity was to be maximized.

The quantitative results of the experiment are to be analyzed by various statistical methods, illustrating the effects of each MO ranking method on the GP evolution as well as the distribution of feature scores in the final populations. Three tests are to be executed using different ranking methods – summed rank, Pareto and random search. It is expected that the summed rank method will prove more appropriate for this MO problem, as the evolution will be more fitness-driven and result in fewer outliers, which are undesirable. It is important to note that this experiment is not a critique of Pareto ranking, but instead a study of the appropriateness of its use in this particular application.

### 8.3 Results

There are two aspects of the experiment that will be considered here – the performance of each MO method with respect to each feature score, and the diversity of the population. The performance curves for the runs can be seen in Figures 8.1 to 8.3. The graphs show the best and average population fitnesses, averaged over the ten runs. Each graph shows the performance of the GP for a fitness function individually, for both MO methods. An ideal performance would involve the average curves for each fitness function steadily approaching the target value, eventually leveling off – signifying convergence in the population. Although the best curves show the average highest-scoring fitness over the entire population and over the ten runs, this curve is much less meaningful as it does not show the other fitnesses attributed to that individual. As would be expected with Pareto ranking, an individual might reach the optimal fitness target for one function, but fail in the remaining.

For summed rank, the average curves for the three fitness functions rise towards the target of all three functions. Pareto was less successful, as only the curve for symmetry increased, and still much slower than the curve seen in summed rank. This poor performance is likely due to the population diversity produced by Pareto ranking. Since average symmetry can be seen to increase and average complexity is low and DFN high, it is expected that there was a large number of individuals with negative complexity, which results in a decrease in model growth over each iteration of the L-system. From observations of the population scores of the Pareto runs, individuals with negative complexity are more likely to produce simple, symmetrical models that garnish high DFN scores, as they are more ‘blocky’ and result in fewer unique face normals. This effect has a similar impact on the performance of the average curve for the complexity function the using summed rank method, though not as heavy.

The results of the final generation for both Pareto and summed rank were subjected to a two-tailed T-test, assuming unequal variances. For this test, the average fitness scores for the entire population were used, one value per run. This test was done for each fitness function individually, and only for the final generation. The results of the T-tests – one executed for each measurement – can be found in Table A.3. Using an alpha value of 0.05, it can be concluded that summed rank outperformed Pareto on average for all measurements in the first generation with a 95% significance. Since the average fitness scores of the random search were worse than Pareto, it can be assumed that the same conclusion can be made regarding random search and summed rank.

The quality of the final populations generated from the two MO strategies is examined, to act as a baseline. In addition, a third search method – random search – was executed using the same parameters and settings as summed rank and Pareto. The random search is simply a GP execution using a tournament size of one with no selection pressure. Figure 8.4 shows the box and whisker plots of the scores obtained from the highest-scoring individuals across all runs, for each MO method. In the charts, each pair of runs along the x-axis (1 and 2, 3 and 4, 5 and 6) represents a population



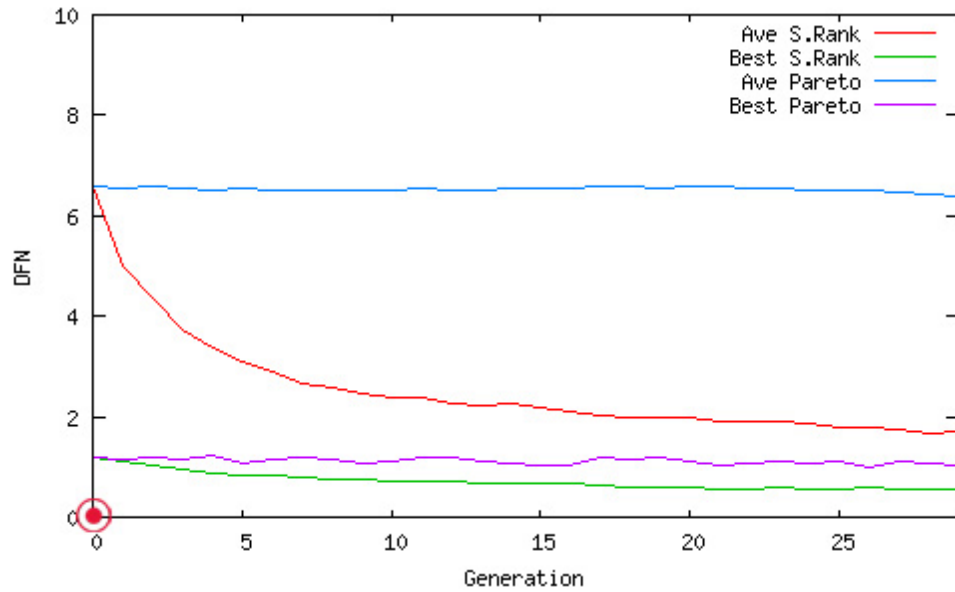


Figure 8.1: Average per-generation population DFN scores for summed rank and Pareto, with target shown

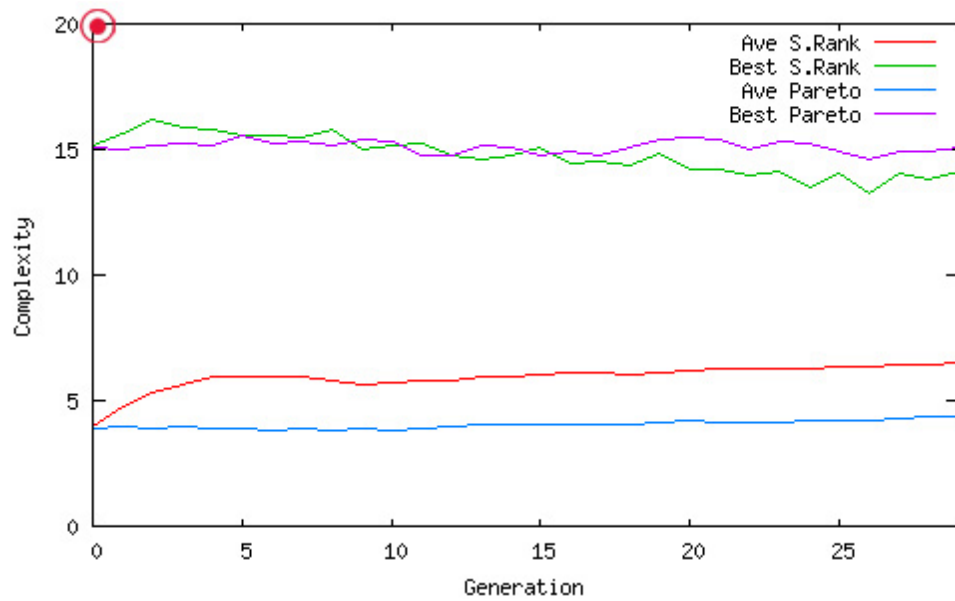


Figure 8.2: Average per-generation population Complexity scores for summed rank and Pareto, with target shown

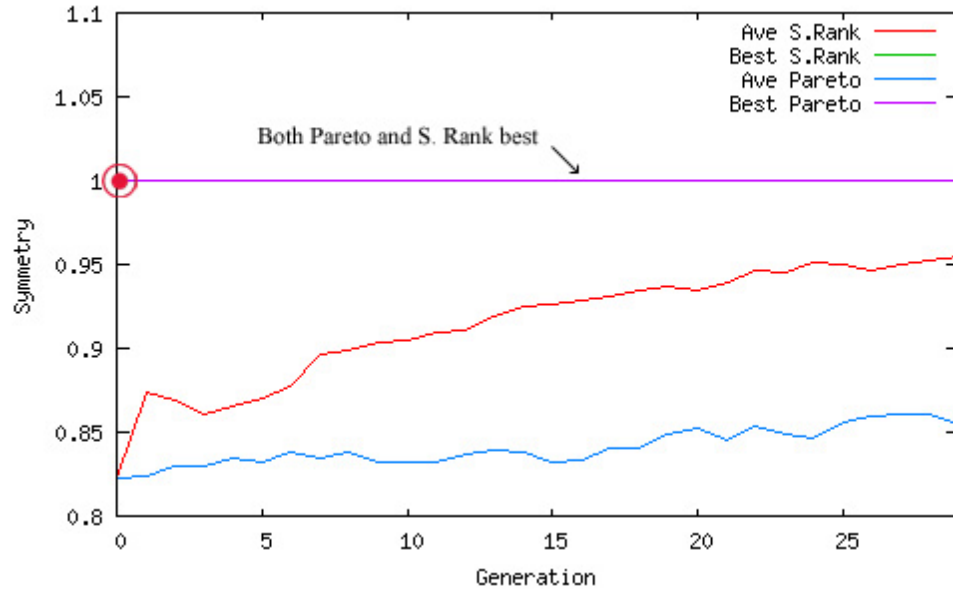


Figure 8.3: Average per-generation population Symmetry scores for summed rank and Pareto, with target shown

generated by a different MO method, with and without duplicate fitness scores. Run 1 in the charts are the ranges of the top 100 individuals (10 from each run) from the runs using summed rank. Run 2 shows the ranges of the top 100 individuals with duplicates removed, displaying only unique individuals. Runs 3 and 4 show the top 100 highest-ranked for Pareto, duplicate and non-duplicate, and runs 5 and 6 show the top 100 for random search, duplicate and non-duplicate. In the event that there were more than 100 top-ranked individuals for Pareto, this number was used instead. In the charts, ranges that cover the target fitness areas closely are preferred, as are smaller quartile ranges.

The fitness ranges for summed rank are smallest for both DFN and symmetry, meaning that the population has likely converged to a small solution space around the target. For both fitness functions, the removal of duplicates makes no visible change to the chart for summed rank. For complexity, the removal of duplicates makes a significant change, which further supports the hypothesis of convergence. Pareto, as expected, displays a wide range of population fitness scores and large quartile ranges. This shows a very diverse population, which is similar – though more diverse – to the results of the random search. Pareto did outperform both other methods with respect to complexity, though from looking at the performance curves in Figure 8.2, these individuals are most likely outliers in the population.

Further support of the diversity of Pareto’s population can be seen in Table 8.1, which shows the average score, overall best score and standard deviation of the population’s fitnesses for each

MO method. Lower standard deviations are preferred, and minimized for all fitness function when summed rank is used.

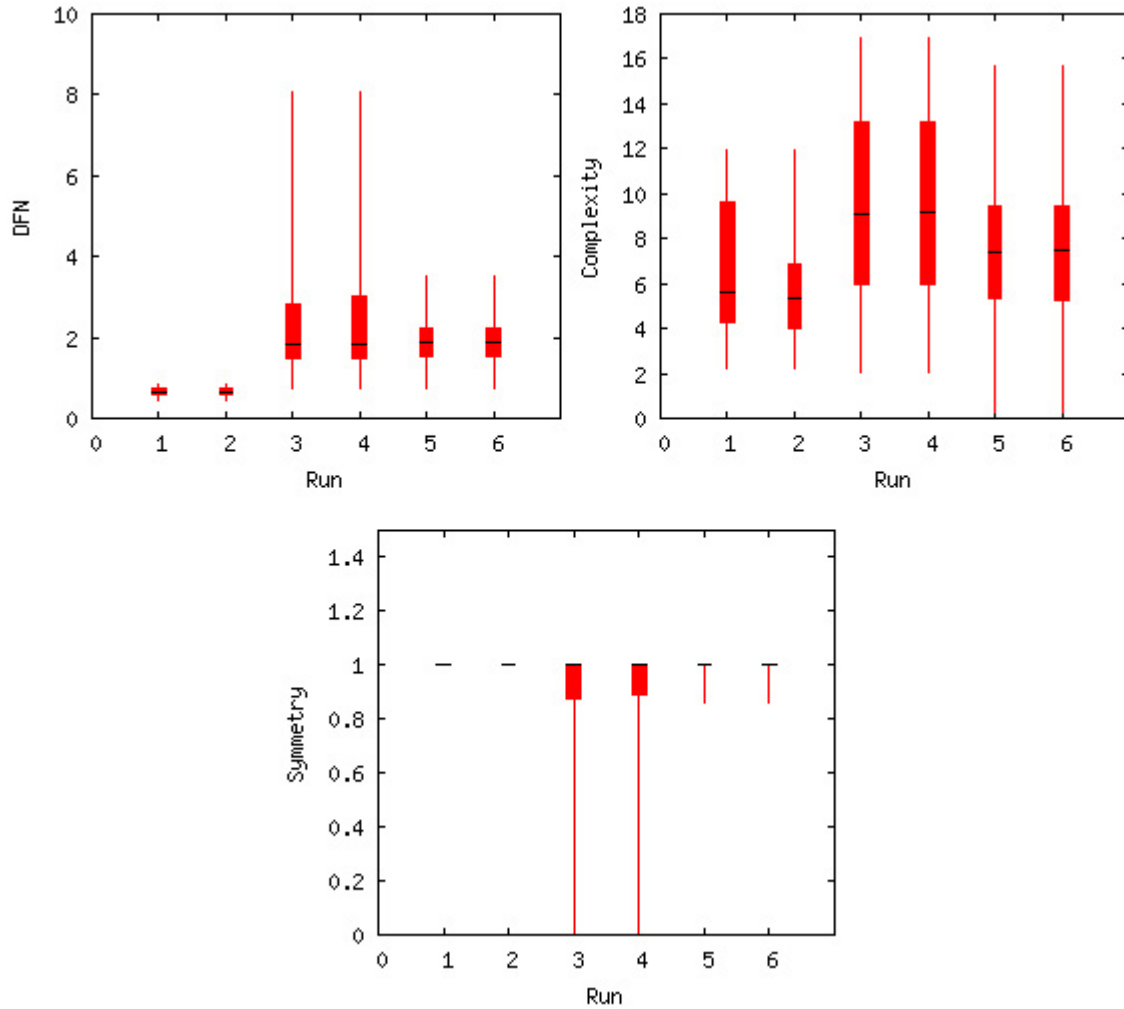


Figure 8.4: Range bars for fitness functions, showing min, max, average and quartiles 1 and 3. Columns represent (1) summed rank, (2) summed rank without duplicates, (3) Pareto, (4) Pareto without duplicates, (5) random search, (6) random search without duplicates.

## 8.4 Examination of Visual Results

Although the visual appeal of the models produced is entirely subjective, it is still a topic worth discussion in order to justify the use of a MO approach. The models chosen from the top-ranked individuals at generation 30 can be seen in Figure 8.5, using the summed rank method. When

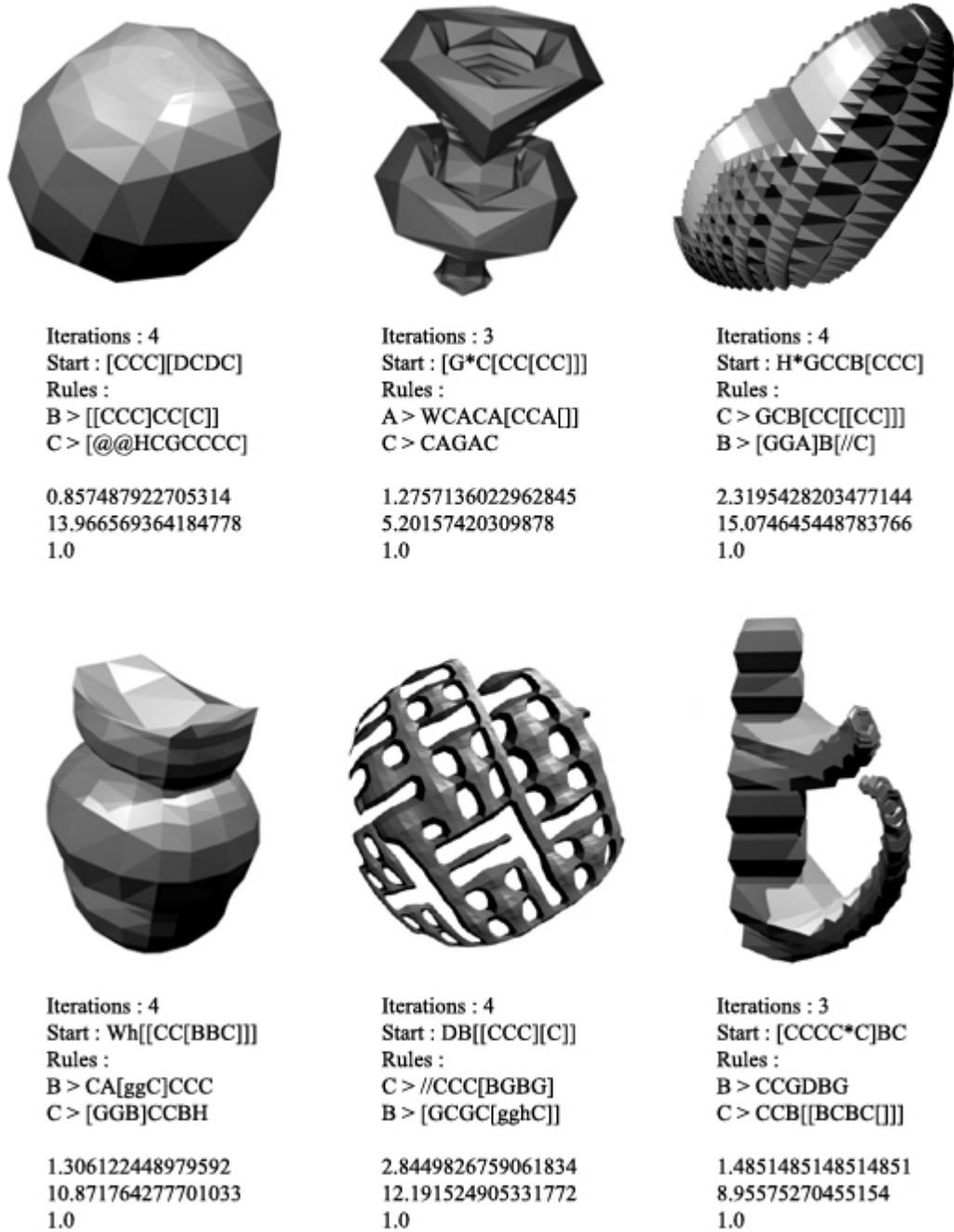


Figure 8.5: Sample models chosen from the top ten individuals of various runs. Shown with L-system and fitnesses (DFN, complexity and symmetry)

Table 8.1: Best and average fitnesses across all runs for the MO experiment using DFN, complexity and symmetry as aesthetic measurements, as well as standard deviation. (1) uses summed rank, (2) uses Pareto, (3) uses random search. Best results are in boldface.

Test	DFN			Complexity			Symmetry		
	$\mu$	best	$\sigma^2$	$\mu$	best	$\sigma^2$	$\mu$	best	$\sigma^2$
(1)	<b>1.7603</b>	<b>0.4279</b>	<b>1.7367</b>	<b>6.5203</b>	17.1326	<b>2.2154</b>	<b>0.9544</b>	<b>1</b>	<b>0.0913</b>
(2)	6.4055	0.7542	2.0979	4.3381	<b>17.1898</b>	3.7098	0.8675	<b>1</b>	0.2662
(3)	5.3049	0.7534	3.9185	4.7128	16.1350	3.6771	0.8338	<b>1</b>	0.2925
target	0.0			max			1		



Figure 8.6: Sample models chosen from a set of rank 0 individuals using Pareto ranking. Shown with fitnesses (DFN, complexity and symmetry)

comparing these to the single-objective results seen in Figure 7.7 and 7.8, it can be safely said that these models are generally more organized, fluid and retain full shape and form. The erratic and rough nature of the single-objective runs is largely due to the fact that there is no further constraint on the model other than the single fitness function. When DFN is only used, the GP can produce a wide variety of shapes and forms to fit the criteria. One of the best examples is using the dimension fitness function – it is easy to satisfy as only the width, height and depth of the model are considered. Any conceivable form within this bounding box can be generated. The grouping of several fitness functions is ideal, as it forces constraint on the GP by narrowing the number of potential 'optimal' solutions, while also increasing the overall solution space and difficulty of the problem.

This increase in problem difficulty is made apparent by looking at the fitnesses of the example models. The DFN scores are much higher in the highest-ranked individuals, which is due to the performance of each model with respect to their complexity score, which is high. The models with the lowest DFN are much more simple, while those with higher DFN are complex and often contain 'holes' in the mesh, as is seen in the third and fifth images.

A few examples were also taken from those on the Pareto front from the previous experiment, as seen in Figure 8.6. These three have lower complexity than those from the summed rank runs, but the same effects can be seen with respect to DFN. Many of the models observed from both MO methods even closely resemble everyday objects, such as pots, jars, fountains, boats and plants. Though it cannot be concluded that the DFN – or any aesthetic function used in this thesis – produced aesthetic models, it can be said that a MO approach using different sets of these functions does in fact constrain the models produced, and limits the number of erratic, shapeless forms produced.

## 8.5 Conclusions

The results of this experiment showed that summed rank's superior performance over Pareto in this thesis makes it an appropriate choice for this research. The overall diversity of the population attributed to Pareto ranking is largely unwanted for these experiments, especially due to the outliers produced. Many – if not all – of the fitness functions have an exceptionally large number of solutions that can satisfy their targets, most of which are visually uninteresting. Individually, it is simple for the GP to produce a population that reaches the target of each fitness function. Summed rank, in this case, does not necessarily result in a completely converged population with no diversity for each run, but does produce a less diverse population than that of Pareto.

The visual results of this experiment have shown that a grouping of two or more fitness functions have the capacity to produce more interesting results, which is due to the constraints imposed by certain functions. Whether or not each aesthetic measure can be concluded to produce aesthetic models in this thesis, their contribution to an overly aesthetic model is definitely worth further study.

## Chapter 9

# Multi-objective with Entropy and DFN

### 9.1 Introduction

The previous experiments outline the goal of producing aesthetic models – or at least more interesting ones – by using a MO approach as opposed to a single objective one. By combining three aesthetic functions – DFN, complexity and symmetry – it can be argued that the results produced are in fact more interesting, organized and detailed. This is largely due to the fact that the fitness functions do not clash – high or low target values for each fitness function individually will not influence the success or failure of the others to reach their own targets. Therefore, each function contributes independently to the overall success of the model produced through EC. For example, as seen previously, a low DFN can be achieved while also maximizing symmetry and complexity.

The goal of this experiment is to examine the effects of using two aesthetic measures that are expected to be so closely correlated that certain target values for one will be unreachable for certain targets of the other, when using a MO approach. More specifically, the GP will be executed with two fitness functions, and the final populations will be compared to see the effect on evolution performance. During initial empirical study, it was suspected that the DFN and entropy measurements are closely correlated, and will therefore be used for this experiment. To prove this correlation exists, four different tests were executed, using different targets for each function. As a reminder, both fitness functions produce values in the range 0-10, and produce a fitness of -10 in cases where the model – or L-system – is invalid. For entropy, a low fitness score typically indicates a simple and uninteresting model, where a high value indicates a more complex and interesting one. The DFN function is the reverse of this. In addition, where the DFN has shown to produce scores within the full range of its expected output, entropy rarely produces a score higher than 5.0.

The fitness targets will be in the range of high (10.0) and low (0.0), and the four combinations of these will be executed – for DFN/entropy, high/high, low/low, low/high and high/low. It is expected that the low/high and high/low tests will yield the most optimal fitnesses – those closest to the target – due to the correlation between the two functions. Also due to the expected correlation, the high/high and low/low tests are expected to result in GP’s favor of one of the functions, while the other’s value suffers. Depending on the favorable fitness function – and its target value – this could result in the production of aesthetically interesting results. For example, during the high/high test, if entropy is high, then DFN will be low, which will result in a more interesting model despite the large fitness error between the DFN and its target.

## 9.2 Results

As with the other experiments, 10 runs were done for each of the four tests, using the parameters in Table 6.1. The number of generations was reduced to 30 to decrease runtime, as convergence has been seen by generation 30 in most previous runs. The results of the population’s average progress for each test can be seen in Figures 9.1 to 9.4. Looking at the graphs for the high/high and low/low tests, it can be seen that the average fitnesses curves of the population for entropy and DFN never approach one another – one is always low and the other high. The graphs also show the overall simplicity of the problem, as the average curves appear to level out early in the run, displaying the convergence of the population. For the high/high test, DFN was chosen by GP as the dominant fitness function, likely due to the simplicity of producing a model with a high DFN. This in turn resulted in an overall low average entropy. For the low/low test, entropy was chosen as the dominant function, and the reverse is seen – also likely for the same reasons. It is interesting to note that in both these graphs, the best curves of both submissive functions are always close to the target, which is due to certain outliers in the population.

The graphs for the low/high and high/low functions are similar but more defined. In the case of the low/high test – which should hypothetically result in the production of the most aesthetic models – convergence is slower, which was to be expected as this is the most difficult target of the four tests. There is a clear separation between the two functions as is present in the graphs of all four tests, which further suggests correlation. The high/low test is the direct opposite, being the simplest target to achieve of the tests. In the case of the high/low test, the most hypothetically uninteresting models are expected to be produced.

The final populations of the ten runs for four tests – 5000 individuals each – are combined in scatter plots in Figures 9.5 to 9.8, showing the distribution of the populations’ fitnesses at generation 30. These plots further support the theory that the two fitness functions are closely correlated. All four plots show a population arc from a low DFN and high entropy to a high DFN and low entropy, with virtually no individuals outside the arc. This suggests a non-linear correlation between the two



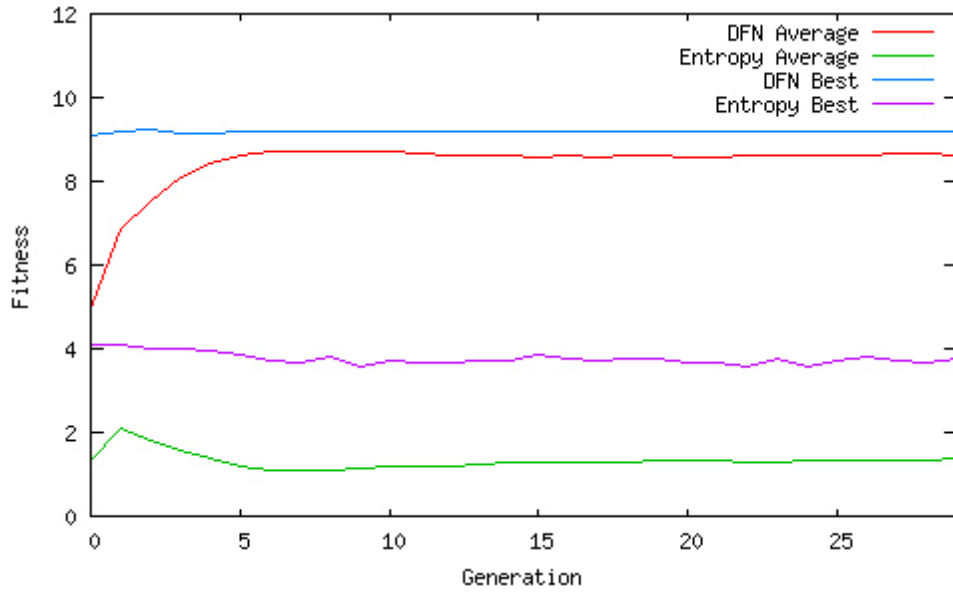


Figure 9.1: Per-generation results of high DFN versus high entropy fitness targets

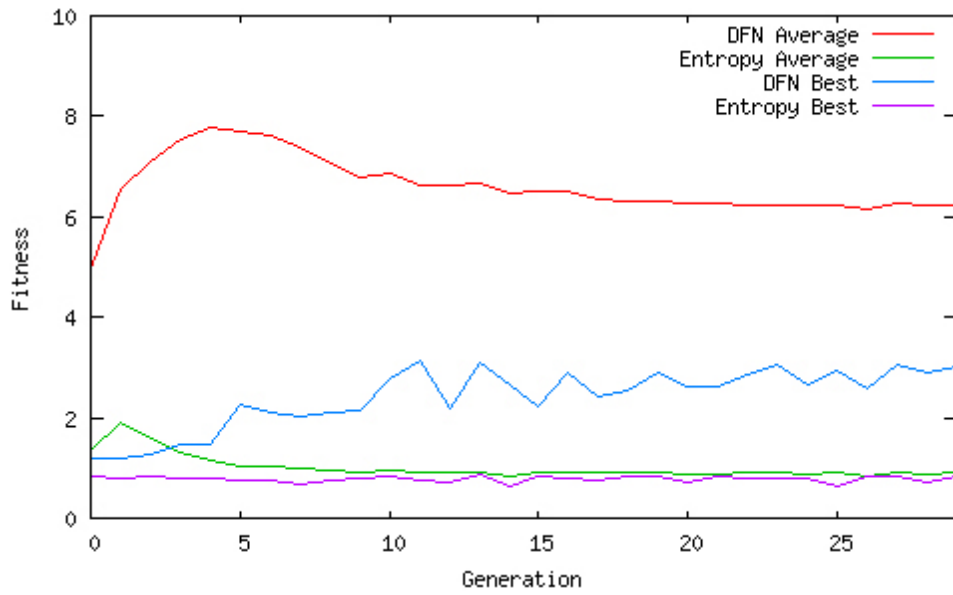


Figure 9.2: Per-generation results of low DFN versus low entropy fitness targets

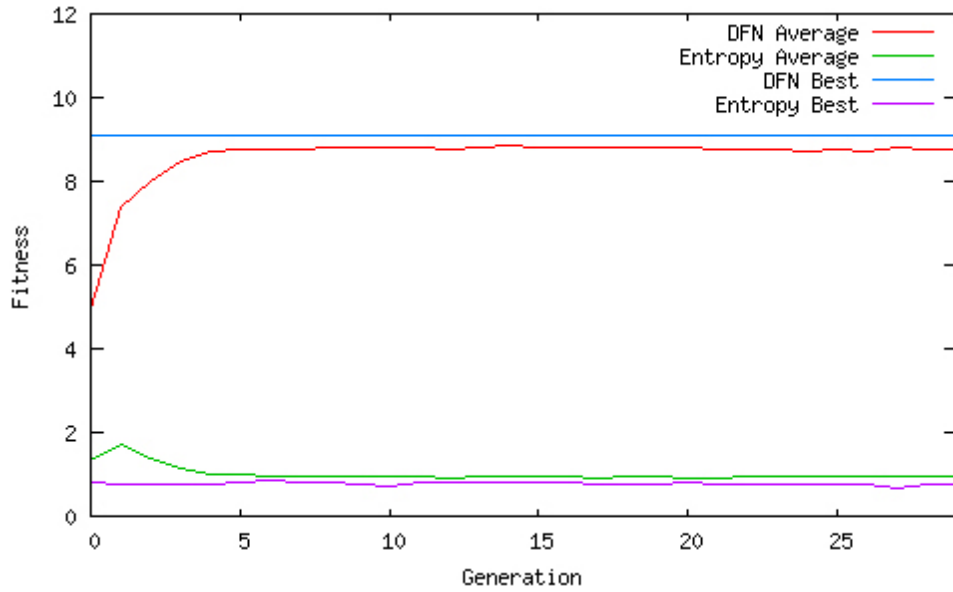


Figure 9.3: Per-generation results of high DFN versus low entropy fitness targets

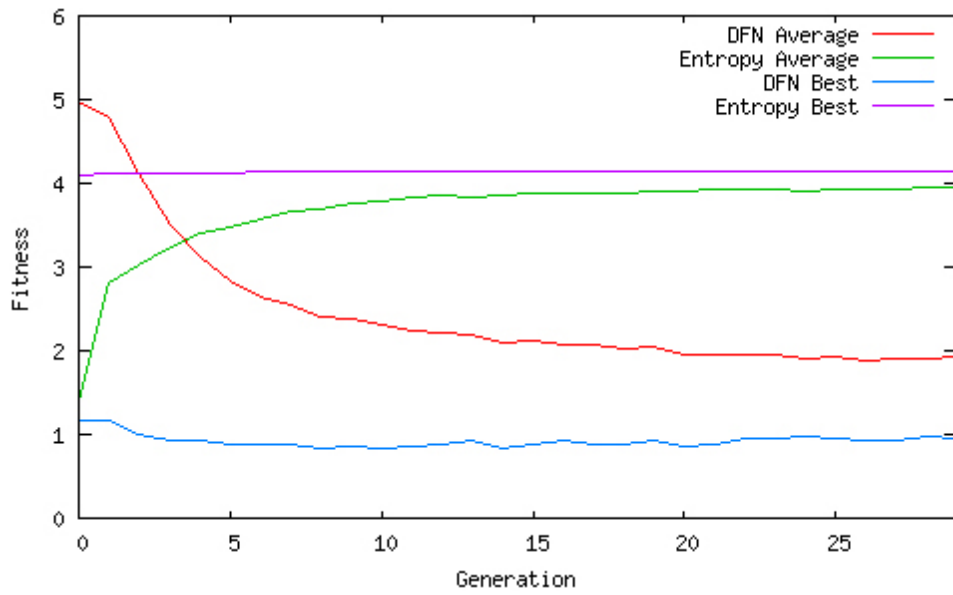


Figure 9.4: Per-generation results of low DFN versus high entropy fitness targets

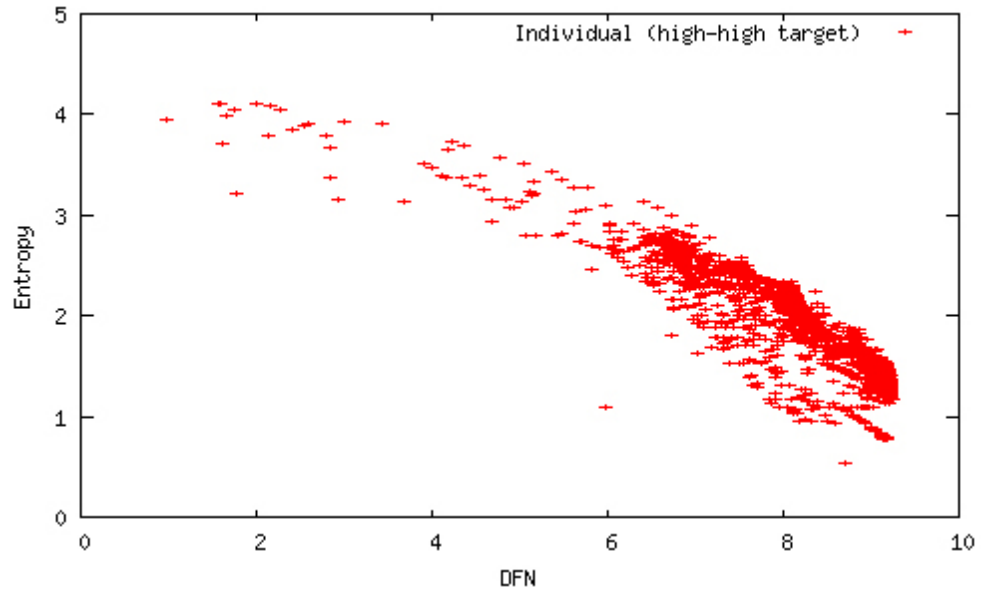


Figure 9.5: Distribution of final population for 10 runs, of high DFN versus high entropy fitness targets

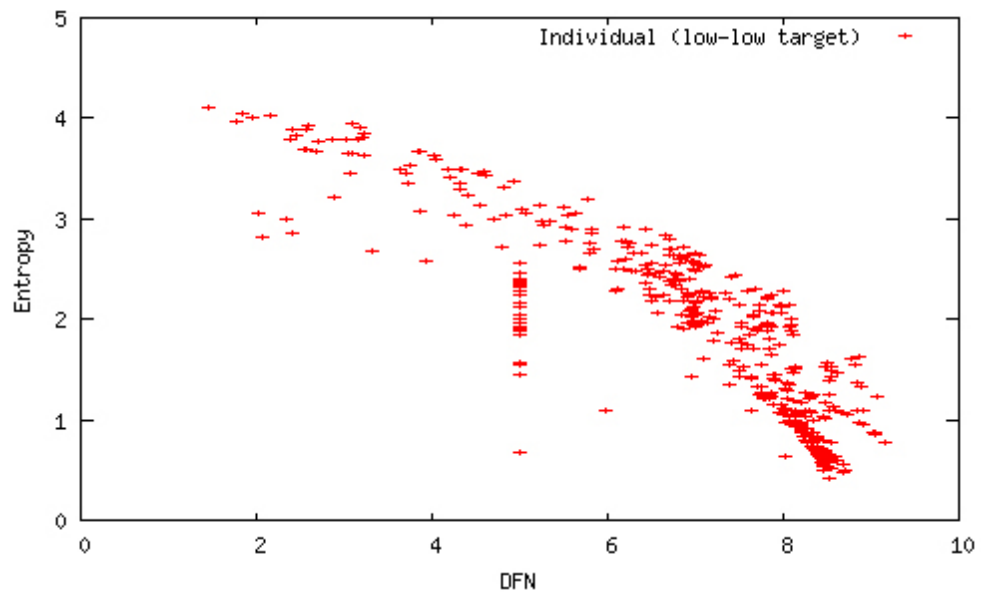


Figure 9.6: Distribution of final population for 10 runs, of low DFN versus low entropy fitness targets

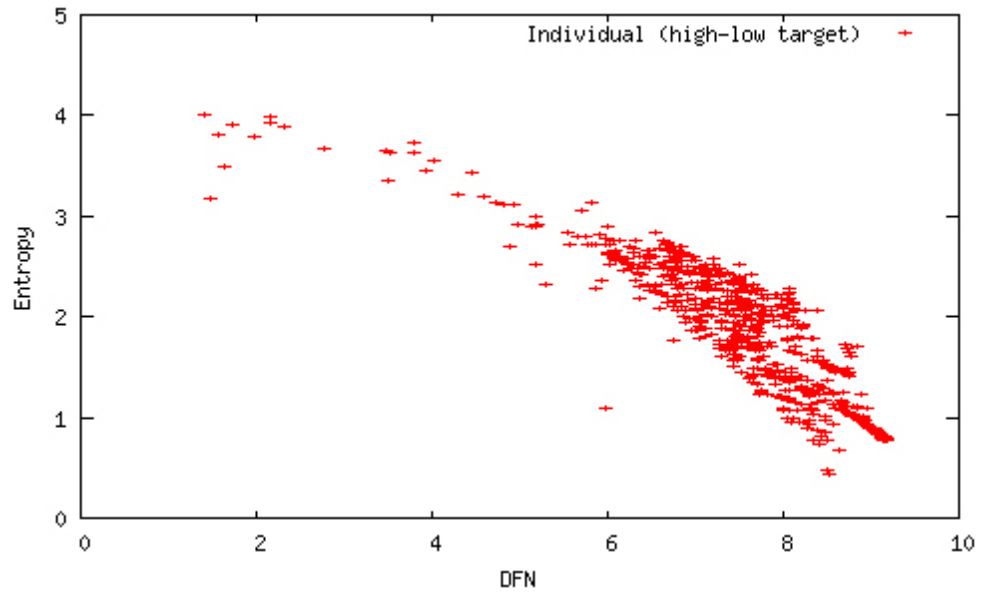


Figure 9.7: Distribution of final population for 10 runs, of high DFN versus low entropy fitness targets

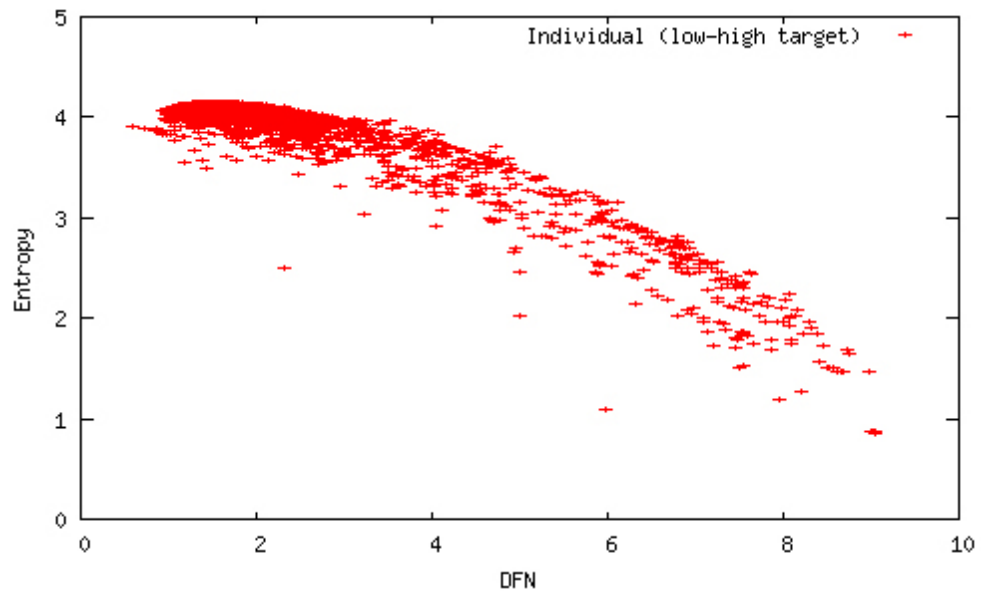


Figure 9.8: Distribution of final population for 10 runs, of low DFN versus high entropy fitness targets

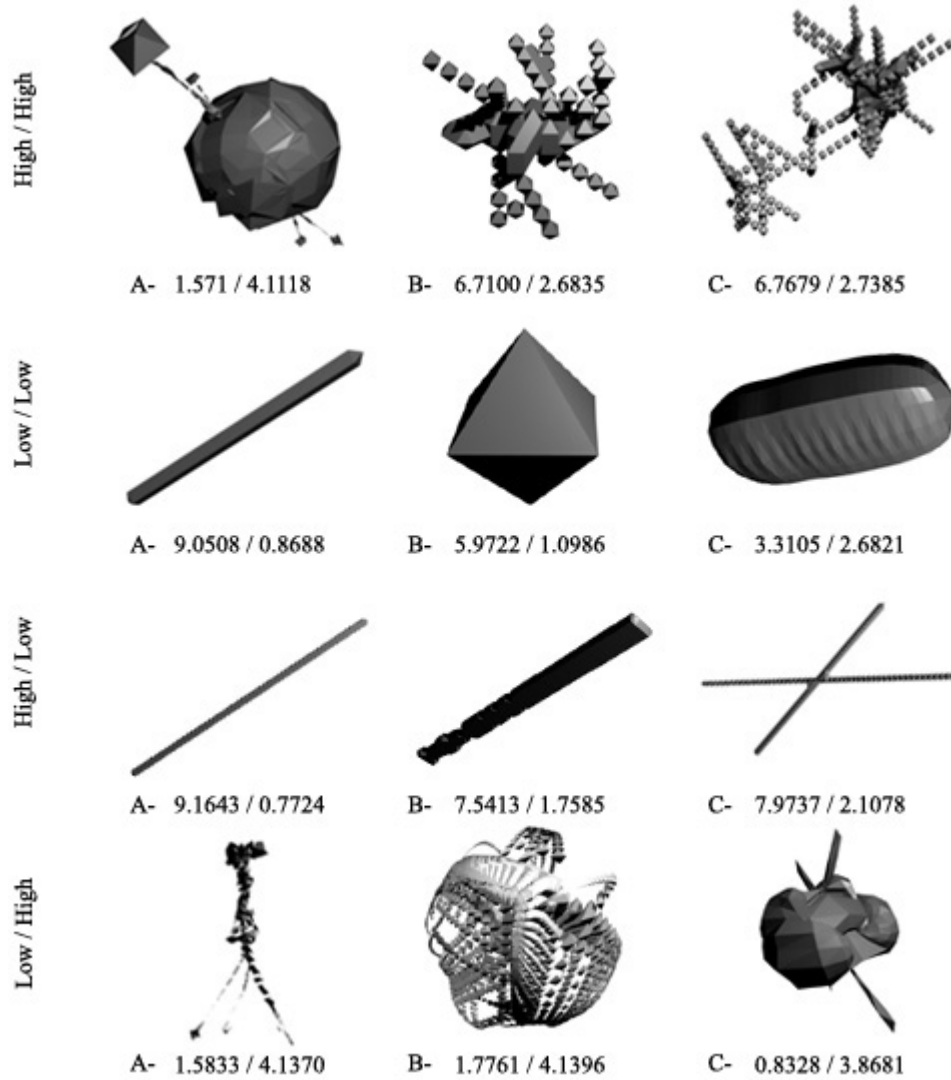


Figure 9.9: Sample models from each of the four tests, with corresponding fitnesses (DFN / entropy)

fitness functions. In the low/high and high/low tests, the population clusters near the target but never reaches the optimal, as is expected. In the other two tests, similar clusters are seen, though the population is more scattered. The four corners of the graphs are seemingly inaccessible, or at least extremely difficult to reach.

The application of the correlation function on the non-invalid population fitnesses for all final individuals (20,000) yielded a correlation coefficient of -0.8994. This value indicates a very high correlation between the entropy and DFN fitness functions. The invalid models were omitted, as their fitnesses are technically not calculated during evolution.

Several samples of models selected from each of the four runs can be seen in Figure 9.9. The majority were chosen from one of the random ten runs for each experiment, from the top ten-highest ranked individuals. There are a few exceptions – models B and C from the low/low test were actually chosen from the lowest ten ranks of two separate runs, in order to display the variations seen with a lower DFN fitness instead of a lower entropy. As mentioned before, the low/low tests resulted in a lower entropy being favored over a lower DFN, likely due to the ease in finding a model fitting this description. In the high/low test, almost 90% of the individuals in the final generation of each run were identical to model A shown in Figure 9.9, which was the most commonly-produced model yielding a high DFN in all runs since the initial testing phases. It results from an L-system that continues to draw a line until the voxel-space boundaries are reached, at which point it resets and loops. Due to this phenomena, the last two models were hand-picked from the lower 10% ranked individuals, in order to show some variety in the models. As can be seen from the images, there is very little alteration from the initial 'stick' model.

The results from the low/high run were easily the most complex, symmetrical and interesting. In addition, the final population was exceptionally diverse, as opposed to the final population of the first three tests. It is interesting to note that from observation of the models produced by this test, models with higher entropy are more 'disconnected', resulting in crystalline figures whose mesh is not entirely complete and fully connected. Models with a lower DFN are the opposite – most often resulting in full-connected, symmetrical meshes.

### 9.3 Conclusions

From the results of this experiment, it can be concluded that there is a definite correlation between the entropy and DFN fitness functions. In addition to this discovery, it can also be concluded the most interesting models are produced from runs with a low target DFN and high target entropy. The effect on evolutionary performance when using these two functions together in a MO problem depends greatly on the targets used. In the situation when the DFN target is low and entropy high, there is a reduced pressure on both fitness functions. The same situation occurs when the DFN target is high and entropy low, but this is an undesirable case considering the models produced from

such a test. There is a high pressure on either function when the target DFN is high and entropy high, and DFN low and entropy low, which was expected.

## Chapter 10

# Miscellaneous Runs

### 10.1 Introduction

This section showcases the rendered results of various experiments with assorted parameters, fitness targets and goals. The purpose of these experiments is to show the potential of the system to produce a wide variety of interesting models, which can be further used as design inspirations, or even imported into external 3D modeling software for further use. Some of these experiments were run with specific goals in mind, such as to create vase-like models or organic, cellular structures. Such experiments showcase the system’s abilities in producing models which not only fit the user’s fitness targets, but can be constrained to fit the user’s conceptual design criteria as well. The results of previous experiments have already shown the success of the system’s ability to reach fitness targets for a MO problem, and so this section is dedicated to showcasing the creative results of interesting experiments.

Each experiment and the general goals behind it are outlined briefly, along with its parameter settings and fitness targets. The majority of these experiments follow the GP parameter settings outlined in Table 6.1, and any alterations to these parameters are discussed as well. The discussion of the results of each experiment will follow. The models from each experiment were hand-picked based on their aesthetic appeal, and therefore were not necessarily chosen from the top-ranked individuals of the population. To show the ability of the system to export its models into other systems, and the potential each model has for expansion, the models in this experiment were imported into the Blender 3D modeling software for rendering. For many of these results, lighting, textures, subsurface division and smoothing techniques were also applied.



## 10.2 New DFN Target

The goal of the first experiment is to examine the results of a run when a DFN target of 3.5 is used. Although a model with a DFN this high does not usually exhibit a distribution curve closely resembling the curve of a normal distribution, it has been seen from the examination of the pre-made models in Chapter 5 that the majority of these models have a DFN around this target. In addition to the DFN, a symmetry target of 1.0 was used as well as a relatively low dimension constraint. The GP parameters stayed the same for this experiment.

The visual results of this experiment varied. Although the majority of the top-ranked individuals were generally uninteresting, one model stuck out from the others. This model resembled a piece of jewelry, and has a DFN of 3.2325 and a symmetry score of 1.0. Despite its place in the top-ranked individuals, it did not resemble the others, which shows the overall diversity of the population after the span of 30 generations and population convergence. This model can be seen on the left in Figure 10.1. Out of curiosity, the L-system which created this model was subjected to a different grammar – the SMS – to look for any similarities and differences in the newly-created model. The generated model, which resembles a goblet, is seen on the right of the ring in Figure 10.1. This new model has a DFN of 5.3323 and a symmetry score of 1.0 as well. Although it is difficult to visualize the full 3D model in 2D, the ring has “jewels” set into it that are evenly-spaced around it. On the goblet, these jewels are instead interpreted by the SMS as patterns, which are also evenly-spaced. Both models are quite similar, but also show the major differences in models created from the same L-system but using two different drawing grammars. To show possible extensions to these models, they were imported into Blender and textures and smoothing techniques were applied. The major differences between the originals and the models which were improved can be seen in Figures 10.1 to 10.4. Blender and other 3D modeling software allows direct access to all model data for the user, which enables virtually any alterations to these models.

Two other high-scoring models from similar runs can be seen in Figure 10.5. Image A closely resembles a bottle, and was textured to reflect this. Its DFN is 3.3498, and symmetry score is 1.0. This form was actually found to be quite common amongst those with DFN scores between 2 and 4, and other variations of this form were seen in the top-ranked portion of the population. Image B was taken from a run using a DFN target of 3.5 and a volume target of 100,000, and had a DFN of 3.5012 and volume of 119,284. Volume was chosen to force the GP to produce larger models, or at least bulky ones. The model appears organic, and although difficult to see in a 2D image, the model is hollow inside, with square openings surrounding the model’s axis. The inside of the model contains a series of pillars as well, likely a by-product of the use of gravity wells. As with Image A, the majority of the population’s models were variations of this form.

Despite the large number of uninteresting models produced using a DFN target of 3.5 for runs in this experiment, there was at least one gem in each population worth keeping. The same can be

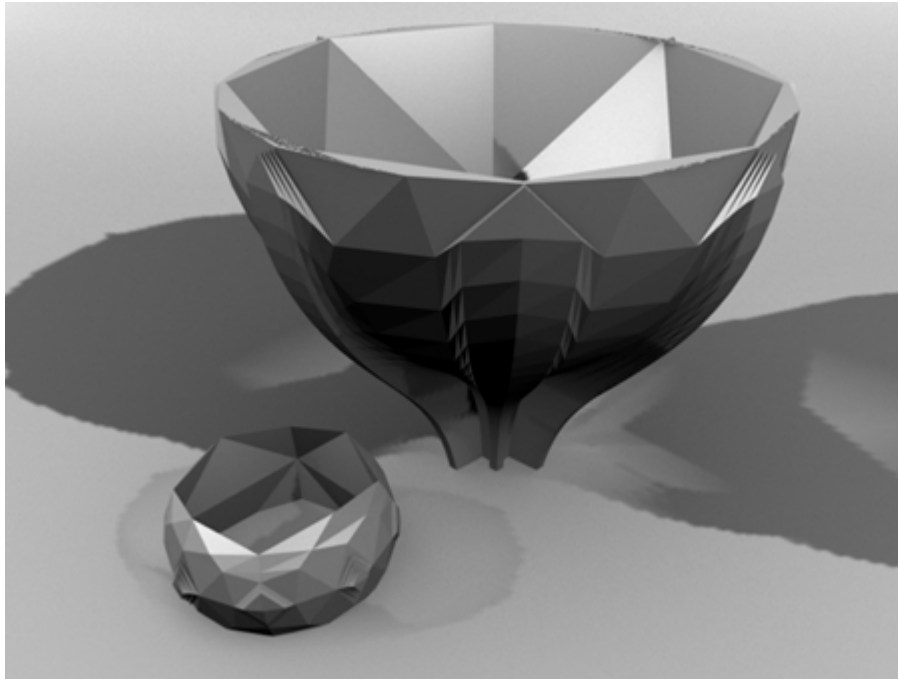


Figure 10.1: Rendering of low-DFN models using no texturing or smoothing techniques

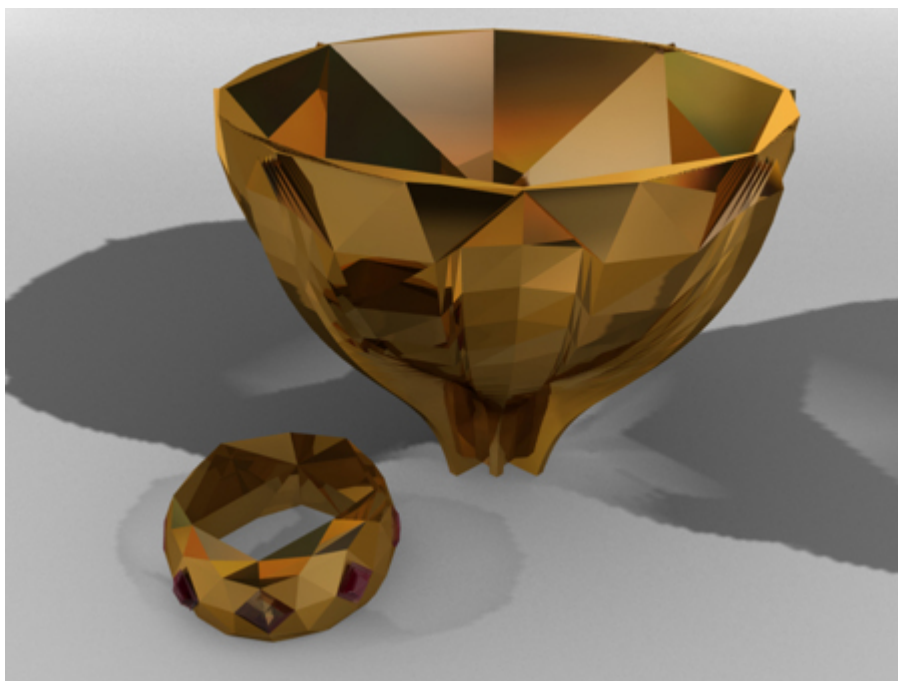


Figure 10.2: Rendering of low-DFN models using texturing and no smoothing techniques

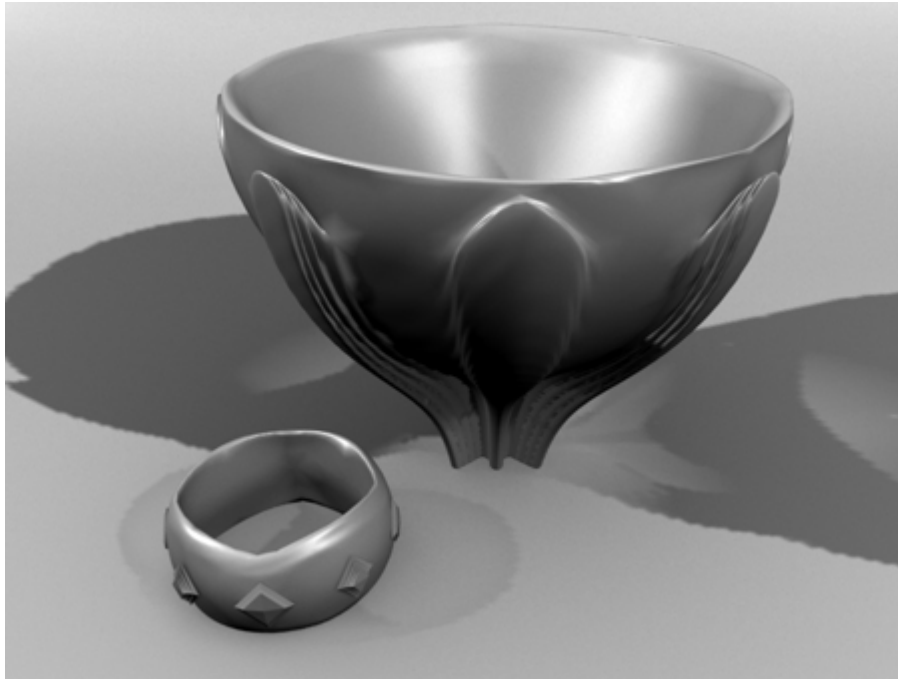


Figure 10.3: Rendering of low-DFN models using smoothing techniques and no texturing

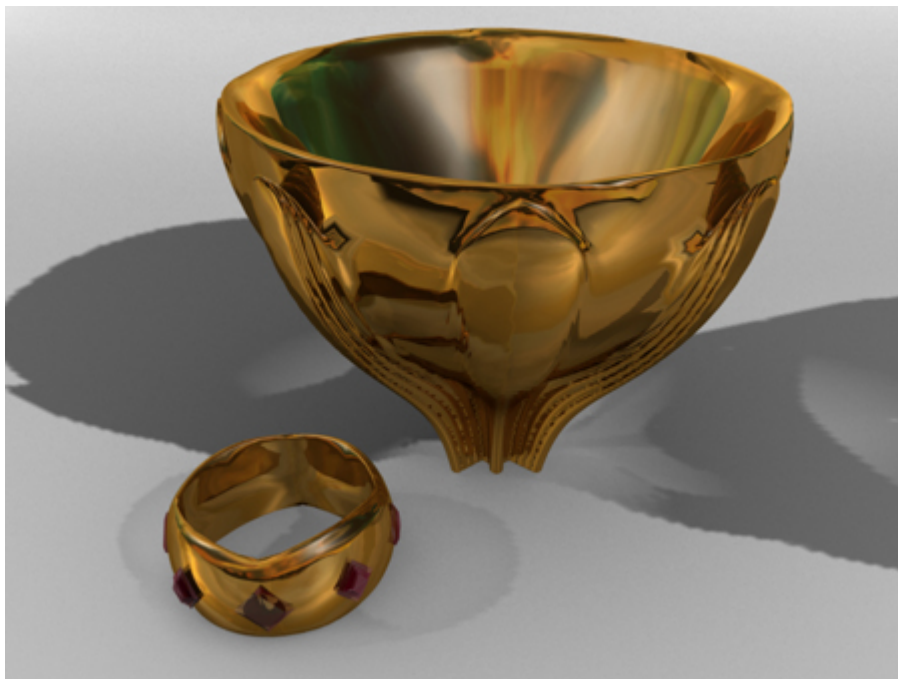


Figure 10.4: Rendering of low-DFN models using texturing and smoothing techniques



Figure 10.5: Hand-picked results of two separate runs using a DFN target of 3.5, and rendered with textures and smoothing techniques in Blender

said about many of the models found in any such experiment, and this does not necessarily reflect the failure of the system. As a design tool it can produce a wide variety of design ideas, and due to the subjective nature of aesthetics and the heuristic nature of aesthetic modeling, no single aesthetic measure will consistently produce widely-accepted aesthetic results.

### 10.3 Organic Forms

This experiment aims to show the system’s ability to produce more organic and fluid forms. While many of the results seen in previous chapters have generally had curved surfaces, they were chosen from the top-ranked individuals in the population due to their fitness scores and with no other real purpose or goal. With this experiment, the exact opposite approach will be taken. The population will be examined and the most visibly-organic form will be hand-picked. This is meant to emulate a real design situation, where a designer will look through many potential design ideas before choosing a personal favorite. The fitness scores, though necessary for evolution, are rather meaningless to a user who does not fully understand their purpose. Instead, a typical user would simply examine the shape and aesthetic appeal of each model before choosing.

As mentioned in earlier sections, models with higher DFN scores typically take on block-like forms, while models with lower DFN scores are usually more fluid, symmetrical and curved. Unfortunately, many of the models produced with a lower DFN are simple and small, as GP tends to favor smaller distributions in order to more easily satisfy the lower DFN target of a run. For this

experiment, a target DFN of 0 will be used, alongside at least one other fitness function which will try to influence a greater model size on the GP.

The first test uses the DFN, a target volume of 50,000, target surface area of 1,000,000 and attempts to maximize the number of unique normals. This large difference in surface area and volume is expected to result in large, thin and complex forms. Maximizing unique normals is usually a difficult task which results in the production of exceedingly complex models. This in turn increases the time for the run substantially. The final results were as expected, and consisted of large models with high complexities and tangled, wire-like forms. From this population, a few similarly-shaped models exhibited organic properties and one of this set can be seen as image A in Figure 10.6. This shape closely resembles a cephalopod, such as an octopus with eight arms evenly-spaced around a centralized body or head. Each arm curves inward at roughly the same angle, which is indicative of the fractal nature behind its construction. The model's fitness scores are 0.9591 for DFN, 45,786 for volume, 2,143,418 for surface area and 8275 unique normals.

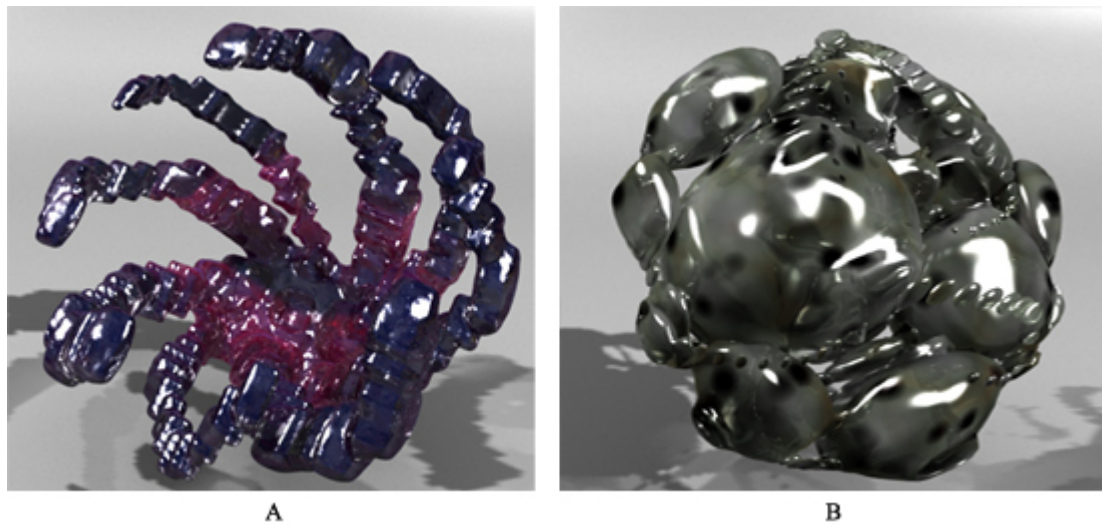


Figure 10.6: Hand-picked results of two separate runs aiming to produce organic forms, and rendered with textures and smoothing techniques in Blender

The second test uses the DFN, a target mean of 0.005, target standard deviation of 1.5 and attempts to maximize complexity. While this set is similar to the fitness functions used in the experiments in Chapter 8, the target mean and standard deviations were added as they have been seen from earlier empirical study to be common in many organic forms. The use of complexity here is similar to the use of volume and surface area in the previous test. As opposed to the previous test, this final population consisted almost entirely of variations of the model seen in image B of Figure 10.6. This model closely resembles a cellular form, or even a cluster of fish or insect eggs.

Its fitness scores were 1.2531 for DFN, 0.0051 for mean, 1.5 for standard deviation and 7.3761 for complexity. These two models were considered the most interesting, with no consideration about their actual fitnesses or ranking within their populations. As with a real design situation, being able to find one great design out of many possibilities makes the whole process worthwhile for the designer.

## 10.4 City Layouts

This experiment outlines a more practical use for this system – generating city layout designs. There is a great deal of prior knowledge and experience that goes into city layout planning. Cost, efficiency and practicality are of great importance when deciding where certain structures should be built, how large they can be and what they are to be used for. Of course, there are situations when this does not matter. Cities used in movies, video games and animations do not necessarily need to fit these criteria, as the placement and cost of each building does not effect the overall purpose of the city or the people who construct it. It exists merely for aesthetic purpose, as a place for video game characters to explore and for protagonists to travel in a movie. This experiment investigates the generation of city layouts as models for the use of design inspiration, or even for dynamic environment generation for the gaming industry.

The SMS grammar is perfect for this problem, as it builds from the ground up using voxels. Gravity wells, which help produce organic, curved surfaces, are not needed for this problem, and so were removed from the L-system alphabet. The choice of fitness functions reflects the problem at hand. Cities generally build farther outward than upward, and so a dimensional constraint was used to reflect this. In order to maximize the amount of surface covered, a surface area target of 100,000 was used. As mentioned in Chapter 5, the use of targets for these model constraint functions help to constrain the model, but will generally be hard to reach exactly. As seen from the results of previous experiments, the DFN will not be appropriate here, as it tends to favor fluid, curved models, which are undesirable for this particular problem. Instead, the entropy function is used with a target of 5.0. Entropy has shown in the previous experiments to produce crystalline, disjointed models, which will help to separate each building in the city. The remainder of the GP parameters are consistent with those in Table 6.1.

Four of the results of this experiment can be seen in Figures 10.7 and 10.8, which were hand-picked from different sections of the ranked population. These models were textured in Blender, but their surfaces left intact. As can be seen from the images, each model varies in size and shape, but all are similar in overall appearance. Their surface areas vary (32003, 33054, 83875, and 117151) but their entropy scores are similar (3.4701, 3.4684, 3.2910, 3.2086). This indicates convergence on the entropy function, which is the most desirable for this problem.

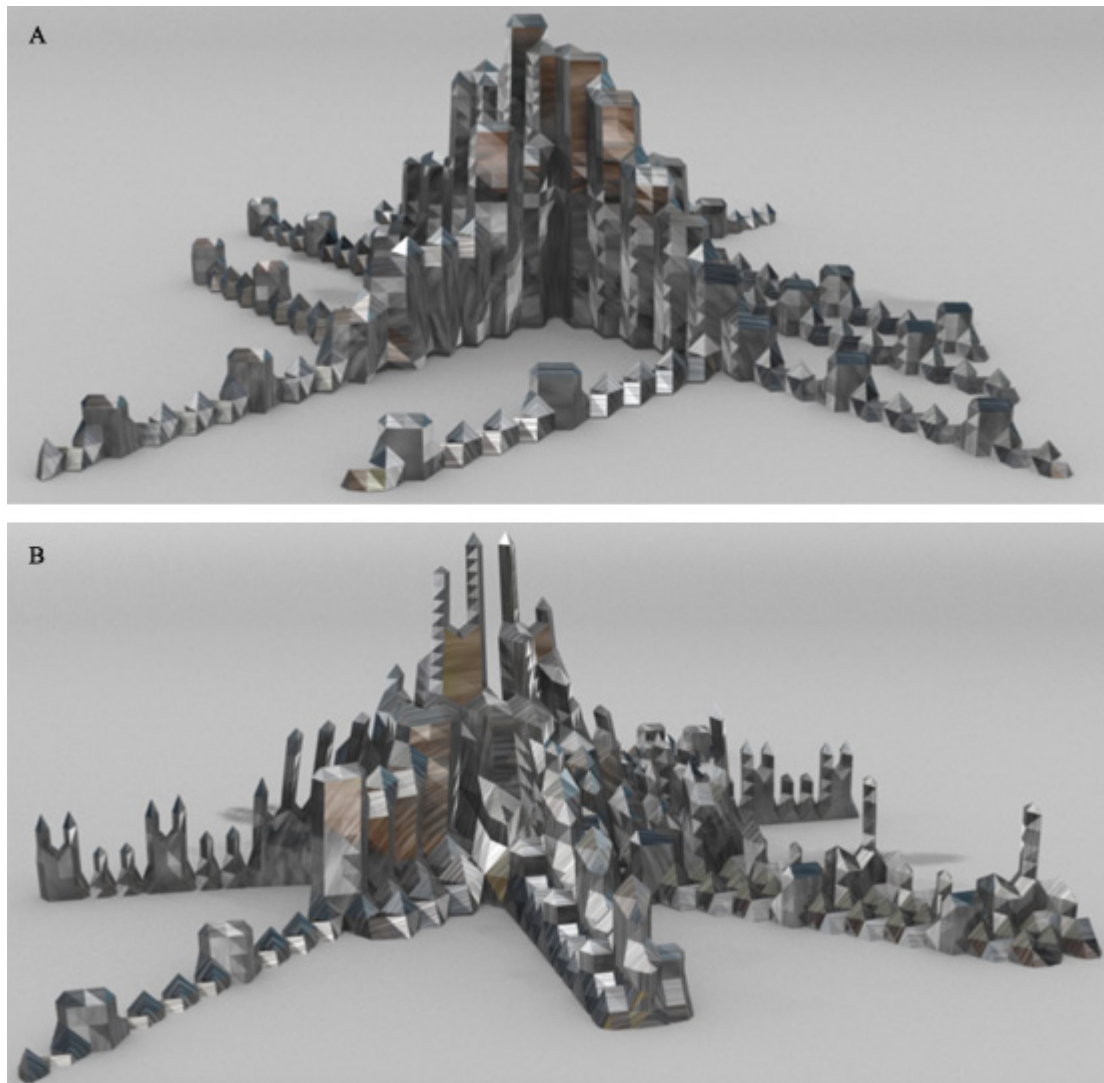


Figure 10.7: Renderings of two models evolved to resemble city layouts



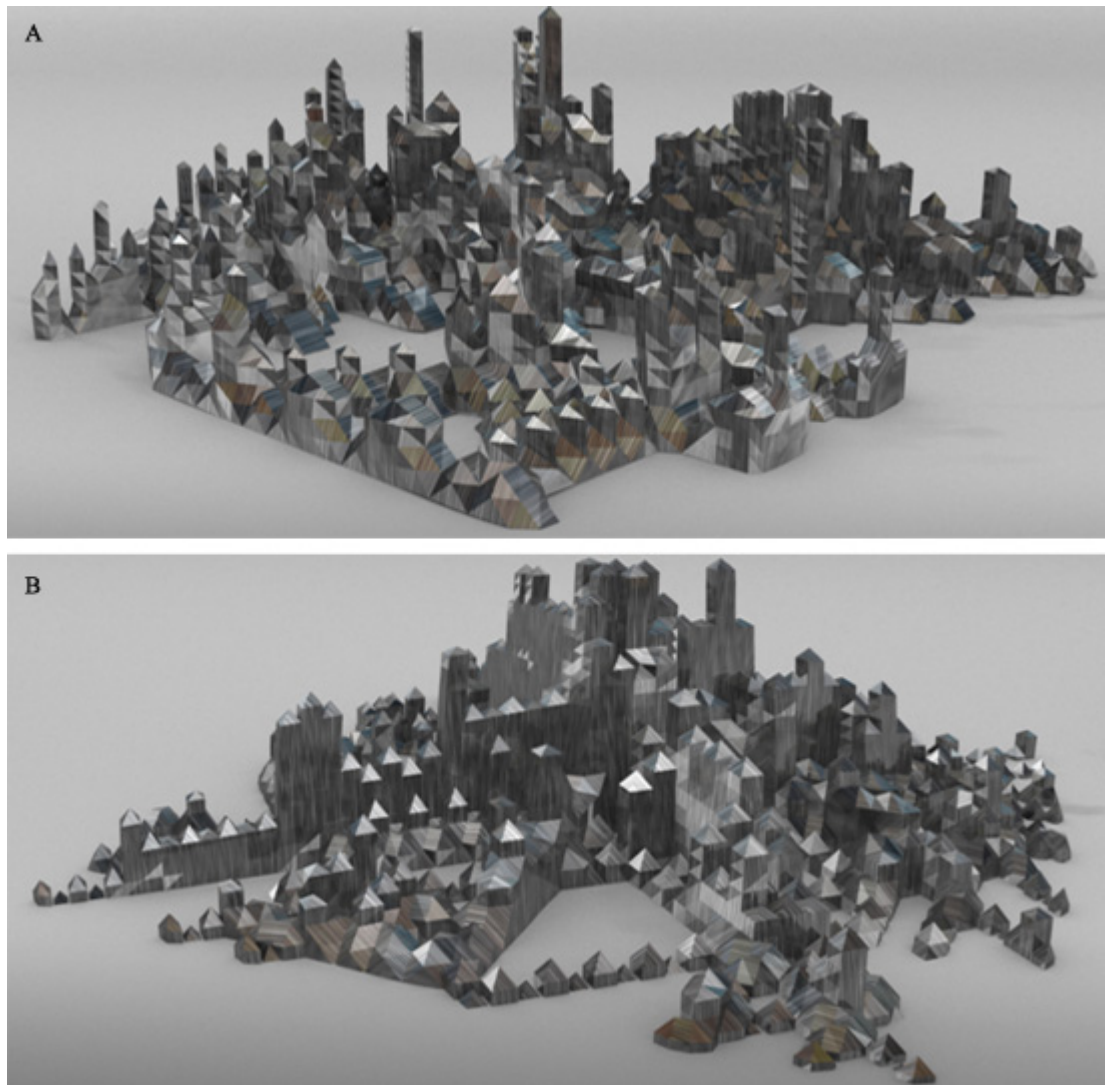


Figure 10.8: More renderings of two models evolved to resemble city layouts



These models are perfect representations of fractal-generated surfaces, which have resulted in a build-up of large buildings in the center of each city, surrounded by smaller, more sparsely organized structures [9]. In most cases, the smaller structures are only a few voxels high. The spaces left by the fractal-drawing could be used for places like parks, museums, or other structures that require a large amount of room. These results are another perfect example of the diversity of models generated using the system, despite convergence. This experiment has shown that the system can be used to generate models for specific goals by properly choosing and tweaking each fitness function and its target, as well as the L-system alphabet and drawing grammar. With the number of possible adjustments and combinations of these settings, the possibilities are endless.

## 10.5 More Results and Conclusions

This final section showcases various interesting results from experiments done during the research in this thesis. These experiments generally had no fixed purpose, but instead were used to test the system's abilities to reach certain fitness targets, produce different types of models and meet certain time-constraints. Some of these results can be seen in Figures 10.9 and 10.10. Each model came from a different run with different parameters, and were hand-picked as with all other models in this section. They serve as examples of the variety of models that can be produced, and the modifications that can be made possible with the use of external modeling software. The model in Figure 10.10, while not necessarily being a target goal expected at the end of its run, was chosen due to its potential resemblance to an ice sculpture when textured. It is a perfect example of what the system can do: it can generate these models using the constraints set by the user, but it never strictly dictates which models will be considered interesting in the end. The use of the aesthetic measures are merely the system's suggestions for what may be an aesthetically pleasing model.

The ability of the system to produce a diverse population of forms, while also converging onto a certain type of model which satisfies fitness targets, is exceptionally useful. Although much of the final population of any run will be discarded, the same can be said with many potential design ideas in any field that requires them. This system provides a means of automatic production of many varied designs. This, paired with the user's ability to control and alter any aspect of the evolution and drawing controls, has resulted in a powerful design tool. Further improvements to the system might enable it to generate more specific forms, such as tables, architectural structures or plants. The potential for the system to generate any conceivable form is astounding, but of course limited by the user's understanding of the controls and imagination.

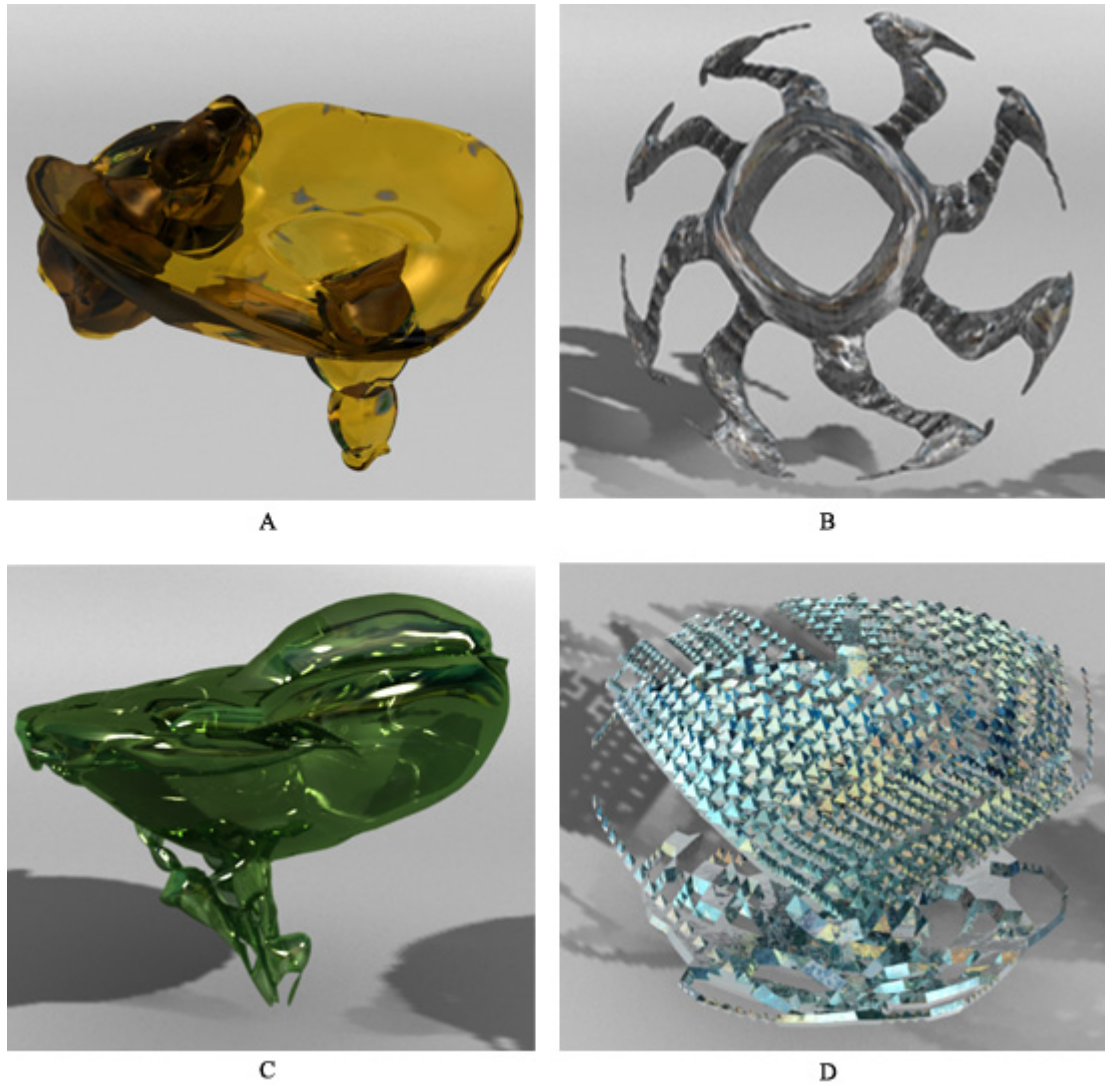


Figure 10.9: Renderings of models using smoothing and texturing, each evolved using separate parameters

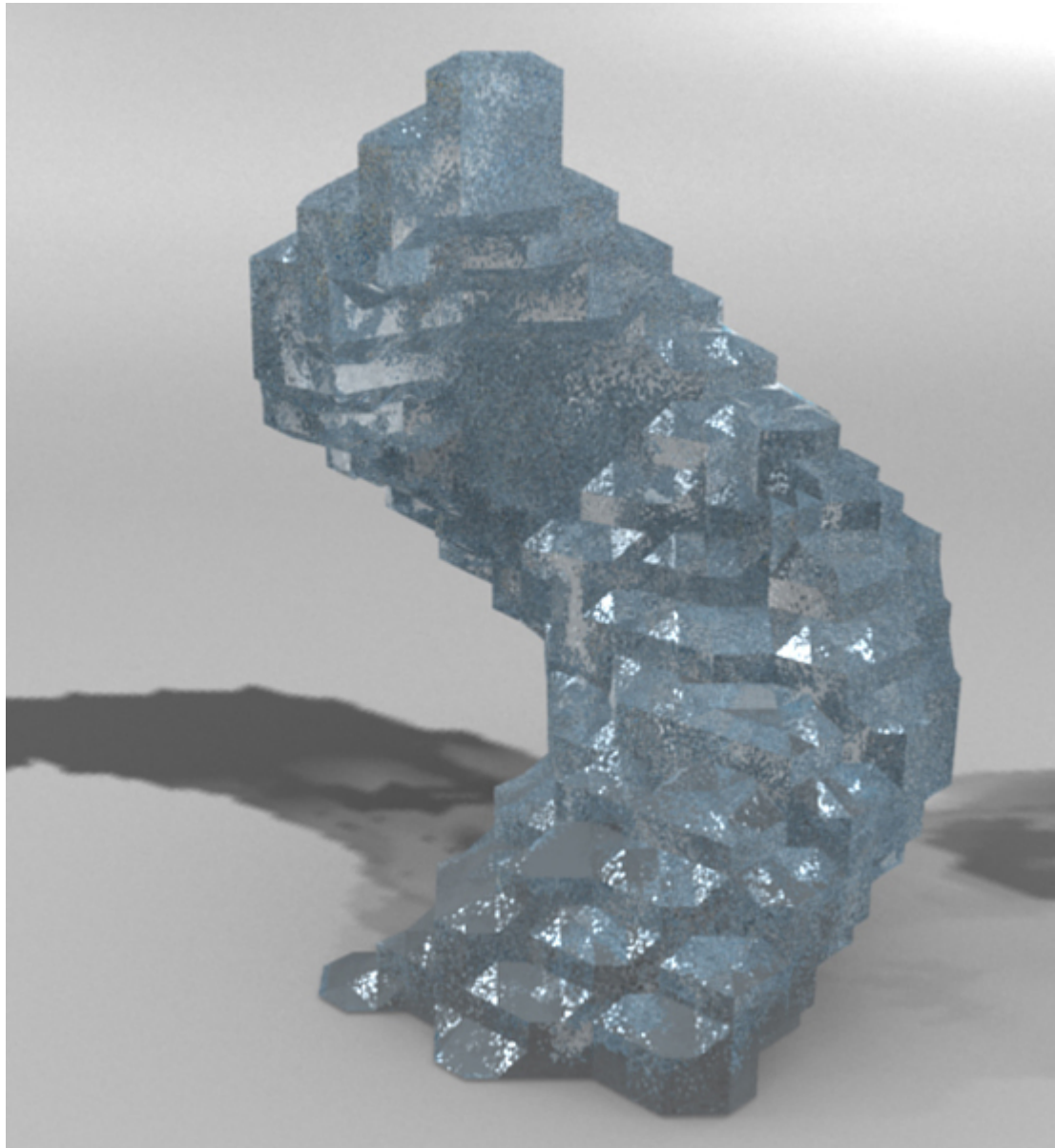


Figure 10.10: A high-quality rendering of an ice-like sculpture

## Chapter 11

# Human-centered Study of Aesthetic Measures

### 11.1 Introduction

Due to the subjective nature of the aesthetic measures presented in this thesis, it is difficult to make any definitive conclusions on their effectiveness or usefulness. While it can be seen from previous experiments that certain fitness functions can be used to produce more complex models, symmetrical models and so on, the actual aesthetic value of the models produced is a subject of debate. These fitness functions have been shown in the past to relate to the theories and concepts reflecting aesthetic value, but their actual applicability to EC and the production of automated artwork is subjective. One of the primary issues with research involving quantifiable aesthetic measures is that human opinion is rarely taken into consideration, at least in those cases where user-guided EC is not used. Surely the introduction of a human element could only strengthen the argument that a certain aesthetic measure directly reflects the visual aesthetics of a model.

Walsh *et al.* incorporated a survey of the opinions of human participants to justify their automatically-generated aesthetic terrains, using Kolmogorov complexity as a fitness measure [63]. Using ten tests, where each test consisted of a pair of terrains – one low- and one high-scoring – they asked their participants to choose the more aesthetic between the two. Their results were subjected to the non-parametric sign test, which tests the null hypothesis that there is no difference in the medians of the continuous distributions of two random variables  $X$  and  $Y$  [62]. The test makes no assumptions about the nature of the distributions under the test, which makes it useful and commonly used in market surveys measuring consumer preferences.

For this experiment, a similar test was conducted. Twenty model pairs were chosen at random, which were all automatically generated using EC and the DFN measuring surface normals as a

target. In addition, many of the models were generated using a multi-objective strategy, pairing the DFN with other fitness functions. The fitness functions that were chosen do not detract from the success of the DFN from reaching its target, which is why they were selected. A multi-objective approach was chosen in addition to single-objective in this experiment due to the results seen in previous experiments reflecting the aesthetic appeal of models generated using multi-objective EC. The method for choosing the best models was simple for the single-objective examples, which involved picking the highest-ranked individual from the population, which were evolved using the parameter set found in Table 6.1. For the multi-objective runs, a more involved approach was taken. First, the population was sorted according to rank and the top ten individuals were set aside. For tests using pareto ranking instead of summed-rank, all of the rank one individuals were set aside instead. At this point, the individual with the best DFN score was chosen from this set, regardless of the other fitness scores, if any.

For the lower-scoring models, the worst possible DFN score – between 8 and 10 – was not automatically chosen to oppose the better score for the survey. This is because models with the lowest DFN are typically cubes or rectangular prisms, and would have likely imposed a heavy preferential bias towards the more complex models. Instead, a random individual was chosen amongst the middle-ranked individuals in the population at generation zero, which were not subjected to any evolutionary pressure. If the randomly-chosen individual's DFN exceeded 8.5, then a new individual was chosen. A list of the tests and their fitness targets can be found in the Appendix in Figures A.1 to A.4. For each test, the renderings of both models are shown as well as the fitnesses of the models. In most cases, the high-scoring model pareto dominates the low-scoring, except for tests 6, 8, 9, 15, 19 and 20. For these tests, the fitnesses of the low-scoring model that dominate the high-scoring one are bolded.

The survey was hosted online, and the models were rendered in a manipulatable Java applet environment, which allows the user to rotate and examine each model. Each participant was presented with the same model pairs – or tests – in the same order. The models in each pair were placed on the left or right side with an equal probability prior to the start of the experiment. The participants were given no indication as to the fitness scores of the models. The survey required all 20 tests to be completed in order for a participant's results to be included and ran for one week, open to the public. The survey was announced via email to all students in the Brock Computer Science department, who were also encouraged to spread it as well. In total, 50 individuals filled out the survey, with only 34 of these being complete, which was likely due to technical difficulties on both client and server side. A sample of one of the tests found in the survey can be seen in Figure 11.1.

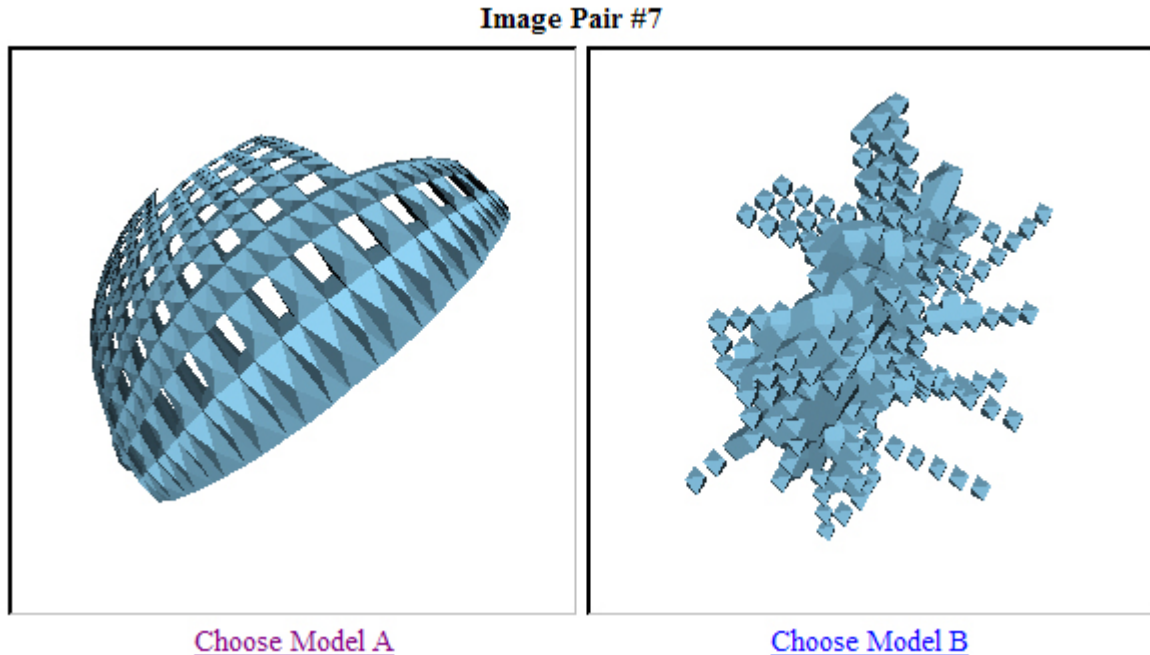


Figure 11.1: Example model pair test used in the survey

## 11.2 Results

The raw results of the survey can be found in Figure 11.2. This table shows the correct guesses of each participant in each row, for each image test. A value of 0 represents an incorrect guess, and a 1 a correct guess. The totals for each participant is shown in the column at the right, and the totals for each test for all participants is shown in the bottom row. These results of the survey were measured in a few different ways. The general results focusing on the number of correct guesses made by all participants for each image pair can be seen in the left histogram in Figure 11.2. The horizontal line represents the midpoint of 17, where most tests were hoped to exceed. From this graph alone, it is difficult to make any sound conclusions about the participants' preference for models with higher DFN scores, as seven of the tests had the number of correct guesses below the midpoint. Using the sign test, it was confirmed that the participants preferred those models that scored higher with respect to fitness, with a 95% certainty. Of all participants, from the results of the survey approximately 71% of them chose the correct, higher-scoring model over the lower-scoring ones. This analysis is a step forward in justifying a preference for the DFN.

As mentioned previously, a few of the tests contain high-scoring MO models which do not completely dominate their lower-scoring partner. The right histogram in Figure 11.2 shows the results with these tests omitted, which removes five of the seven tests which have a number of correct



Figure 11.2: (Left) Histogram displaying number of correct choices, out of 34, for all 20 tests. Horizontal line shows mid-point at 17. (Right) Same as left, except with non-dominating model pairs removed.

guesses below the midpoint, and only one test with the number of correct guesses above it. This result supports the idea that the users were influenced by more than the DFN of the models. For these tests that were removed, the lower-scoring models in 19 and 20 had better symmetry scores. In tests 6, 8 and 9 the lower-scoring models had better complexity scores. In test 15 the lower-scoring model had a better DFN. Using this new data set, of all participants, approximately 80% of them chose the higher-scoring models over the lower-scoring, which is a definite improvement over the previous data set.

Test 15 was an anomaly, in that the DFN of the selected quality model was in fact higher than the low quality model. The model was chosen at random, using the same criteria for choosing models as the others, but yielded a better-scoring model in two fitness scores, though not by a significant amount. This likely contributed to the less-than-midpoint number of successes observed in the graph. Despite the success of the experiment already, the results of this specific test could hypothetically be reversed, resulting in further improvement to the results. This is possible since the suspected lower-scoring model is actually the higher-scoring one, despite being pulled from generation zero instead of being generated from 30 generations of the GP.

### 11.3 Conclusions and Discussion

The results of this experiment were encouraging, showing a user preference for higher-scoring models with respect to DFN score, and aiding in the ongoing justification of the use of quantitative aesthetic

measures. Although it can be argued that humans are seeing “more than just the target fitness function” in these models, the statistical tests have shown that there is a preference to those models with higher DFN scores. Of course, this experiment does not completely justify or support the use of the DFN or any other aesthetic fitness function, but instead encourages further research into the topic of computational aesthetics, and offers supportive evidence of their usefulness.

A possible explanation for the results of this experiment is that certain *sweet spots* exist for these fitness functions, which define fitness areas which humans prefer. For example, perhaps a DFN of zero is not ideal, but instead a DFN range between 0 and 2. Another possible explanation is that certain aesthetic properties implicitly take preference over others when a human judges a model’s aesthetic value. Symmetry, for example, is common in most of the models used for this experiment and likely influenced many of the participants’ choices. This phenomena can also be seen when looking at the models presented in Chapter 5, as many aesthetic models have poor DFN scores, but score higher in other fitness measures. These ideas further support the notion that the area of computational aesthetics is still in its early stages, and requires much more research in order to move forwards.



## Chapter 12

# Conclusions and Future Work

### 12.1 Conclusions

This research investigated the automated evolution of aesthetic 3D forms using GP. The models were evolved using fitness functions measuring geometric properties as physical constraints to size and shape, while using aesthetic-based fitness functions to measure their aesthetic value. GP was able to consistently produce models which satisfied the targets of both sets of fitness functions, in both a single and multi-objective scenario. The multi-objective strategy is effective in producing more interesting models of varying complexity when using a combination of model constraint and aesthetic-based fitness functions. The fractal-based drawing system, using L-systems at its core, is useful as a drawing mechanism in voxel-space. Despite the limitations imposed on the system – such as drawing bounds, string limits, computational time and speed – the L-system alphabets and parsing algorithm were able to produce a wide variety of detailed models within this limited drawing space.

The new L-system encoding introduced in this thesis is capable of enforcing the production of valid and complete L-systems within a GP chromosome, including valid production rules, starting strings and the necessary relationships between the two. While this thesis is not a critique on existing encodings, it does offer enlightenment in the possibilities for encodings which do not need to take chromosome-repair into account during evolution. Such an encoding is not problem-specific, and can be taken from this thesis and used for many problems involving L-system evolution using GP. In addition, the large number of constraint adjustments available to this encoding enable it to produce any conceivable L-system, given the L-system alphabet and required number of iterations.

During this research, extensions to existing aesthetic measures were also explored. From the experiments shown in previous sections, many of these extensions were successful in properly ranking models which exhibited properties distributions similar to their 2D counterparts. In essence,

although each new aesthetic measure and their 2D equivalent did not necessarily measure the same criteria, they did in fact rely on the same means of measuring distribution data. In this sense, the extensions were a success, as they have shown to be applicable to measuring aspects of the aesthetic value of 3D models as well. Other functions, such as complexity and symmetry, were also successfully applied to 3D models. It has been shown that the use of the complexity function as a fitness function has resulted in the production of more complex models, just as symmetry has produced more symmetrical models. The relationships between some of these fitness functions, primarily entropy and DFN, were investigated as well, and were shown to exhibit similar properties which helped them to work well in conjunction. The results of the user survey helped to justify the use of these measures, as it showed a general preference for those models with a lower DFN.

The system itself excels in its intent as a tool for designers. The overall simplicity of the tool and its automation remove the requirement for more advanced users, and the parameter control and diversity of the final population allow users to produce a wide variety of similar solutions. All results can be exported as raw model data, and with little work can be further converted for import in any commercial system. This allows future users to manipulate the final results to their liking. As this tool is meant as a design aid for designers, its many features, controls and user-friendliness may pave the way for future systems, aiming to make the world of design a simpler and more practical one.

## 12.2 Future Work

This work and research merely scratches the surface of the possibilities for evolutionary design using aesthetic value as a fitness measure. Only a small handful of existing 2D aesthetic measures were explored, and research into the applicability of others to 3D model measurement would be of great interest in this problem domain. The measures that were explored in this thesis such as entropy, DFN and  $1/f$ , have the capacity for further research as well.  $1/f$  noise, for example, has a large following in the academic community, although its application to 3D model measurement has not been explored until now. A brief study was done in this thesis to find sample target ranges for these fitness functions, using existing models as samples. Although general target areas were found, the results were not significant enough to make any sound conclusions. Future research is possible in finding more localized target areas for these fitness functions by using a larger data set and simpler models.

The user survey showed a general preference for models with a low DFN, but did not take into consideration other factors, such as symmetry, complexity, and so on. Future work could be done in extending this survey to the other fitness functions, using them as a primary target for the evolutionary process in place of the DFN. The results could also be used to locate the sweet spots for each fitness function by using a larger number of participants.

The extensions of DFN, entropy and  $1/f$  to measure 3D distribution data were relatively easy,

but the conversion of other fitness functions were not so simple. The measurement of symmetry in a 3D environment is generally difficult, and the symmetry function used in this thesis is only an approximation. There is a definite need for future study in 3D symmetry analysis, as the measure used here ignores several aspects of symmetry which are difficult to measure in 3D. Many complexity measurements exist, and generally apply to images, fractals and so on. The complexity measure presented in this thesis is only an estimate of the fractal dimension of a 3D model, and higher precision results could arise using a 3D box-counting method. Such a method was ignored for this thesis due to time constraints and the time required to compute such information. Research into additional measurements is also possible, such as those measuring color and lighting, or various model topology measurements, such as higher-order surface derivatives and textures. More volumetric measurements could also be implemented, such as 2D and 3D space-filling functions which match the model's volume to another pre-computed area or volume.

The models produced by the system are particular to the implementation of the modeling system which created them. Other 3D modeling implementations would likely produce differently-styled results, especially if the use of voxels and the Marching Cubes algorithm were removed. The voxel-space drawing, with the aid of the Marching Cubes algorithm, allowed for the construction of model approximations that could have potentially been of much greater detail. This approach was taken here due to the overall processing complexity of a such models. This fact also resulted in the use of a bounding box for the voxel-space, limiting the overall canvas size. This limitation resulted in the production of many models which were cut off as the drawing pen travelled outside the boundaries. In the end, many forms which would have appeared more fractal-like were cropped by the bounding box. Greater computational resources would allow a larger bounding box, or even suggest the removal of it completely. These resources would also cut down on the runtime of a single GP run in the system, which could range from a few hours to a few days. New drawing grammars and modeling languages are of definite interest, as they will allow the creation of more detailed and specific models.

The L-system encoding could also be further improved to allow for the evolution of specific parts of the L-system. For example, a user could supply the starting string and a few production rules, and the GP would evolve the rest. This would allow for the production of more specific models, or even result in the generation of variations of a user-supplied L-system model. The limitations imposed on the size of each L-system string also resulted in the cropping of longer strings, which may have potentially produced more complex and interesting models. This decision was based entirely on necessary processing limitations, which may not be so daunting in the future. Other possibilities for the encoding used in this thesis are evident, such as the use of parametric L-systems. This was omitted for several reasons, but primarily due to the increase in processing time of complex evaluation strings and the ability for a non-parametric L-system to mimic the behavior of a parametric one. For example,  $F(3)$  is equivalent to  $FFF$ , assuming that  $F$  moves the pen forward one unit.

# Bibliography

- [1] A3D. Archive 3d. In *http://archive3d.net/*.
- [2] Christian Beck. Generalized information and entropy measures in physics, 2009.
- [3] P. Bentley and D. Corne. *Creative evolutionary systems*. Evolutionary Computation Series. Morgan Kaufmann, 2002.
- [4] Peter J. Bentley and Jonathan P. Wakefield. Finding acceptable solutions in the pareto-optimal range using multiobjective genetic algorithms. In P. K. Chawdhry, R. Roy, and R. K. Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, pages 231–240. Springer-Verlag, January 1998.
- [5] S. R. Bergen. Evolving stylized images using a user-interactive genetic algorithm. In Rothlauf [50], pages 2745–2752.
- [6] S. R. Bergen and B.J. Ross. Evolutionary art using summed multi-objective ranks. In O’Reilly U.-M. Riolo R. and McConaghy T., editors, *Genetic Programming Theory and Practice VIII*, pages 227–244. Springer, 2010.
- [7] G. D. Birkhoff. *Aesthetic Measure*. Harvard University Press, 1933.
- [8] C. Coia. *A Genetic Approach to the Grammatical Creation of External Building Architecture*. MSc Thesis, Department of Computer Science, Brock University, 2011.
- [9] Helen Couclelis. Cities and complexity: Understanding cities with cellular automata, agent-based models, and fractals - michael batty. *Papers in Regional Science*, 85(3):471–473, 2006.
- [10] R. Flack. *Evolution of Architectural Floor Plans*. MSc Thesis, Department of Computer Science, Brock University, 2011.
- [11] Jaime Garces-perez, Dale A. Schoenefeld, and Roger L. Wainwright. Solving facility layout problems using genetic programming. In *Stanford University*, pages 28–31. MIT Press, 1996.

- [12] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1 edition, January 1989.
- [13] G. Gunlu and H.S. Bilge. Symmetry analysis for 2d images by using dct coefficients. *ICSCCW*, pages 1–4, 2009.
- [14] H. Jurgens H. Peitgen and D. Saupe. *Chaos and fractals, new frontiers of science*. 1993.
- [15] E. Den Heijer and A. E. Eiben. Comparing aesthetic measures for evolutionary art, 2010.
- [16] Martin Hemberg and Una-May O’Reilly. GENR8 - using grammatical evolution in A surface design tool. In Alwyn M. Barry, editor, *GECCO 2002: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference*, pages 120–123, New York, 8 July 2002. AAAI.
- [17] Brian Henderson-Sellers and David Cooper. Has classical music a fractal nature? - a reanalysis. *Computers and the Humanities*, 27(4):277–284, 1993.
- [18] G. Hornby. Evolving L-systems to generate virtual creatures. *Computers & Graphics*, 25(6):1041–1048, December 2001.
- [19] Gregory S. Hornby. Measuring complexity by measuring structure and organization. In *2007 IEEE Congress on Evolutionary Computation*, pages 2017–2024. IEEE Press, 2007.
- [20] K. J. Hsu and A. J. Hsu. Fractal Geometry of Music. *Proceedings of the National Academy of Science*, 87:938–941, February 1990.
- [21] Christian Jacob. *Illustrating evolutionary computation with Mathematica*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.
- [22] Christian Jacob, Aristid Lindenmayer, and Grzegorz Rozenberg. Genetic l-system programming. In *Parallel Problem Solving from Nature III, Lecture Notes in Computer Science*, pages 334–343. Springer-Verlag, 1994.
- [23] Yannick Joye. Fractal architecture could be good for you. *Nexus Network Journal*, 9:311–320, 2007.
- [24] Michael Kazhdan, Bernard Chazelle, David Dobkin, Thomas Funkhouser, and Szymon Rusinkiewicz. A reflective symmetry descriptor for 3d models, 2004.
- [25] Rafal Kicinger, Tomasz Arciszewski, and Kenneth De Jong. Evolutionary computation and structural design: A survey of the state-of-the-art. *Comput. Struct.*, 83:1943–1978, September 2005.

- [26] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
- [27] Matthew Lewis. Evolutionary visual art and design. In Juan Romero and Penousal Machado, editors, *The Art of Artificial Evolution*, Natural Computing Series, pages 3–37. Springer Berlin Heidelberg, 2008.
- [28] Ming Li and Paul Vitanyi. An introduction to kolmogorov complexity and its applications: Preface to the first edition, 1997.
- [29] Markus Lipp, Peter Wonka, and Michael Wimmer. Interactive visual editing of grammars for procedural architecture. In *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, pages 102:1–102:10, New York, NY, USA, 2008. ACM.
- [30] H. Lipson and W. Cochran. *The determination of crystal structures - 3rd revised and enlarged ed.* Cornell University Press, 1966.
- [31] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21:163–169, August 1987.
- [32] S. Luke. ECJ - a java-based evolutionary computation research system. In <http://cs.gmu.edu/eclab/projects/ecj/>.
- [33] Penousal Machado and Amlcar Cardoso. Computing aesthetics. In *Proceedings of the Brazilian Symposium on Artificial Intelligence, SBIA98*, pages 219–229. Springer-Verlag, 1998.
- [34] Edoardo Milotti. 1/f noise: a pedagogical review. *arxiv preprint, physics/0204033*, April 2002.
- [35] C. Neufeld, B. J. Ross, and W. Ralph. The evolution of artistic filters. In Juan Romero and Penousal Machado, editors, *The Art of Artificial Evolution*, Natural Computing Series, pages 335–356. Springer Berlin Heidelberg, 2008.
- [36] R.A. Novelline and L.F. Squire. *Squire's fundamentals of radiology*. Squire's Fundamentals of Radiology. Harvard University Press, 2004.
- [37] Michael O'Neill, John Mark Swafford, James McDermott, Jonathan Byrne, Anthony Brabazon, Elizabeth Shotton, Ciaran McNally, and Martin Hemberg. Shape grammars and grammatical evolution for evolutionary design. In *GECCO*, pages 1035–1042, 2009.
- [38] M. Ostwald. Fractal architecture: Late twentieth-century connections between architecture and fractal geometry. *Nexus Network Journal*, 3(1):73–83, 2001.
- [39] Mine Özkar and George Stiny. Shape grammars. In *ACM SIGGRAPH 2009 Courses*, SIGGRAPH '09, pages 22:1–22:176, New York, NY, USA, 2009. ACM.

- [40] Wenjun Pang and K. Hui. Interactive evolutionary 3d fractal modeling. *The Visual Computer*, 26:1467–1483, 2010. 10.1007/s00371-010-0500-8.
- [41] Yoav I. H. Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 301–308, New York, NY, USA, 2001. ACM.
- [42] Riccardo Poli and William B. Langdon. Genetic programming theory I & II. In *Genetic and Evolutionary Computation Conference*, pages 3015–3056, 2009.
- [43] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A field guide to genetic programming*. Published via <http://lulu.com> and freely available at <http://www.gp-field-guide.org.uk>, 2008. (With contributions by J. R. Koza).
- [44] P. Prusinkiewicz and J. Hanan. Lindenmayer Systems, Fractals, and Plants. *Lecture Notes in Biomathematics*, 75, 1989.
- [45] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc., New York, NY, USA, 1996.
- [46] W. Ralph. *Painting the Bell Curve: The Occurrence of the Normal Distribution in Fine Art*. In preparation, 2006.
- [47] Jaume Rigau, Miquel Feixas, and Mateu Sbert. Conceptualizing birkhoff's aesthetic measure using shannon entropy and kolmogorov complexity. pages 105–112, 2007.
- [48] M. A. Rosenman. The generation of form using an evolutionary approach. *Artificial*, pages 643–662, 1996.
- [49] Brian J. Ross, William Ralph, and Hai Zong. Evolutionary image synthesis using a model of aesthetics. In Gary G. Yen, Lipo Wang, Piero Bonissone, and Simon M. Lucas, editors, *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 3832–3839, Vancouver, 6-21 July 2006. IEEE Press.
- [50] Franz Rothlauf, editor. *Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009, Companion Material*. ACM, 2009.
- [51] N. Sala. Fractal models in architecture: A case of study. In *Proceedings International Conference on Mathematics for Living*, pages 266–272, 2000.
- [52] R. Scruton. *The aesthetics of architecture*. University paperbacks. Methuen, 1979.
- [53] K. Sims. Artificial evolution for computer graphics. *Computer Graphics*, pages 319–328, 1991.

- [54] Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 15–22, New York, NY, USA, 1994. ACM.
- [55] B. Spehar, C. W. G. Clifford, B. R. Newell, and R. P. Taylor. Universal aesthetic of fractals. *Computers and Graphics*, 27:813–820, 2003.
- [56] P. Steadman. *The Evolution of Designs: Biological Analogy in Architecture and the Applied Arts - revised edition*. Routledge, London and New York, 2008.
- [57] Kostas Terzidis. *Algorithmic Architecture*, volume 1. Architectural Press, 2006.
- [58] S. Todd and W. Latham. *Evolutionary art and computers*. Academic Press, 1992.
- [59] M.F. Triola. *Essentials of Statistics: International Edition*. Pearson Education, Limited, 2010.
- [60] Peter von Buelow. *Genetically Engineered Architecture - Design Exploration with Evolutionary Computation*. VDM Verlag, Saarbrucken, Germany, 2007.
- [61] Richard F. Voss and John Clarke.  $\frac{1}{f}$  noise from systems in thermal equilibrium. *Phys. Rev. Lett.*, 36:42–45, Jan 1976.
- [62] Dennis D. Wackerly, William Mendenhall III, and Richard L. Scheaffer. *Mathematical Statistics with Applications*. Duxbury Advanced Series, sixth edition edition, 2002.
- [63] Paul Walsh and Prasad Gade. The use of an aesthetic measure for the evolution of fractal landscapes. In *IEEE Congress on Evolutionary Computation*, pages 1613–1619. IEEE, 2011.
- [64] M. Sei Watanabe. *Induction Design: A Method for Evolutionary Design*. Birkhauser, Basel, Switzerland, 2002.



## Appendix A

# Miscellaneous Analysis and Results

Table A.1: Two-tailed T-test statistics comparing the first generation results of both encodings, over 10 runs. One test is applied for unused production rules and one for complexity.

<i>Stats</i>	<b>Unused Rules</b>		<b>Complexity</b>	
	<i>Encoding A</i>	<i>Encoding B</i>	<i>Encoding A</i>	<i>Encoding B</i>
Mean	0.83774877	0.066604914	15.51863704	828.3132519
Variance	0.000523531	6.12014E-05	1892.205561	13064.76526
Observations	10	10	10	10
Hypothesized Mean Difference	0		0	
t Stat	95.6705		-19.9379	
P(T≤t) two-tail	3.81E-16		2.21E-09	
t Critical two-tail	2.2281		2.2281	

Table A.2: Two-tailed T-test statistics comparing the first generation results of both encodings, over 10 runs. One test is applied for the percentage of invalid individuals and one for unchanged starting strings.

<i>Stats</i>	<b>Invalid Individuals</b>		<b>Unchanged Strings</b>	
	<i>Encoding A</i>	<i>Encoding B</i>	<i>Encoding A</i>	<i>Encoding B</i>
Mean	0.441555556	0.101555556	0.770444444	0.027555556
Variance	0.000235778	0.000133778	0.000966778	5.47778E-05
Observations	10	10	10	10
Hypothesized Mean Difference	0		0	
t Stat	53.0591		69.7291	
P(T≤t) two-tail	1.74E-18		1.30E-13	
t Critical two-tail	2.1314		2.2621	

Table A.3: Two-tailed T-test statistics comparing the average fitnesses for Pareto and Summed Rank from the final generation, for each fitness function.

<i>Stats</i>	<b>DFN</b>		<b>Complexity</b>		<b>Symmetry</b>	
	<i>DFN</i>	<i>Pareto</i>	<i>DFN</i>	<i>Pareto</i>	<i>DFN</i>	<i>Pareto</i>
Mean	1.760388	6.405594	6.520341	4.338154	0.954483	0.854715
Variance	0.087623	0.015399	5.631522	0.275057	0.000482	0.000709
Observations	10	10	10	10	10	10
Hyp. Mean Difference	0		0		0	
t Stat	-45.7656		2.839383		9.139721	
P(T≤t) two-tail	7.73E-15		0.017567		5.69E-08	
t Critical two-tail	2.178813		2.228139		2.109816	

Table A.4: Raw results from survey participants. A value of 0 is an incorrect guess, and a value of 1 is a correct guess. The totals for each model test is shown in the bottom row, and the totals for each participant is shown in the last column

Tests 1-5					Tests 6-10					Tests 11-15					Tests 16-20					T
1	1	0	0	0	0	1	1	0	0	1	0	0	1	1	0	0	1	0	1	9
1	0	1	0	1	1	1	0	0	0	1	0	0	1	0	0	0	0	0	1	8
0	1	1	1	0	0	1	0	0	0	0	1	0	1	0	1	1	1	0	1	10
0	1	1	1	0	0	1	1	1	1	1	1	1	0	0	1	0	0	1	0	12
1	1	1	1	1	1	1	0	0	0	1	1	0	1	1	0	0	1	0	0	12
1	1	0	1	1	1	0	0	1	1	1	1	1	0	0	1	0	0	0	1	12
1	0	0	1	0	1	1	0	1	1	1	1	1	1	0	1	1	0	1	0	13
0	0	1	1	0	1	1	0	0	1	1	1	1	0	1	0	1	1	0	0	11
1	0	0	0	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1	11
1	1	1	0	1	0	1	0	0	1	1	1	1	0	0	1	1	1	0	0	12
1	0	0	1	0	0	1	0	0	1	1	1	1	1	0	1	1	0	1	0	11
1	1	1	1	1	1	1	1	0	0	0	1	0	1	0	1	0	0	0	0	12
0	0	1	0	1	1	1	0	0	0	1	0	1	1	1	1	0	1	0	1	11
1	1	0	1	0	0	0	0	1	1	1	1	1	0	1	1	1	0	1	0	12
1	0	0	0	0	1	1	1	0	0	0	0	1	1	1	1	1	0	0	1	10
1	0	1	0	0	0	1	1	0	0	1	0	1	1	1	1	1	0	0	1	11
1	1	0	0	1	1	1	0	1	1	1	1	0	0	0	1	1	1	1	0	13
1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	0	1	0	16
1	1	1	0	1	1	1	1	0	0	1	1	1	1	1	1	0	0	0	1	14
1	0	1	1	1	0	1	0	0	1	1	1	0	1	0	1	1	1	0	1	13
1	0	1	0	1	1	1	1	0	0	0	0	0	1	1	1	1	0	1	1	12
1	1	1	1	1	0	1	1	0	0	1	0	1	1	1	1	0	1	1	1	14
0	0	0	1	1	1	1	0	0	0	1	1	0	0	0	0	0	1	0	0	7
0	1	1	1	0	0	1	1	1	1	1	1	1	0	0	1	0	0	1	0	12
1	1	0	0	1	1	1	0	0	0	1	0	0	1	0	0	1	0	0	0	9
1	0	1	0	0	0	1	0	0	0	1	1	0	1	0	1	0	0	1	0	9
1	1	1	1	0	0	1	0	0	0	1	1	1	1	0	0	0	1	0	0	10
1	1	0	1	0	1	1	1	1	0	1	1	1	0	0	0	0	0	0	0	11
0	1	1	1	1	1	1	1	0	1	1	1	1	1	0	0	1	0	1	1	15
1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	0	0	1	14
1	1	1	1	1	1	1	0	1	0	1	0	1	1	1	1	0	0	0	0	13
1	1	1	1	0	1	1	1	0	0	1	0	1	1	1	1	0	1	0	1	14
1	0	1	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	8
1	1	0	0	1	1	1	1	0	0	0	1	0	1	1	1	0	0	0	0	10
27	21	22	20	19	23	32	15	8	12	27	22	22	25	16	17	25	14	9	15	391

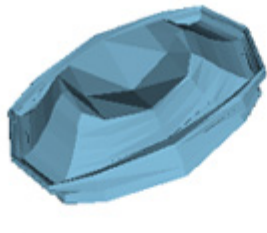
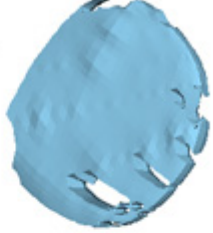

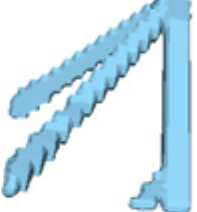

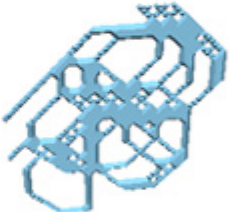

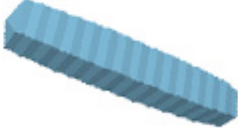
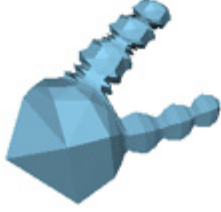
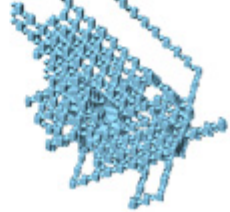
Fitness Target(s)	High Fitness Model		Low Fitness Model	
	Model	Fitness	Model	Fitness
DFN (0.0)		0.5875		5.137
DFN (0.0)		0.3523		6.2187
DFN (0.0)		0.3804		7.469
DFN (0.0)		0.5289		8.0899
DFN (0.0) Symmetry (1.0) Complexity (MAX)		1.3989 12.79 1		6.1111 8.4437 0.6574

Figure A.1: Fitnesses, targets and models for tests 1-5 used in human-oriented survey

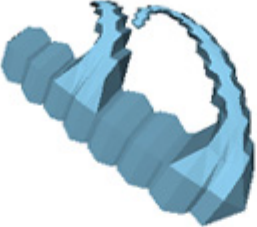

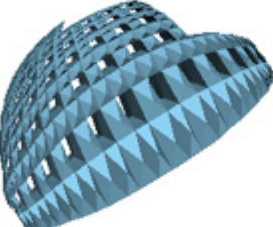
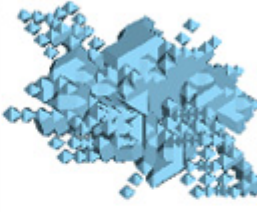






Fitness Target(s)	High Fitness Model		Low Fitness Model	
	Model	Fitness	Model	Fitness
DFN (0.0) Symmetry (1.0) Complexity (MAX)		1.4851 8.9557 1		8.1355 <b>10.2252</b> 0.8803
DFN (0.0) Symmetry (1.0) Complexity (MAX)		2.3195 15.0746 1		6.7633 6.9857 0.9022
DFN (0.0) Symmetry (1.0) Complexity (MAX) (PARETO)		1.0402 6.0633 0.875		3.1698 <b>8.4314</b> 0.8044
DFN (0.0) Symmetry (1.0) Complexity (MAX) (PARETO)		1.348 9.389 1		3.6214 <b>18.0791</b> 0.7333
DFN (0.0) Entropy (MAX)		1.2588 4.1293		4.5235 2.8956

Figure A.2: Fitnesses, targets and models for tests 6-10 used in human-oriented survey

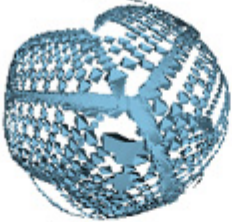

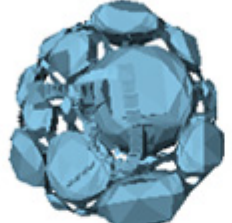







Fitness Target(s)	High Fitness Model		Low Fitness Model	
	Model	Fitness	Model	Fitness
DFN (0.0) Entropy (MAX)		1.7761 4.1396		4.5068 2.9513
DFN (0.0) Complexity (MAX) Mean (0.005) Standard Dev. (1.5)		1.2531 7.3761 0.0051 1.5001		2.839 1.7774 -0.0022 0.6529
DFN (0.0) Surface Area (50K)		1.2968 4020		7.6689 11479
DFN (0.0) Dim. (300, 100, 80) Surface Area (50K)		0.8266 422.5116 1569		4.7279 425.6088 484
DFN (0.0) Mean (0.025) S. Dev. (0.25)		3.1166 1.30E-04 0.3121		<b>2.7026</b> <b>6.95E-04</b> 0.8376

Figure A.3: Fitnesses, targets and models for tests 11-15 used in human-oriented survey











Fitness Target(s)	High Fitness Model		Low Fitness Model	
	Model	Fitness	Model	Fitness
DFN (0.0) Symmetry (1.0)		1.117 1		2.5071 0.9517
DFN (0.0) Complexity (MAX) Symmetry (1.0)		1.0932 12.6966 1		5.9297 -0.9987 1
DFN (0.0) Complexity (MAX) Symmetry (1.0) (SPHERES)		4.92 13.3874 1		6.8493 6.7189 0.9912
DFN (0.0) Complexity (MAX) Symmetry (1.0)		0.5796 5.0199 0.7032		4.8828 2.0921 1
DFN (0.0) Complexity (MAX) Symmetry (1.0)		1.3176 5.1577 0.8078		3.5981 2.1088 1

Figure A.4: Fitnesses, targets and models for tests 16-20 used in human-oriented survey