Title: **Evolutionary algorithms in modeling and animation**

Authors: Anargyros Sarafopoulos [1] and Bernard F. Buxton [2]

[1] Media School, Bournemouth University, asarafop@bournemouth.ac.uk

[2] Computer Science, University College London, B.Buxton@cs.ucl.ac.uk

Contents:

1 Introduction

Generating computer animation involves two interwoven components. The first component is *the set of tools* (*tools*) used to generate and render computer animation. Tools consist of software and hardware that allow the creation of abstract geometric models, modification of these models over time, as well as their rendering. The second component is *the sequence of instructions to be carried out by the software and hardware tools*, to generate and render a specific animation sequence, here referred to as *execution plan* or *execution*. Execution usually resides in the thoughts, story-boards, and drawings of director and animators that have to carry out the task of creating a specific sequence. Most modern software and hardware focus on the creation of *tools* that allow for the generation and rendering of realistic looking models and motion. Modeling and rendering tools are very important as they provide the materials used by the computer animator. Novel tools/materials often create new avenues for visual exploration. However, the quality of an animation sequence depends equally on the quality of the work carried out by the director and animators in terms of using the tools available, i.e. that quality of *execution*. Sometimes the vision of the director and animator exhausts the capacity of the tools at hand so new tools and materials need to be employed or invented to assist execution. Execution as noted above, formed in the mind of the director and animators, often results to a chain of sampling of data (scanning of images or film), button presses, menu selections, and/or evaluation of scripts. However, an execution plan is not conceived in an instant or in a moment of inspiration. The director/animator will usually test and research a large number of forms and motions until they decide on how the final work will be made. The making of an execution plan often proceeds by trial and error, searching through many options open to the animator in terms of creating a sequence. Animators often compose using a process of progression through trial and error. Models and animation are initially created using a given set of tools. The newly created temporal and spatial forms are typically tested against goals set at the beginning of the development process. The models or animation are then modified in order to fulfill preset requirements more closely and the process is repeated until the development time has been exhausted or the goals have been reached. Often there is a recurrent production cycle, where the fitness of execution is tested against the given set of goals. This results in a gradual improvement of execution plans through a "survival of the fittest" approach to creative design.

A parallel can thus be drawn between organic evolution and the human creative design process. Based on this analogy, new software is starting to emerge that focuses on assisting creative design in engineering [Furuta, Maeda, Watanabe, 1995] [Koza, Bennett, Andre, Keane, 1999] and in the arts [Todd, Latham, 1992] [Lund, Pagliarini, Miglino, 1995] [Bentley 1999] [Bentley, Corne,

2001]. This software is developed as a tool to assist with the invention of new algorithms, or the exploration of complex spaces generated by mathematical or procedural tools and can therefore be seen as a *meta-tool*. Such Evolutionary Algorithms (EAs) applied to computer animation are often used as a *meta-tool* made to assist in the second important component of computer animation mentioned above, the generation of *execution plans.* Evolutionary algorithms are an interdisciplinary research field that connects computer science, artificial intelligence, and biology. EAs are based on simulation of natural evolution on the computer and form a computational search technique. "They combine survival of the fittest with structured yet randomized information exchange to form a search algorithm with some of the innovative flair of human search" [Goldberg, 1989]. Steven Dawkins describes his seminal work on generating two dimensional (2D) branching forms (bioforms) using interactive evolution on the computer [Dawkins, 1986-7]. Dawkins' idea was extended and applied to the field of computer graphics by Todd, Latham [Todd, Latham, 1991-2] and Sims [Sims, 1991-3, 1997]. Latham and Sims use evolutionary algorithms for the generation of procedural models and animation. Reynolds [Reynolds, 1992, 1994a-e] uses EAs for the evolution of the behavior of artificial agents in 2D environments, whilst other researchers present work for the evolution of behavioral animation [Zhao, Wang, 1998] [Griffiths, Sarafopoulos, 1999] [Lim, Thalmann, 1999]. Gruau, Gritz, and Kang describe the evolutionary design of controllers for articulated structures [Gruau, 1996a] [Gritz, Hahn, 1995, 1997] [Gritz, 1999] [Kang, Cho, Lee, 1999]. Sims describes the evolution of behavior and topology using EA [Sims, 1994a-b]. Evolution of shaders and textures is investigated by [Ibrahim, 1998], and [Wiens, Ross, 2000-1]. Using high-level commercial animation tools (like Houdini, and Maya) to evolve models and textures is described by [Lewis, 2000]. Today there are numerous [Lewis, 2001] [Bentley, 1999] interactive systems available for the evolution of models and animation.

This chapter provides some biological background and describes the main evolutionary algorithms as general optimization and search tools. We then concentrate on presenting two case studies applied to computer graphics and animation based on research carried out by one of the authors [Sarafopoulos 1995, 1999 and 2001]. The first case study involves interactive selection and the evolution of 2D procedural textures the second focuses on the evolution of iterated function systems.

2 Biological Background

2.1 Evolution

Evolutionary algorithms are based on the simulation of a model of organic evolution formulated by Charles Darwin [Darwin, 1895]. The Darwinian theory of evolution explains the emergence of complex living organisms as the result of gradual improvements through a recurrent process of *selection* and *reproduction* with *variation*. Selection is the processes of assigning *fitness* to individual organisms under a given environment. Individual organisms better equipped to survive and breed are said to have higher *fitness*. For living organisms to persist through the aeons, characteristics of parent individuals are transmitted to offspring through the process of reproduction. Reproduction is not a procedure of exact replication from parent to offspring. A controlled amount of variation is involved during the process of reproduction so offspring are not identical to the parents. Variation leads to random and undirected changes between *phenotypes*, i.e. the behaviour and embodiment of individual organisms.

Natural selection favours individual organisms that are better adapted to their environment; in environments with finite resources better-adapted individuals exploit scarce resources more effectively and therefore survive to produce offspring. Individuals that are not adequately adapted constantly die off. Over many generations of individuals, natural selection continually filters out and eventually kills off unfit organisms. Since more individuals are born than survive to breed, natural evolution progressively amplifies fitness. Individuals with variations that are favoured by the environment gain an *adaptive advantage*. Advantageous variations accumulate over time leading to highly adapted populations. A "*natural selection*" occurs resulting in populations of highly adapted living organisms.

The modern theory of evolution known as the *synthetic theory of evolution, modern synthesis*, or *neo-Darwinism* [Ridley, 1996] unifies the Mendelian and molecular theory of inheritance with the Darwinian theory of natural selection. According to the Mendelian theory of inheritance characteristics from parent to offspring are transmitted in the form of discrete particles called genes. Genes control the development of living organisms and encode their phenotype. The genetic make up of an organism, that is, all phenotypic characteristics encoded in genes, is referred to as *genotype*. It is well known today that DNA (deoxyribonucleic acid) stores the genetic information of living organisms. The *synthetic theory of evolution* describes evolution in nature, as a process of *selection* and *reproduction* with *variation*, where: selection is a process that operates on phenotypes and variation a process that operates on genotypes. That is, where selection acts on an individual's physical embodiment and behaviour, variation operates on the

molecular level of DNA strands. Variation is a stochastic process that results from random errors during reproduction (*mutations*), and by the shuffling of genetic information passed from parent to offspring during sexual reproduction (*recombination*). Selection, however, is predominantly deterministic. To survive and procreate individuals depend on specific phenotypic characteristics (traits) that render useful skills in exploiting resources available under a given environment. Hence, selection can be seen as a function of the traits of an individual organism.

2.2 Fitness Landscapes

Through the process of natural selection populations evolve over time that become better adapted to their environment. The most common way to visualise evolutionary change is by depicting individual organisms as points in a multidimensional space, where one axis corresponding to each biological trait and an additional dimension is used to depict fitness. This abstraction is usually referred to as a *fitness landscape*, and was originally coined by the biologist Sewall Wright [Wright, 1931] in the context of population genetics. In the simple three-dimensional (3D) case, individuals are represented as points on a 3D surface, having two trait co-ordinates and a fitness value. Such a 3D surface resembles a natural landscape with peaks and valleys depicting areas of high and low fitness (see figure 1). However, from the biological point of view the concept of a static fitness landscape is often not realistic. There are several reasons why this might be so. Firstly, fitness in nature can be measured only indirectly (by the propensity of individual organisms to survive and procreate), since the exact biological traits and how they affect fitness is usually not known. The fitness of phenotypes may depend on their frequency [Ridley, 1996]. Secondly, populations often modify the resources of their environment, and consequently the mapping from traits to fitness may also vary as a function of time. Such environment population interactions are a process that is inadequately understood.

Apart from natural selection, there are other factors that can change the way populations evolve [Ridley, 1996]; variation in small populations and changes of the state of the environment due to chance can cause the increase or decrease of certain traits, a phenomenon known as *genetic drift* [Provine, 1986]. Motto Kimura proposed the neutral theory of molecular evolution [Kimura, 1983] where he argues that “most evolutionary changes at the molecular level are due to the random process of genetic drift acting on mutations, rather than natural selection. While recognising the importance of selection in determining functionally significant traits, he holds that the great majority of the differences in macromolecular structures observed between individuals in a population are of no adaptive significance and have no impact on the reproductive success of the individual in which they arise. This contrasts with the orthodox neo-

Darwinian view that nearly all evolutionary changes have adaptive value for the organism and arise through natural selection" [Martin, Hine, 2000].

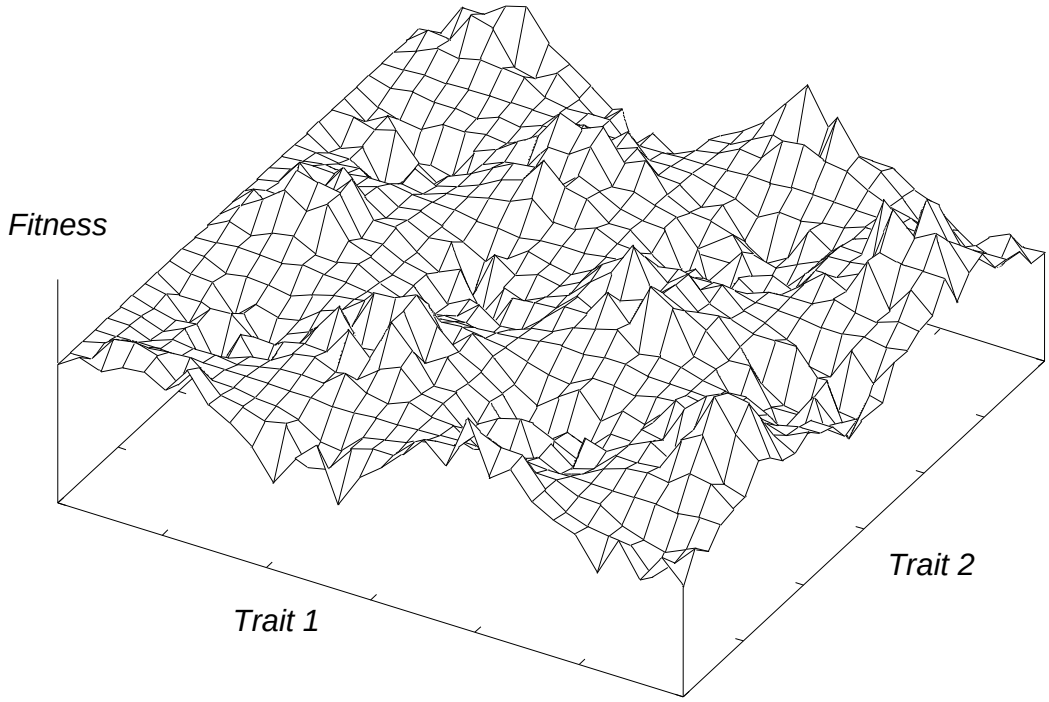

Figure 1. 3D fitness landscape schematic. Evolution guides populations along the fitness landscape in certain ways, for example, adaptation makes a population move towards local maxima, i.e. peaks in the landscape. According to Sewall Wright's *shift balance* theory, genetic drift can cause sub-populations to step across ridges from a low fitness end of the landscape to another.

3 Evolutionary Algorithms

"The problem of how to solve problems is unsolved-and mathematicians have been working on it for centuries."

Evolution and optimum seeking, Hans-Paul Schwefel

3.1 Artificial evolution in search and optimisation

The problem of finding solutions to problems is itself a problem with no general solution. In computer science and artificial intelligence finding the solution to a problem is often thought of as a search through the space of possible solutions. The set of possible solutions defines the *search space* for a given problem. Solutions or partial solutions to a problem are viewed as points in the search space. In engineering and mathematics finding the solution to a problem is often thought of as process of optimisation. Problems at hand are often formulated as mathematical models expressed in terms of functions, or systems of functions. Hence, in order to find a solution, we need to discover the parameters that optimise the model or the function components that provide optimal system performance. There are several well-established search/optimisation techniques in the literature. These are usually classified in three broad categories, enumerative, calculus based, and stochastic [Langdon, 1998].

Enumerative methods are based on the simple principle of searching through potentially all points in the search space one at a time. In the field of artificial intelligence enumerative methods subdivide into two categories, *uninformed* and *informed* methods. Uninformed or blind methods like the *minimax* algorithm (used in game playing) search all points in the space in predefined order. Informed methods like Alpha-Beta and A* perform a more sophisticated search using domain specific knowledge in the form of a cost function or *heuristic* in order to reduce the cost of the search [Russell, Norvig, 1995, chapters 3 and 4].

Calculus based techniques, are also often classified into two sub-categories: direct and indirect. Indirect or analytic methods stem form the origins of mathematical optimisation in the calculus and are based on discovering values that take function derivatives to zero. Direct or numerical methods like Newton-Raphson and Fibonacci are usually iterative techniques that navigate the fitness landscape using gradient/function information to move in the direction of the solution. Like a sightless climber that feels his way to the highest peak, they are also known as *hill climbing* strategies. For an overview of hill climbing strategies, see [Schwefel, 1995].

Stochastic methods are iterative methods that navigate through the search space using probabilistic rather than deterministic rules. A popular random method is *simulated annealing*. It was developed in explicit analogy with natural annealing, the process whereby, in order to harden

steel, low energy crystals are formed in copper by heating copper to liquid state and then gradually cooling until it freezes [Kirpatrick, Gelatt, Vecchi, 1983].

In the 1950s and 1960s several researchers introduced stochastic search algorithms based on a simulation of Darwinian theory of natural selection, in order to solve search and optimisation problems. Evolutionary Algorithms (EAs) are stochastic search techniques based on a computer simulation of the genetics of natural selection. EAs operate on a population of points in the search space. The population is able collectively to learn better solutions by a process of selection and reproduction with variation. Individuals in the population gradually accumulate advantageous variations through a selection pay-off that leads the search to, or close to, a solution. Evolutionary algorithms allow us effectively to search spaces in which other traditional calculus or enumerative-based techniques fail. This is because EAs often operate on a coding of the problem not the problem itself, they use selection pay-off, as opposed to other quality information (such as derivatives), and they also operate a parallel search by simultaneously sampling many points of the search space [Goldberg, 1986]. There are four main variations of evolutionary algorithms: genetic algorithm (GA), genetic programming (GP), evolution strategies (ES), and evolutionary programming (EP). The main difference between the above variations of evolutionary algorithms are on the encoding of an individual, and therefore in the representation or definition of the nature of the search space. A different encoding also implies a different method to stochastically modify individuals. Thus, each paradigm has a set of dedicated operations that allow for mutation and recombination of individuals in the population.

3.2 Genetic Algorithm (GA)

Holland first of conceived genetic algorithms as a theoretical framework for investigating artificial and natural evolution [Holland, 1992]. The theoretical framework specified by Holland was based on adaptation of a population of structures described as strings made out of characters of a discrete alphabet. A GA probably resembles natural evolution more closely than other evolutionary algorithms. GA genotypes are defined as strings made out of a binary alphabet, analogous to the 4-letter alphabet made out of A (adenine), C (cytosine), G (guanine), and T (thymine) nucleotide bases that make up the genetic code of living organisms [Sedivy, Joyner, 1992]. The genotypes of individuals are defined as fixed-length binary strings. In order to encode a problem, the free variables have to be represented as fixed-length binary substrings of the string that represents the genotype. The analogy between natural evolution and a genetic algorithm may be described as follows:

- A chromosome or genotype in the context of genetic algorithms thus refers to a binary string that is a coding of some aspect of a candidate solution to the problem.
- A gene is a bit or a short sequence of adjacent bits in the chromosome that encodes for a particular feature or features of the problem. (In the context of function optimisation for example, genes usually encode the parameters of the function to be optimised.)
- A locus is the position of binary digit or the position of adjacent binary digits along the chromosome.
- Alleles are all the possible configurations of binary digits in a locus i.e. the alleles of a 2-bit string segment are 00, 01, 10, and 11.
- The phenotype or candidate solution to the problem is the decoded structure. (For example, in the context of function optimisation, the decoded structure is often the function parameter set.)

Variation on binary strings as originally conceived by Holland operates in three modes (see figures 2 and 3).

- Recombination that is based on the exchange of genetic material between two parent individuals and is modelled in explicit analogy to the homologous recombination in nature.
- Inversion, modelled by analogy with inversions of DNA during replication, acts on a segment of the chromosome by inverting the sequence of bits along that segment.
- Mutation is a random change of the state of a digit along the chromosome.

The selection scheme used by Holland was one where each individual was selected probabilistically in proportion to observed performance (i.e. fitness proportional selection).

Fitness is calculated by evaluating an *objective* or *fitness function*, which assesses the performance of the candidate solution.

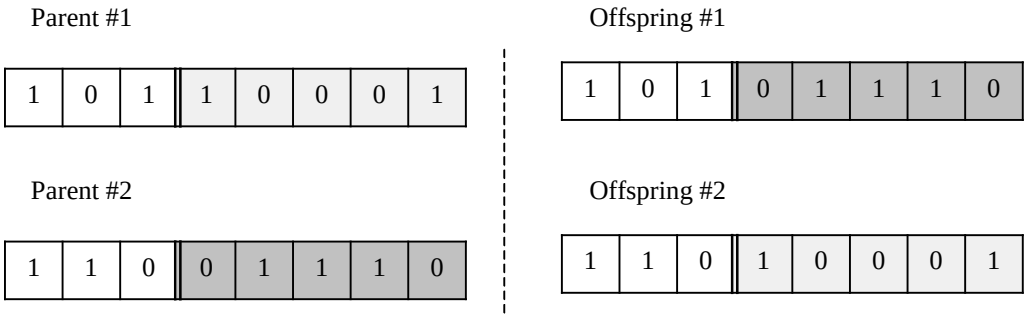

Figure 2: One point cross over in SGA

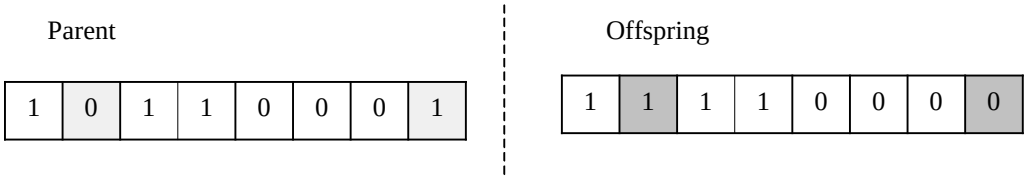

Figure 3: Mutation in SGA

The procedure that is the basis of most modern incarnations of the genetic algorithm is detailed in Goldberg's textbook "Genetic Algorithms in Search, Optimisation, and Machine Learning" [Goldberg, 985] and it is usually referred to as the *simple genetic algorithm* (SGA). The outline of SGA is shown at figure 4.

Simple genetic algorithm (SGA) outline

(1) *Randomly create an initial population of fixed-length binary string chromosomes, set generation count to zero.*

(2) *For each individual chromosome in the population first decode and then calculate its fitness.*

(3) *Select the fittest individuals in the current population in order to create the new population through reproduction and variation using recombination and mutation operations.*

(4) *Replace the current population with the new population*, and increment generation count.

(5) *If termination criteria are satisfied stop, otherwise go to step* 2.

Figure 4: Pseudo-code for the outline of simple genetic algorithm

Given a decoding function *d*, a fitness function *f*, a set of control parameters {*n, l, Pc, Pm* – whose meaning will become clear below} and, a termination criterion *t*, the SGA can be described (more precisely) as follows:

1. *Randomly create an initial population of n l-bit chromosomes.*

The algorithm starts by initialising the *gene pool* (all genes in the population) randomly, in order to scatter (using a uniform distribution) individuals across the landscape of all possible (2^l) binary string configurations.

2. *For each individual chromosome c in the population first decode and then calculate its fitness by evaluating f(d(c)).*

Assigning fitness involves the decoding $d(c) \rightarrow i$ and the evaluation of the fitness function $f(i)$, for each individual structure *i.* The nature of the encoding (*problem representation*), and therefore the decoding, depends on the task at hand. Sometimes the encoding is almost identical with the decoded structure. For example, in function optimisation the binary coding is often a sequence of equally sized binary strings that represent the parameter set of the function to be optimised. The representation of the parameter set might be Boolean or it might use the so-called Gray code [Hollstien, 1971] [Caruana, Schaffer, 1988]. Other problems where it is not just a set of parameter values that is being optimised the representation may be less direct. One interesting example is encoding solutions for the artificial ant problem [Collins, Jefferson, 1991] in which the objective is to evolve an agent (ant) that is able to traverse a terrain which contains food pellets and gather these pellets in optimal time.

3. *Select the fittest individuals in the current population in order to create the new population through reproduction with recombination probability Pc, and mutation probability Pm.*

The selection scheme used, referred to as *roulette wheel* or *fitness proportional* selection, is meant to be analogous with selection in nature where fitter individuals have greater chance to survive and breed. Hence, the probability $p(x)$ of an individual structure x to be selected for reproduction depends on its fitness in relation to the fitness of other individuals in the population and is given by:

$$p(x) = \frac{f(x)}{\sum_{i=1}^{n} f(i)}.$$

The simple genetic algorithm uses two operations to introduce variation in the gene pool: recombination or *crossover* (see figure 2), and mutation (see figure 3). Inversion, originally proposed by Holland, is typically excluded from the SGA. Fitness proportional selection of two

parent individuals with replacement (that is, a parent individual can be selected more than once from the current population) is followed by recombination with probability *Pc* to produce two new offspring. If no recombination takes place (with probability 1-*Pc*) the offspring are replicated from the parents. This is followed by mutation of the offspring at each locus with probability *Pm*. The new offspring are subsequently copied into the new population. The process repeats until the new population contains *n* new chromosomes. In Goldberg's original description of the SGA, the genetic operations of crossover and mutation are applied probabilistically on chromosomes during reproduction with crossover probability *Pc* = 1 (i.e. crossover is always performed), and mutation probability *Pm* = 0.001. The probabilities of crossover and mutation can vary but it is important that there is high crossover and low mutation probability. When the crossover probability is less than one, for example Pc = 0.8, it is possible for individuals to be copied verbatim (*reproduced*) to the next generation, thus implicitly gaining a longer life span.

4. *Replace the current population with the new population.*

This is often referred to as generational GA, that is, progress is achieved through a series of well-defined and separate populations of individuals. However other algorithms exist such as "steady state" algorithms where there are no distinct population intervals [Syswerda, 1991].

5. *If the termination criterion t is satisfied stop, otherwise go to step* 2.

The algorithm iterates several times through steps 2 and 5, and a single iteration is referred to as a *generation*. Typically we need many generations (e.g. form 50 to 500) in order to find an individual structure that is a solution to the problem. A complete sequence of generations through to termination is usually referred to as a run. The termination criterion is usually specified as a fixed number of generations, or an execution time limit. When the algorithm terminates one hopes to arrive either at a solution(s) or at a near solution to the problem, as the population usually tends to converge in exploiting a specific area of the search space that appears to provide optimal solutions or near-optimal solutions for that run. In order to solve a problem we usually perform several runs, typically executed by starting the search anew from a fresh set of uniformly distributed individuals in order to minimise the effects of random initial conditions and *premature convergence*. Premature convergence occurs when the population is trapped in local minimum, by becoming fixated on particular gene combinations and virtually losing almost all other gene variations. (Premature convergence can be regarded as analogous to the phenomenon of niche pre-emption in nature where a biological niche tends to be dominated by a single species [Magurran, 1988] [Koza, 1992, pages 191-192].)

GAs rely mostly on recombination to provide improvements in fitness, whilst mutation is used mainly to maintain variation within the population. The emphasis on recombination is based on

Holland's schemata theorem. Holland argues ibid. the schemata theorem [Holland, 1973] that, in certain cases, GAs provide near optimal use of the information provided by the search so far in order to guide the search in the next generation. By using an analogy with natural evolution, the notion of schemata can be thought of as a collection of certain configurations of genes. A collection of configurations of genes that combine well together to effect an increase in the performance of an individual is known as a "building block". The supposition that recurrent crossover, and sampling through selection pay-off, of short and fit schemata leads to strings of high fitness, is known as the *building block hypothesis.* Given the existence of building blocks, the genetic algorithm can progressively evolve from a random initial population of mostly unfit structures, individuals with chromosomes that contain exponentially larger numbers of useful building blocks, thus evolving fitter structures.

Many of the early practical studies of GA investigated problems of function optimisation [Hollstien, 1971] [De Jong, 1975] [Bethke, 1981] with applications to engineering. Today the GAs have being applied to a wide range of fields, from biology and engineering, to sociological sciences [Gen, Cheng, 1999] [Man, Tang, Kwong, 1999] [Mitchell, 1996]. Several extensions of the SGA exist such as messy-variable length GA, and hierarchical GA, that include new operations, selection methods, and representations [Mitchell, 1996].

3.3 Genetic Programming (GP)

Genetic programming is an extension of the genetic algorithm employed for the automatic generation of computer programs. Several researchers have investigated the induction of computer programs using evolutionary algorithms via different representations [Cramer, 1985] [Fogel, 1999]. However it was John Koza who systematically tested and formalised the use of LISP symbolic expressions (S-expressions) as the representation of choice for "the programming of computers by means of natural selection" [Koza, 1992]. Koza claims genetic programming to be the most general search paradigm in machine learning [Koza, 1992]. Perhaps the most important and most characteristic feature of genetic programming is the fact that solutions to problems are encoded directly as computer programs via the use of hierarchical, LISP-like, symbolic expressions. This feature is responsible for much of the generality of the genetic programming paradigm [Banzhaf, Nordin, Keller, and Francone, 1998, pages 21-22] and, in fact, it can be shown that under certain conditions genetic programming is computationally complete [Teller, 1994].

In order to demonstrate the nature of the encoding in GP, consider the task of program induction where we are asked to discover a program that calculates the area of a circle given its diameter. We could trivially write such a program using C, or LISP as follows:

```
/* calculate the area of a circle in C */
double area(double diameter) {
      double Pi = 3.14;
      return (Pi*(diameter*diameter))/4;
}
;;; calculate the area of a circle in LISP
(defun area (diameter)
      (setf Pi 3.14)
      (/ (* Pi (* diameter diameter)) 4))
```

Apart form the syntactical differences between the two languages, the important part of the program is the fragment where the calculation `(Pi*(diameter*diameter))/4` in C, or `(/(*Pi (*diameter diameter)) 4)` in LISP, takes place. In both cases the above two fragments of code perform the same task and in both cases the order of execution of the calculation involved is the same and can readily be visualised using a hierarchical tree graph (see figure 5). This graph is in fact equivalent to the data structure that most compilers generate internally to represent computer programs before translation into machine code and is usually

referred to as a *parse tree*. LISP S-expressions, because of their simple prefix syntax, can be directly depicted as parse trees. The internal nodes of the tree are referred to as *non-terminal functions* (functions that accept arguments), and the external nodes or leaves as *terminal functions* (functions that accept no arguments or constants) like *Pi* and variable *diameter* in figure 5. The root of the tree is the function appearing first after the left-most parenthesis of the S-expression. Execution of the tree is carried out in a recursive, depth first way, starting from the left. In genetic programming the terms *parse tree* and *S-expression* are used to mean the same thing. In GP the encoding of a solution, and hence the gene pool is made out of parse trees. The nature of the encoding is inherently hierarchical and of variable length in contrast to the SGA where the encoding is linear and of fixed length. It is interesting to notice that, at least, in the case of program induction there is no distinction between genotypes and phenotypes, i.e. between the encoding and the decoded structures.

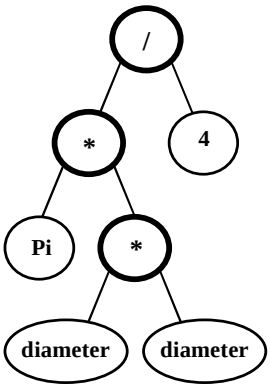

Figure 5. Depicts the parse tree for the S-expression `(/(*Pi(*diameter diameter))4)` used to calculate the area of a circle. The set of functions used in internal nodes of the tree constitutes the non-terminal function set (or *function set*) *F* = {/, *} for this S-expression, whilst the functions used as leaf nodes compose the terminal function set (or *terminal set*) *T* = {2, 4, Pi, diameter}. Discovering a function that calculates the area of circle, can therefore be though of as search through the space of all possible S-expressions generated by composition of functions and terminals that belong to the combined set *C* = *F* U *T*.

The algorithm that forms the basis of most modern incarnations of GP (including Koza's own, ongoing research on GP [Koza, 1994] [Koza, Bennett, Andre, Keane, 1999]) is outlined in [Koza, 1992] and it is, here, referred to as standard genetic programming (*standard* GP).

Standard genetic programming (standard GP) outline

(1) *Set generation count to zero. Generate an initial population of n S-expressions of initial maximum depth Di made out of random compositions of functions from the combined set C = T U F, where T is a set of terminals and F is a set of non-terminal functions*.

(2) *For each individual S-expression in the population evaluate its fitness.*

(3) *Select the fittest individuals in order to create a new population of symbolic expressions through reproduction with probability Pr, recombination with probability Pc, and mutation with probability Pm, where Dc is the maximum allowed depth size of S-expressions created during the run*.

(4) *Replace the current population with the new population*, and increment generation count.

(5) *If the termination criterion t is satisfied stop, otherwise go to step* 2.

Figure 6. Pseudo-code for the standard genetic programming (GP) algorithm outline

Given (the provision of five ingredients) a terminal function set *T*, a non-terminal function set *F*, a fitness function *f*, a set of control parameters {*n, Pc, Pm, Pr, Dc, Di* – the necessity of which will become clearer below}, and a termination criterion *t* standard GP is outlined in figure 6.

Maybe the most important decision that needs to be made (apart from the selection of a fitness function) before starting a GP run is choosing the functions and terminals that are required for the representation of a problem. The choice of functions and terminals is often referred to as the architecture or representation of a GP run. In standard GP, the composite set C, made out of the union of function and terminal sets, has to meet two requirements: The first requirement is that the set C is adequate to solve the problem, a property known as *completeness*. The second is that the set is closed; that is, all functions and terminals (and all their compositions) return values and/or accept arguments of the same data type, a property known as *closure*. The first requirement ensures that the search space contains solutions to the problem and the second ensures that genetic operations (as described below) produce legal parse trees. For example, if we assume that the variable *diameter* is a positive real number other than zero, the composite set C as specified in figure 5 is both closed and complete.

A more precise description with discussion of particular aspects of the algorithm follows.

1. *Generate an initial population of n S-expressions of initial maximum depth Di made out of random compositions of functions from the combined set C = T U F.*

In order to spread the population across a wide variety of parse trees of various sizes and shapes the generation of the new population may be initialised, according to Koza [Koza, 1992], using either "*full*", "*grow*" or "*ramped half and half*" methods.

The full generation method is based on creating the initial population out of S-expressions with each non-backtracking path between a leaf node and the root equal to maximum depth *Di*. The grow method involves generating parse trees with branches extending at variable depths from the root, but with no path between a leaf and the root allowed to exceed depth *Di*. Finally according to the *ramped half and half* method the population is divided into equally sized groups. Each group has a unique associated depth value that belongs to an interval ranging from a minimum depth to the maximum specified depth. Half of the members of each group are created using the grow method and the other half are created using the full method. For example, a population made of 1000 S-expressions, where the minimum depth value is 2 and maximum depth value is 6, will be divided into 5 groups, each group comprising of 200 members and each group associated with maximum depth values 2, 3, 4, 5, and 6 respectively. One hundred members of each group will be generated using "grow", the rest using the "full" method, where the maximum depth for the full and grow methods is the group's associated depth value.

2. *Calculate the fitness of each S-expression x in the population by evaluating f (x).*

Assignment of fitness is explicitly provided by a problem dependent user defined fitness function. Fitness is often evaluated over a set of fitness tests. For example, consider devising a method that tests the performance of evolved S-expressions for the problem of program induction mentioned above; the calculation of the area of a circle given its diameter. The fitness function has to be representative of the problem domain as a whole. We cannot simply test the quality of evolved programs against the area of one circle of specified diameter. Instead we have to test newly generated programs against circles of varying diameters in order to estimate the quality of evolved solutions. Each such comparison, against observed data, is usually referred to as a *test case* or *fitness case*. Sometimes a predefined number of fitness cases are adequate for assessing the quality of a program. For example, we can sample 50 (*x*, *y*) pairs so that $y = \pi \cdot x^2 / 4$, each pair acting as a fitness case. In situations where the problem domain is vast or infinite, new fitness cases may have to be sampled for each generation of a GP run. In our example, this would imply sampling a different set of (*x*, *y*) pairs at each generation of the GP run.

A common method of measuring fitness in GP is the so-called *standardised fitness*, according to which the fittest individual is assigned a value of zero. In effect, standardised fitness measures

error, the less erroneous an individual the better. For example, we could specify the standardised fitness f_s of an S-expression *f* calculating the area of a circle, for each (*x*, *y*) pair described above, as follows:

$$f_s = \sum_{i=1}^{n} | y - f(x) |,$$

where *n* is the number of fitness cases, *x* the diameter, and *f*(*x*) is the area returned by the S-expression *f*. This problem can be seen as a *symbolic regression* problem, that is, given a sampling of data points we are asked to find a function (in symbolic form) that matches the given data, in this case a series of data points generated by the well-known relation $y = \pi \cdot x^2 / 4$ (see figure 7).

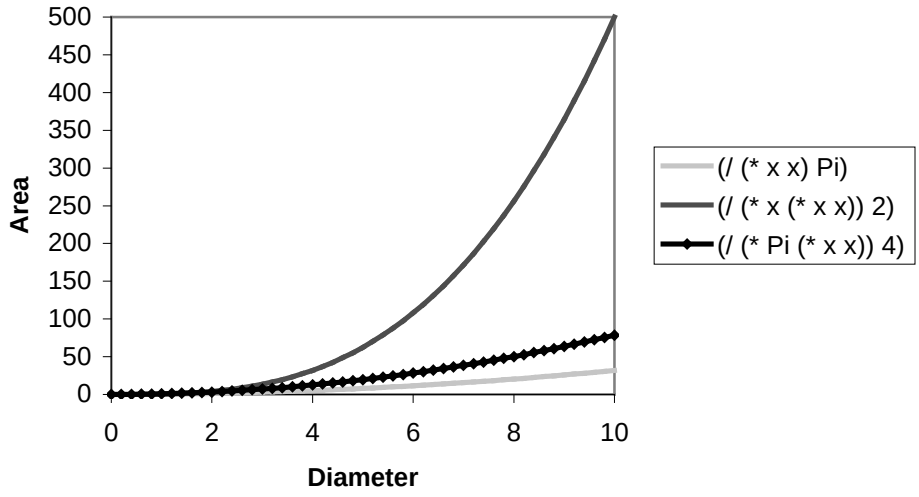

Figure 7. Plots of three S-expressions made out of the functions and terminals specified in figure 5. S-expression (/ (* Pi (* x x)) 4) is a 100% correct solution to the problem of calculating the area of a circle given its diameter, and therefore has standardized fitness 0. The S-expression (/ (* x (* x x)) 2) has standardized fitness 5157.175 thus performing much worse than the S-expression (/ (*x x) Pi) whose standardized fitness is 801.029. The dots of the (/ (* Pi (* x x)) 4) plot represent the fitness cases generated by sampling 50 equally spaced (diameter) values between 0 and 10.

3. *Select the fittest individuals in order to create a new population of symbolic expressions through reproduction with probability Pr, recombination with probability Pc, and mutation with probability Pm, where Dc is the maximum allowed depth size of S-expressions created during the run.*

Selection in *standard* GP is performed mainly using fitness proportional selection with replacement, as in the SGA, actually the user can, if desired, specify other selection methods,

such as *tournament* selection [Koza, 1992, pages 604-606]. Tournament selection is based on competitions or tournaments between a small number of individuals in the population. The number of individuals taking part in a tournament is referred to as the *tournament size*. Tournament individuals are chosen at random and the best individual in a tournament is selected for reproduction. In the simplest case, two individuals are selected at random (tournament size of two) and the best of the two is kept for reproduction. Selection proceeds with replacement, i.e. parents can be re-selected. The fitness pressure can thus be adjusted when using tournament selection, by modifying the tournament size, the larger the tournament size the greater the selection pressure.

Creation of the new population proceeds by choosing one genetic operation amongst the group of available genetic operations, where operations are chosen stochastically depending on their associated probabilities. If reproduction or mutation is chosen, one individual is selected from the current population to create one new offspring. Two individuals are selected from the current population to breed two new offspring in the case of recombination. The new offspring is/are then added to the new population. The process repeats until the new population contains *n* new S-expressions.

The genetic operation of reproduction is defined as replication without modification of an individual from the current population. Corresponding to crossover and mutation in the SGA, the genetic operations of crossover (see figure 8), and mutation (see figure 9) are modified in GP to work with S-expressions. The GP crossover operation, as in GA, is claimed to provide the creative/innovative transformations that lead to adaptation with mutation used as a *secondary* operation in order to introduce variation within the population of S-expressions. This claim is based on extending the schemata theorem and building block hypothesis from GA to GP where schemata are made out of S-expression templates [Langdon, Poli, 2001]. However the notion of building blocks in GP is problematic [O'Reilly, Oppacher, 1995]. This is because GP crossover incurs large changes in the structure of programs. These changes can be substantial enough to disturb building blocks frequently. Empirical observations indicate that GP crossover often behaves as a macro-mutation rather than a structured information exchange akin to the GA one-point crossover [Banzhaf, Nordin, Keller, Francone, 1998, pages 148-156]. As opposed to GA one-point crossover which leads to lexical convergence (since crossing identical individuals produces identical offspring), the lexical structure of S-expressions does not converge via the use of GP crossover even though the behaviour of S-expressions may converge. GP crossover maintains diversity within a population of individuals since crossing identical individuals is likely to produce different offspring [Koza, 1992]. Several new crossover operations have thus been

proposed that allow a more structured information exchange between parse trees such as: context sensitive crossover [D'haeseleer, 1994], one-point GP crossover [Poli, Langdon, 1997], and the use of crossover templates [Jacob, 1996].

In Koza's original description of GP [Koza, 1992] a number of secondary genetic operations are also described. These include (apart from mutation), permutation, editing, encapsulation and decimation. Permutation is a generalisation of the inversion operator, described by Holland (see section on GA). Editing provides the means of simplifying symbolic expressions. Encapsulation allows automatic identification and reuse of potentially useful code segments in S-expressions.

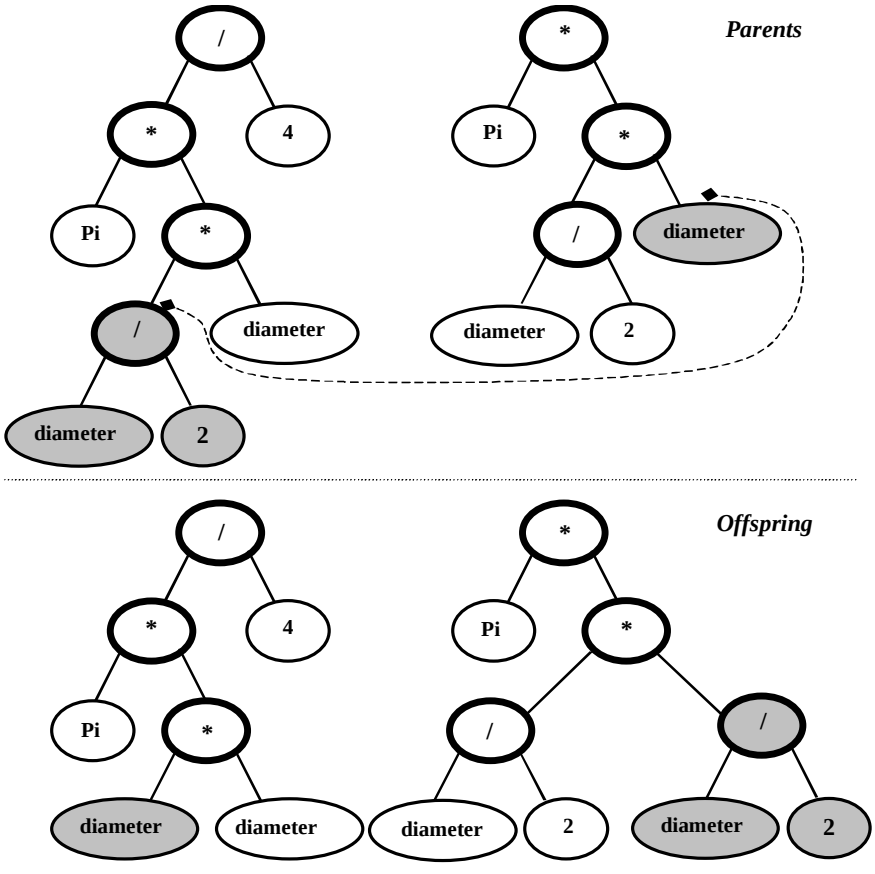

Figure 8. Consider S-expressions made out from the function set F = {*, /} and terminals set T = {Pi, diameter, 2, 4}. The two parent expressions at the top recombine to produce two new offspring depicted at the bottom of the figure. The operation of crossover is defined as selecting a node (usually at random) that marks a sub-tree for each parent individual and then swapping the sub-trees emanating from the selected nodes to create the offspring. The left parent S-expression swaps the sub-tree (*/ diameter* 2) with the terminal (*diameter*) from the right parent to form two offspring each of which is an expression that calculates the area of a circle. Since most nodes of a tree are leaf nodes, selecting nodes is usually biased so that a greater proportion of internal nodes is selected during crossover.

Decimation is used to destroy very low fitness and expensive (in terms of use of computational resources) individuals in a specified generation of a run, typically the first generation. However, only reproduction and crossover are used in the majority of the work by [Koza, 1992], as the initial population is deemed to be large enough to contain enough variation for crossover alone to build working programs. Hence only few experiments described therein use mutation or other secondary operations.

4. *Replace the current population with the new population.*

Like to the SGA algorithm, the standard GP algorithm is generational, although variations of GP exist where the bounds between generations are not distinct, such as *steady state* GP implementations [Reynolds, 1994e] [Banzhaf, Nordin, Keller, Francone, 1998, pages 134-135].

5. *If the termination criterion t is satisfied stop, otherwise go to step* 2

Typically the algorithm terminates after a given number of generations, or when a solution has been found (i.e. an individual found with zero standardised fitness). At this stage the output of the algorithm is the best individual(s) in the population.

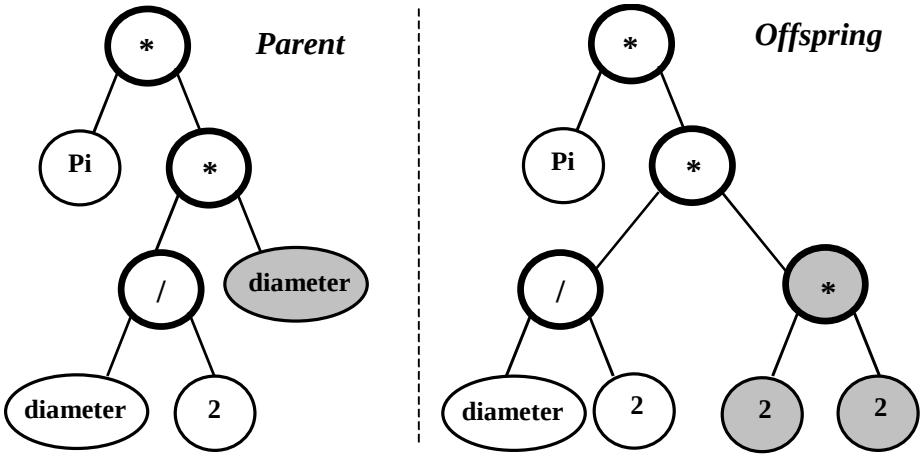

Figure 9. The mutation operation is specified as selecting a node at random from the parent parse-tree and then replacing the sub-tree emanating from that node with a new randomly composed S-expression. In the above the terminal node (*diameter*) is selected from the parent (on the left) and is replaced with the expression (* 2 2) in the offspring on the right.

Koza and others have proposed many extensions to the standard GP algorithm. These include, encapsulation mechanisms that allow for hierarchical problem solving using function decomposition such as *automatically defined functions* (ADFs) [Koza, 1994]. Strongly typed genetic programming (STGP) [Montana, 1995] allows the evolution of programs that do not obey closure, whilst syntactically constrained systems that allow extensions of closure have been proposed by [Koza, 1992, pages 479-526] [Whigham, 1996] [Gruau, 1996b]. Langdon presents a

study of GP using data structures [Langdon, 1998]. Koza et al. [Koza, Bennett, Andre, Keane, 1999] describe a set of genetic operations referred to as *architecture altering operations* that allow the evolution of an S-expression's architecture (i.e. function set, terminal set, and ADFs) as well as its topology. Hybrid algorithms have also been proposed such as GP-ES hybrids [Sarafopoulos, 2001] [Keller, Banzhaf, Mehnen, Weinert, 1999].

Koza's original genetic programming system was written in LISP [Koza, 1992, pages 735-755]. However, the relative simplicity and generality of the paradigm, allowed genetic programming systems to be implemented in many high-level (imperative, object oriented, and logic) programming languages [Langdon, 1998, page 38], as well as in low-level machine code [Banzhaf, Nordin, Keller, Francone, 1998]. An in-house GP system in C++ was used to carry out the experiments described in this chapter.

3.4 Evolution Strategies (ES)

Rechenberg and Schwefel jointly developed evolution strategies as a method for function optimisation inspired by the Darwinian theory of evolution and natural selection. The technique was originally applied to discrete hydrodynamical problems, such as drag minimisation of a joint plate [Rechenberg, 1965], and structure optimisation of a two-phase flashing nozzle [Schwefel, 68] [Klockgether, Schwefel, 1970] [Herdy, 2001]. The application of the technique was initially based on experimental set-up. In the case of the flashing nozzle, for example, joints of the nozzle segments were physically adjusted and the performance of the adjusted nozzle was evaluated. When current adjustment of the construction performed better than previous ones it was used as the basis for further trials. Each adjustment was viewed as mutation of the previous construction, and, founded on the observation that in nature large mutations appear more rarely than small ones, the discrete mutations were sampled by means of a binomial distribution with constant variance. Where traditional optimisation methods failed to provide sufficiently good results this apparently, simple technique was surprisingly successful.

Schwefel was the first to simulate such ES using computers by testing different versions of the strategy for the optimisation of continuous functions of real variables [Schwefel, 1995]. Different types of ES are known as (1+1)-ES, (μ + λ)-ES, and (μ, λ)-ES, in a notation that describes the selection method applied during a generation of individuals where μ parent individuals selected from the current population produce λ offspring. An early incarnation of ES [Schwefel, 1995] is the (1+1)-ES or *two membered ES*, where each generation consists of one *n*-dimensional real valued vector and variation is introduced by normally distributed mutations with expectation zero and given variance. The parent vector is modified using normally distributed mutations with the same standard deviation for each component to generate one offspring. The offspring replaces its predecessor in the event that it improves performance.

Here we concentrate on the (μ + λ)-ES and (μ, λ)-ES that are the most commonly used and which, as opposed to the (1+1)-ES, are population based. In the (μ + λ)-ES case, μ parent individuals produce λ offspring that subsequently compete against themselves and the parents (i.e. μ+λ individuals compete) to form the new generation of μ parent individuals. In the case of (μ, λ)-ES only the λ offspring compete to form the new generation of parent individuals and each individual therefore has a life span of one generation only. The outline of the (μ, λ)-ES algorithm is shown in figure 10. In the case of the (μ + λ)-ES, step (2) of the algorithm outline (in figure 10) has to be modified so that the intermediate population, *Pop*(*t'*), is generated from parents and

offspring, i.e. *Pop*(*t'*) comprises of μ + λ individuals, where parents are copied verbatim into *Pop*(*t'*) form the current population *Pop*(*t*).

Out line of the (μ, λ)-ES algorithm

(1) Set generation count to zero, i.e. $t = 0$, and create initial population of μ parent individuals *Pop*(*0*). Where each member of the population is a vector $\vec{a} \in I = (\vec{x}, \vec{\sigma}) = R^n \times R_+^{n_\sigma}, where\ n_\sigma \in \{1, \ldots, n\}$. Component $\vec{x}$ is a real valued vector that holds *n object variables* encoding the problem at hand. $\vec{\sigma}$ is also a real valued vector of *strategy parameters* storing the standard deviations that specify the mutation step lengths for components of $\vec{x}$.

(2) Create an intermediate population *Pop*(*t'*) made out of λ new offspring, where each new offspring $\vec{a}_i'', i \in \{1, \ldots \lambda\}$ is generated by: firstly using recombination operation *r* on *Pop*(t) to generate $\vec{a}_i'$, i.e. $\vec{a}_i' = (\vec{x}_i', \vec{\sigma}_i') = r(Pop(t))$, and subsequently mutating offspring $\vec{a}_i'$, i.e. $\vec{a}_i'' = (\vec{x}_i'', \vec{\sigma}_i'') = m(\vec{a}_i')$.

(3) Select the most promising μ individuals from *Pop*(*t'*) and store them in the new population of parent individuals *Pop*(*t*+1).

(4) Increment generation count, $t = t + 1$.

(5) If the termination criterion is met stop, otherwise go to step (2).

Figure 10. Pseudo-code for (μ, λ)-ES algorithm outline

ES focus on mutation where recombination is deemed as a secondary operation [Beyer, 1995]. Mutation is usually introduced as normally distributed mutation of the components of the vector to be optimised, where each component is mutated separately. The step lengths (i.e. standard deviations) can be constant or can be adapted during a run. Step length control of mutations is very important. If the step lengths are too long, mutations can produce long random jumps across the fitness landscape that may step over nearby by minima of the landscape. If on the other hand, step lengths are too small, the system might never get close to a distant solution in a reasonable amount of time. If we constantly reduce the step length during the run, we might improve performance towards the end of the run, when arriving close to a possible solution. However, in doing so we run the risk of damaging performance at the beginning of the run, and may need more steps in order to come sufficiently close to a solution that could have been required with a fixed step length.

Clearly there is an exploration-exploitation trade-off, when step control is small we can efficiently search for potential solutions that exist in the vicinity of the current solution, but we lose out on solutions that exist in other parts of the search space. On the other hand, when the step length is long, we explore the fitness potential of the whole landscape more efficiently but we cannot efficiently "fine-tune" the search in order to exploit promising areas. Ideally adaptive mutations learn how big the step length has to be at each stage of the run, so we can maximise performance accordingly.

An example of an adaptive method for controlling the step length was introduced by Rechenberg [Rechenberg, 1973] and is known as the 1/5 success rule [Back, 1996, pages 83-87] which states: "The ratio of successful mutations to all mutations should be 1/5. If it is greater than 1/5, increase the standard deviation, if it is smaller, decrease the standard deviation" [Back, 1996]. Rechenberg derived the 1/5 rule from convergence rate results by analysis of the behaviour of two model objective functions:

$$f_1(\vec{x}) = F(x_1) = c_0 + c_1 \cdot x_1, \forall_i \in \{2,\ldots,n\} : -b/2 \le x_i \le b/2, \text{and}$$

$$f_2(\vec{x}) = c_0 + c_1 \sum_{i=1}^{n} (x_i - x_i^*)^2 = c_0 + c_1 \cdot r^2,$$

f_1, is the so-called *corridor model*, where progress is achieved by moving only along the direction of the x-axis in a corridor of width *b*. f_2, is the *sphere model*, which represents a hypersphere or radius *r*, where the minimum $\vec{x}^*$ is located at the centre of the hypersphere. Convergence rate φ is defined as the expectation of the distance *k'* covered by mutation, that is;

$$\varphi = \int p(k')k'dk',$$

where $p(k')$ is the probability of a mutation moving an individual structure distance k' closer to the optimum [Back, 1996, pages 83-87]. According to the 1/5 success rule, the mutation operation on individual $\vec{a}' = (\vec{x}', \vec{\sigma}')$ is specified as:

$$x_j'' = x_j' + N(0, \sigma_j'),\ j \in \{1,\ldots,n\}, and$$

$$\sigma_j'' = \begin{Bmatrix} c_d\sigma_j' \text{ if } r < \frac{1}{5} \\ c_b\sigma_j' \text{ if } r > \frac{1}{5} \\ \sigma_j' \text{ if } r = \frac{1}{5} \end{Bmatrix}$$

where $N(0,\sigma)$ is a normal distribution with mean zero and standard deviation σ. r is the relative frequency of successful mutations (given by the ratio of successful mutations over all mutations) sampled over an interval of a small number of past generations, e.g. 10. Schwefel suggests [Schwefel, 1995, pages 110-113] that constants c_d and c_b should take values within the interval [0.817, 1].

In self-adaptive (μ, λ)-ES a vector of standard deviations (that represents mutation step lengths) is attached to object variables [Schwefel, 1995, pages 142-145], as described in figure 10. It acts as a set of strategy parameters. An individual's object variables as well as step lengths are then subject to mutations. The expected result of mutations of the step lengths without selection is neutral; increase of step length is as likely as decrease; and, if we sample using a log-normal distribution, small changes occur more often than large ones (sampling using log-normal distribution). Good mutations of step lengths gain, therefore, adaptive advantage through selection pay-off, and poor ones are filtered out. Using this plan an individual $\vec{a}' = (\vec{x}', \vec{\sigma}')$ is mutated by:

$$\sigma_i'' = \sigma_i' \cdot e^{\tau N(0,1)}, \text{ and}$$
$$x_i'' = x_i' + N(0, \sigma_i''), i \in \{1, \ldots, n\},$$

where τ is a constant that represents an overall learning rate, Schwefel suggests [Schwefel, 1995, page 144] that; τ should be inversely proportional to the square root of the number of object variables

$$\tau \propto \frac{1}{\sqrt{n}}.$$

This equation was also studied by Beyer [Beyer, 1996][Beyer, 2001] who provides an improved formula (based on analysis of spherical models) to calculate τ for (μ, λ)-ES, given by:

$$\tau = \frac{c_{\mu,\lambda}}{\sqrt{n}},$$

where $c_{\mu,\lambda}$ is the so-called "progress coefficient", which can be computed numerically for different μ and λ. A table of $c_{\mu,\lambda}$ values, for selected (μ, λ)-coefficients, is provided in [Beyer, 1996].

Schwefel proposed a similar method for self-adapting not only the size of mutation step lengths but also potential linear correlation of mutations of object variables, by introducing another vector of rotation angles as part of the representation [Schwefel, 1995, pages 240-242]. There are $n(n-1)/2$ such rotation angles that correspond to linear correlation of mutations of n object

variables. Rotation angles are self-adapted using additive, normally distributed mutations [Back, 1996, pages 68-73].

Hansen et al. [Hansen, Ostermeir, Gawelczyk, 1995] propose *generating set adaptation* (GSA) as a method that is independent of the co-ordinate system of standard deviations vector $\vec{\sigma}$. This is carried out by linear modification of the basis vectors of $\vec{\sigma}$. A history of successful mutation steps is stored and the weighted sum of all mutation steps of the individual's history is used to alter the co-ordinate basis of $\vec{\sigma}$. This strategy was extended by Hansen and Ostermeir [Hansen, Ostermeir, 1996] with the introduction of *covariance matrix adaptation* (CMA). The use of a covariance matrix allows the technique to be invariant to linear transformations of the search space such as scaling and rotation.

Recombination is often used in ES when $\mu > 1$. Recombination can be applied to object variables as well as strategy parameters. There are several recombination techniques in ES. They are subdivided into sexual and panmictic. In both cases, partners are selected randomly from the parent population. In sexual recombination two parents are randomly selected to generate a new offspring, whereas in panmictic recombination one individual is randomly selected which remains fixed and, for each component (of its vectors), a new partner is selected randomly from the population. Recombination methods are also divided into discrete and intermediate. Discrete recombination of the object variables of individuals in the parent population is defined as:

$$x'_{\mu+1,j} = \begin{cases} x_{\alpha,j} & \text{if } u_j \le 0.5 \\ x_{\beta,j} & \text{if } u_j > 0.5 \end{cases}, \forall j \in \{1,\ldots,n\},$$

where u_j are uniform random variables, sampled for each pair of corresponding components (of parent vectors). The parents are selected once in the case of sexual recombination, whilst the second parent is selected anew for each component in the case of panmictic recombination. Strategy parameters can also be recombined in the same fashion. Intermediate recombination of the object variables of individuals in the parent population is defined as:

$$x'_{\mu+1,j} = x_{\alpha,j} + \chi \cdot (x_{\beta,j} - x_{\alpha,j}),$$

where χ takes a value within the range [0,1]. Typically $\chi = 0.5$, in which case the result obtained is the intermediate value between the two parents. In general intermediate recombination performs a linear interpolation of the two parents. Strategy parameters can also be interpolated in the same fashion. Again, the parents are selected once in the case of sexual recombination, and the second parent is selected anew for each component in the case of panmictic recombination. However, object and strategy parameters don't have to be recombined using the same method. Back suggested that empirically better results are obtained when discrete recombination is

performed on object variables, and intermediate recombination is performed on strategy parameters [Back, 1996, pages 73-76].

Although the representation and genetic operations of GA and ES appear very different, i.e. GA operate on binary strings as opposed to ES operating on real valued vectors, the emphasis of both techniques is on an adaptive search viewed as an exploration-exploitation trade-off. This trade-off in ES is based on the selection of mutation step lengths. In the case of GAs the trade off is "built-in" within the crossover operation. This may be seen, for example, by looking at Holland's analysis for the optimal allocation of trials for the two-armed bandit problem using GA [Goldberg, 1989, pages 36-39] [Holland, 1992, pages 75-88]. Alternatively, one could view GA crossover as an adaptive mutation. To do so, we start with the observation that an initial population of randomly generated binary strings will contain individuals that are lexically very different. This implies that crossover operations will produce very different individuals i.e. mutation step lengths will be very high. However, via selection pressure schemata, (i.e. bit-string templates) that perform well will appear more often in the population. The population is thus likely to start converging, and crossover will produce individuals that are likely to be lexically more similar, i.e. mutation step lengths will progressively reduce.

4 Case studies

We use two case studies to demonstrate the applicability of EAs to computer animation and computer graphics in general. The first case study is based on the evolution of procedural textures, the second on the evolution of iterated function systems. The emphasis in both case studies is on the use of evolutionary algorithms as a tool for the artist and the application of such as tool (using techniques described in the above sections) to the generation of aesthetically pleasing and challenging visual content.

4.1 Interactive Evolution of 2D procedural textures

Here we present an application of evolutionary algorithms to the automatic generation and evolution of procedural textures. This is a popular application of EAs to computer graphics (and animation) and leads to interesting visual results. The images and work presented here are based on the animation sequence "textures" generated by one of the authors [Sarafopoulos, 1995]. In order to use an evolutionary algorithm to generate procedural textures we need to decide on an encoding, fitness function, and genetic operations. In this case we choose genetic programming as the EA of choice and therefore the encoding and genetic operations are based on S-expressions (see section on genetic programming). Using a "conventional" fitness function as in standard GP however can prove problematic. To avoid this we will use "interactive selection" as explained later in this section.

4.1.1 Encoding of procedural textures

Procedural textures [Perlin, 1985][Ebert, Musgrave, Peachey, Perlin, Worley, 1998] in computer graphics are simply functions whose domain is $\Re^2$ (in the case of two-dimensional textures) and range is a vector of intensity or color values. Typically we are looking for a function that maps coordinates to intensity or color values of the form:

$$f : \Re^2 \rightarrow I(R, G, B),$$

where the color components *R*, *G*, *B* belong to the interval [0,1] and correspond to red, green and blue intensities of a pixel at a given location of an image. A procedural texture can readily be coded as an S-expression. Thus, genetic programming could be used to explore the space of such functions. Karl Sims originally proposed this method of generating procedural textures using genetic programming [Sims, 1991]. There are several examples of systems that use a similar approach to generating textures including some commercial applications [Ibrahim, 1998][Wiens, Ross, 2000][Wiens, Ross, 2001].

4.1.2 Interactive selection

In order to drive the evolutionary process we need a way of assigning fitness to individual textures. In our case a small number of textures are originally generated randomly and displayed on the computer screen and the user is asked interactively to assign fitness to the texture(s) they consider to be the most pleasing, for example using mouse-driven interaction. Interactive selection avoids the problem of providing a conventional fitness function. Such a fitness function would probably have to quantify what is seen as visually pleasing (under some context), which is itself a very difficult task.

4.1.3 GP architecture

Here, for simplicity, we are looking at expressions that generate gray-scale images. In order to define the components of symbolic expressions using genetic programming we need to decide on function and terminals sets. In our case we chose the following function set F:

$$F = \{\cos, \sin, +, -, *, /, \text{sqrt}\}.$$

That is, the function set consists of simple arithmetic expressions and few simple functions. The terminal set T consists of:

$$T = \{x, y, ephemeral\},$$

where *x* and *y* are variables that denote pixel coordinates of an image, and *ephemeral* is a terminal that is initialized to contain a random floating-point constant.

Figure 11. The texture on the left is generated by the S-expression (*y*), the texture in the middle of the figure is made out of the S-expression (sin *x*), and the one on the right is generated by (* *x y*).

```
(* (cos (* (cos -38.81) (+ -75.55 (sqrt (+ (+ (sqrt (+ (cos-
38.81) x)) -38.81) (* (cos (cos (cos (* (cos 38.81) (+-75.55 (+
(sqrt (+ (+ (sqrt (+ (cos -38.81) x)) x) (* (cos(cos (cos (sqrt
(+ (+ (sqrt (+ (cos 38.81) x)) x) (* (cos(cos (cos (* (cos -
38.81) (+ -75.55 (+ (sqrt (+ (+ (sqrt (+(cos -38.81) x)) x) (*
(cos(cos (cos (* (cos -15.13) (+-75.55 (+ -75.55 (sqrt (+ -75.55
(* (cos -38.81) y)))))))))y)))(cos (*(cos-38.81) (+ -75.55 (sqrt
(+ (+ (sqrt (+ (cos-38.81) x)) x) (* (cos (cos (cos (* (cos -
15.13) (+ -75.55(+ -75.55 (sqrt (+ -75.55 (* (cos -38.81)
y)))))))))y)))))))))))  y))))))  y)))  (cos  (*  (cos  -38.81)
(+75.55(sqrt (+ (+ (sqrt (+ (cos -38.81) x)) x) (* (cos (cos
(cos(* (cos -15.13) (+ -75.55 (+ -75.55 (sqrt(+ -75.55 (* (cos-
38.81) y)))))))) y)))))))))))) y)))))) 12.40)
```

Figure 12. This complex combination of function nodes and terminals results in the liquid-like texture of figure 13.

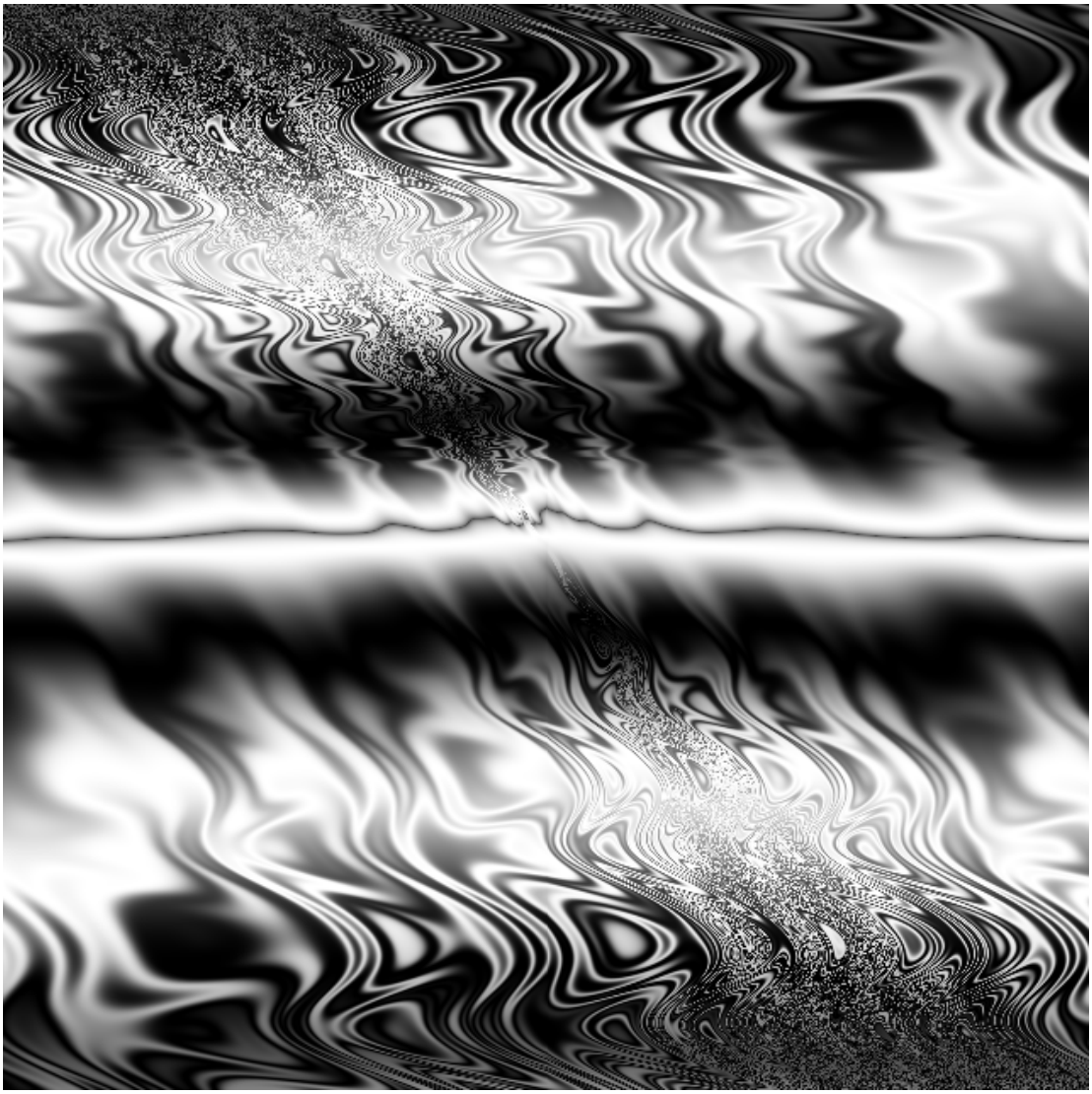

Figure 13.

Resulting expressions can be used to generate a wide variety of textures. However, textures produced by simple S-expressions (made out of one or two function nodes) are also simple see figure 11. The challenge is to discover combinations of functions and terminals that produce pleasing elaborate patterns and textures. Generation of a texture that corresponds to an individual expression proceeds by evaluating that expression for every pixel of the image holding the texture.

4.1.4 Results

After forty or fifty generations, textures that are visually complex start to emerge. A liquid-like texture is presented in figure 13 and the S-expression that generated that texture is given in figure 12. Figure 14 is a screen shot of the interactive system written by one of the authors in order to generate such 2D textures using genetic programming.

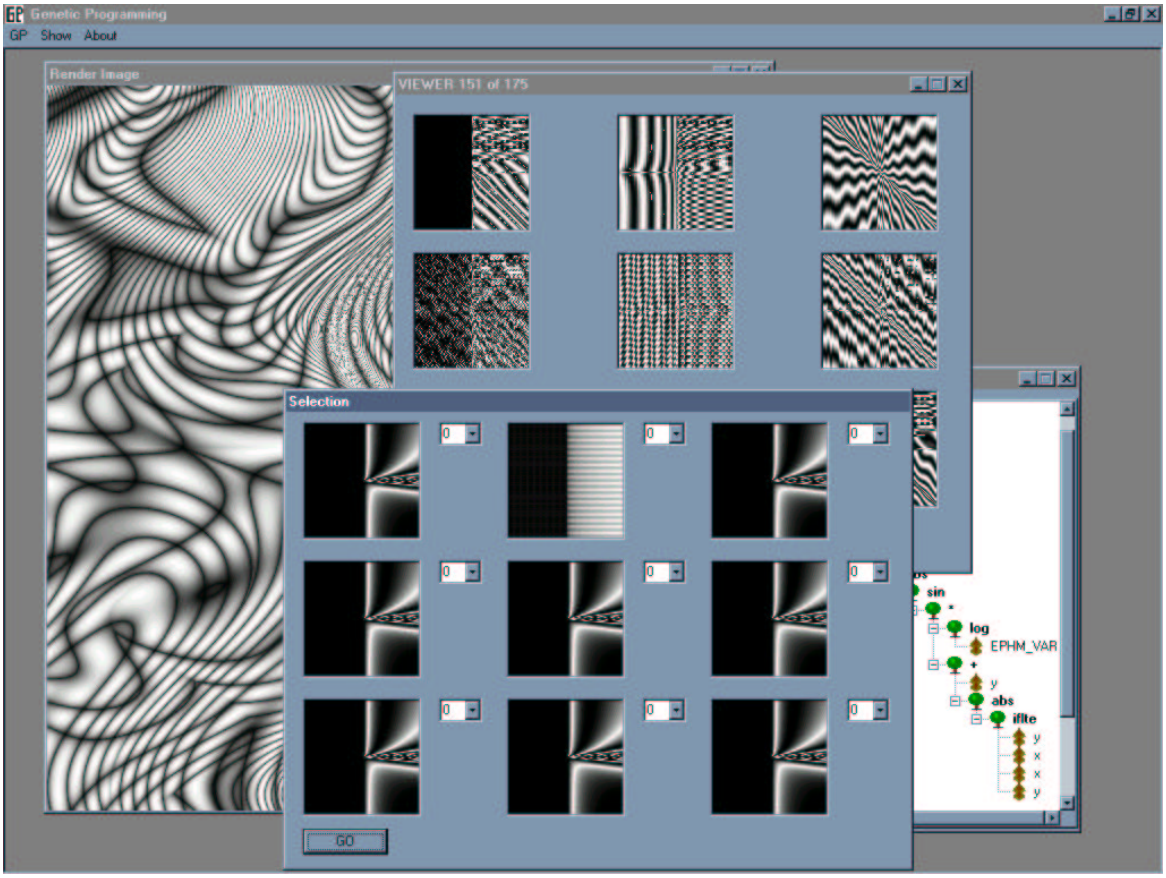

Figure 14.

4.2 Evolutionary Morphing and Iterated Function Systems

Here we focus on the generation of interesting visual content without the use of interactive selection. The question then arises as to how we can guide the generation of images automatically since very little is really known (in scientific terms) about the reasoning underpinning aesthetic judgments. The approach adopted here is one used throughout the history of art, namely "visual representation". An evolutionary algorithm is asked to evolve/generate a shape or a form that is the same as or resembles a shape in nature (in the same way a painter may be asked to represent a natural scene or to draw a portrait, a figure, etc.). If we choose this approach interesting results may be obtained in two respects. First, we don't have to ask for the representational goal to be achieved exactly, thereby allowing for variations on a given theme. Second, the process of reaching the representational goal might itself be interesting, since it will portray a path starting from noise (a random generation of individuals) to a well-defined outcome, i.e., a type of "evolutionary morphing" (between noise and a specific representational goal) could be achieved.

The next challenge is to decide on a symbolic technique that will allow us to approximate a shape or form. There are several options. However, for simplicity and because it is well understood model we use iterated function systems. An iterated function system (IFS) is a fractal mathematical model that can be used to represent natural shapes and structures. IFSs provide a procedural tool for modeling natural objects in computer graphics applications. 2D and 3D objects can be modeled. However, here we restrict ourselves to the representation of 2D fractal-like shapes that are well studied and whose IFS representation is known [Barnsley, Hurd, 1993, pages 82-88].

In order to achieve our aim we use an evolutionary algorithm that is a GP and ES hybrid. Because of the nature of IFSs as arrays of floating point numbers we need an optimization technique well suited to dealing with real valued vectors (i.e. ES). On the other hand, we require a variable length representation since we don't know beforehand the number of codes needed to represent a shape using an IFS (i.e. GP). We call this hybrid approach a *"hierarchical evolution strategy"*.

4.2.1 Hierarchical evolution strategy

Evolution Strategies (ES) operate on real valued vectors. The elements of these real valued vectors that correspond to the coding of a (optimization) problem are referred to as *object variables* (see section on ES). Here using notation from [Back, 1996] an ES individual is defined as a vector a, such as:

$$\vec{a} = (\vec{v}, \vec{s}, \vec{r}) \in I = R^n \times R_+^{n_\sigma} \times [-\pi, \pi]^{n_\alpha},$$
$$n_\sigma \in \{1, \ldots, n\} \qquad \text{(Equation 1)}$$
$$n_\alpha \in \{0, (2n - n_\sigma)(n_\sigma - 1)/2\},$$

where vector *v* represents *object variables*. Each individual also incorporates the *standard deviation* vector *s*, as well as the *correlation coefficient* vector *r*. One of the most important features of ES is variation of *object variables,* which is represented by use of normally distributed mutations (other distributions may also be used for the mutation of object variables). Equally important is the self-adaptation of the *standard deviations* and *correlation coefficients* of the normally distributed mutations [Back, 1996, pages 68-73].

We combine GP and ES by using a method that allow us to encode an ES individual as a LISP symbolic expression (S-expression). In the light of this one could implement ES within a GP system by simply allowing ES individuals to be coded as symbolic expressions, comprised of GP functions and terminals. The structure of an ES program can be seen as a hierarchical tree as illustrated in figure 15. Such a tree representation allows to incorporate ES individuals into GP seamlessly. We call this tree an *evolution strategies' individual* (ES individual). We allow ES style mutation as described by Back [Back, 1996] to operate on ES individuals. We implemented ES individuals, using *strongly typed genetic programming* (STGP) [Montana, 1995] (see following section). STGP recombination can also be applied to ES individuals.

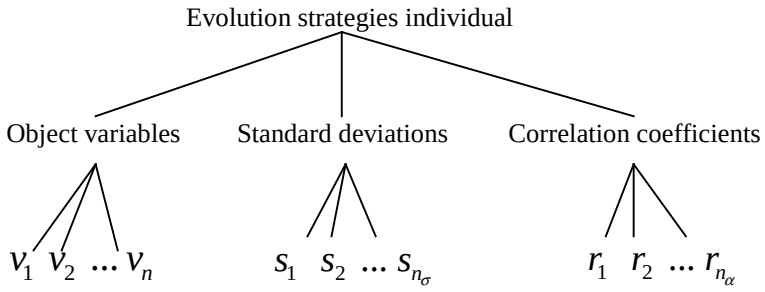

Figure. 15 Tree that defines an ES individual, corresponding to vector *a* of Equation 1

A *hierarchical evolution strategy* (HES) is defined as an S-expression that contains ES individuals. According to this definition, ES itself is a subset of the hierarchical evolution strategy, defined by a single ES individual. Such a structure (as in ES) will be of fixed size. GP architectures that contain scalar terminals could code these terminals as ES individuals. The potential benefit of this is that each ES individual can be fine-tuned separately using ES style mutation operations.

The hierarchical evolution strategy has two major potential benefits: a) it provides a GP

architecture that contains a self adapting mechanism, in terms of mutating scalar terminals; b) it provides a modified ES that is variable length. Variable length coding has been reported in early work using ES [Schwefel, 68]. Nevertheless the architecture has to be described in terms of ES individuals, which have to remain of constant length within the evolutionary simulation cycle.

4.2.2 Strongly typed genetic programming

In GP as described by Koza [Koza, 1992] individuals are S-expressions composed of two types of functions. Functions that accept arguments called *non-terminal functions*, and functions that accept no arguments called *terminal functions*. For brevity we use the term *function* to mean *non-terminal function*. One important constraint of GP is the property of *closure*. That is, all functions have to accept arguments and return values of the same data type. Koza [Koza, 1992] has proposed a method to relax *closure* using *constrained syntactical structures*. We use strongly typed GP [Montana, 1995] to allow S-expressions that are comprised of functions that accept arguments and return values of different data types. The hierarchical evolution strategy is implemented using a STGP system. In STGP we need three sets to describe S-expressions: a set of *data types D*, a set of *functions F*, and a set of *terminals T*. In GP as described by Koza only two sets are needed (*functions* and *terminals*). The notation

$$r \leftarrow f(a_1,\ldots,a_n), where$$
$$r \in D,$$
$$a_i \in D, i \in \{1,\ldots,n\},$$

describes a function *f that* accepts *n* arguments and returns a value of data type *r*, where *D* is the set of data types. If the number of arguments *n* is equal to zero, the above notation describes a terminal returning data type *r*. We use this notation in order to simplify the description of STGP functions and terminals.

4.2.3 Iterated function systems and the inverse problem

IFS offer a method of generating fractal shapes. J. Hutchinson originally developed the idea of IFS [Hutchinson 1981]. An affine IFS is a finite set of contractive affine transformations [Barnsley, 1993] [Barnsley, Hurd, 1993]. Points in space affected by contractive transformations are placed always closer together. It has been shown that a shape can be represented using IFS by discovering a set of contractive transformations that take that shape onto itself. That is, in order to represent a shape *A* using an IFS, the union (*collage*) of the shapes of *A* under the transformations has to generate *A*. This is known as the *collage theorem*. The notation for an IFS is {*wn, n=1,2,…,n*}. Application of an IFS to finite shape *A* is defined as a transformation *W*:

$$W(A) = \bigcup_{i=1}^{n} w_n(A). \quad \text{(Equation 2)}$$

A fractal shape can be constructed by calculating the *attractor* of an IFS. The attractor of an IFS is defined by:

$$\lim_{n \to \infty} W^{\circ n}(A), \quad \text{(Equation 3)}$$

where $W^{\circ n}$ is defined by:

$$A, W(A), W^{\circ 2}(A) = W(W(A)), \ldots, W^{\circ n}(A) = W(W^{\circ n-1}(A)).$$

There are two well-known methods of constructing IFS fractals: the *photocopy algorithm*, and the *chaos game algorithm* [Lu, 1997, pages 34-41]. For a detailed description of IFS see Barnsley [Barnsley, 1993]. A challenging problem when constructing IFS fractals is the inverse or inference problem. This problem can be seen as a restricted form of the general representation problem we mention in the opening section of this case study.

According to the inverse or inference problem for IFS we are asked to find an unspecified number of contractive affine transformations whose attractor is a given image. Here we attempt solve the inference problem for the Barnsley Fern [Barnsley, Hurd, 1993, pages 98-99]. The Barnsley Fern was chosen because of its complex self-similar shape. The fern shape was also chosen because it has a well-known solution that contains four transformations, each different from the others, one of which is singular (i.e. non-invertible). The shape of the fern leaf is one many natural forms whose geometry can be represented using IFSs [Barnsley, Jacquin, Mallassenet, Rueter, Sloan 1988].

4.2.4 Architecture using the hierarchical evolution strategy

An IFS does not contain a predefined number of contractive affine transformations. Encoding is based on S-expressions that represent a list of ES individuals (see figure 15), each ES individual representing an affine transformation. Therefore we use a coding that allows the representation of a variable length list of affine transformations. The coding is based on S-expressions that contain a *list* of ES individuals.

The set of data types *D* for this problem consists of

$$D = \{ifs, map, obj, dev, rot, ephObj_i, ephDev_i, ephRot_j\},$$
$$i \in \{1, \ldots 6\},\ j \in \{1, \ldots 15\}.$$

The data type *obj* stores the *object variables* (see figure 15) of an ES individual. The *object variables* in this case correspond to floating-point coefficients {*a, b, c, d, e, f*} of an affine transformation. *dev* is the data type that stores the *standard deviations* of an ES individual. *rot* is

the data type that stores rotations that stand for *linearly correlated mutation coefficients* of an ES individual. We have six *object variable* coefficients, and six *standard deviation* coefficients. Finally we have *n(n-1)/2 correlation coefficients*, where *n* is the number of object variables. *ephObj* data types, *ephDev* data types, and *ephRot* data types store an ephemeral floating point variable each, that corresponds to an object variable, a standard deviation, and a correlation coefficient respectively. Each coefficient of an ES individual is given a different data type so it could be recombined only with the same coefficient of a different individual.

ifs stores a list of two or more transformations. *ifs* is the return data type of a S-expressions for this problem. The data type *map* stores a list of one or more ES individuals (see figure 15).

The function set *F* for this problem consists of

$$
\begin{aligned}
F = \{ & ifs \leftarrow f\_ifs(map, map), \\
& map \leftarrow f\_list(map, map), \\
& map \leftarrow f\_map(obj, dev, rot), \\
& obj \leftarrow f\_obj(ephObj_1, ephObj_2, \cdots, ephObj_6), \\
& dev \leftarrow f\_dev(ephDev_1, ephDev_2, \cdots, ephDev_6), \\
& rot \leftarrow f\_rot(ephRot_1, ephRot_2, \cdots, ephRot_{15})\}.
\end{aligned}
$$

The function *f_ifs* is used to ensure that IFS have two or more affine transformations. Function *f_list* is used to connect affine transformations in a list.

The terminal set *T* for this problem consists of

$$
\begin{aligned}
T = \{ & ephObj_i \leftarrow t\_objv_i(), \\
& ephDev_i \leftarrow t\_dev_i(), \\
& ephRot_j \leftarrow t\_rot_j()\}, \\
& i \in \{1, \ldots 6\},\ j \in \{1, \ldots 15\}.
\end{aligned}
$$

Functions, terminals, and data types are designed to construct S-expressions that mirror the graph in figure 15.

4.2.5 Fitness function

Two fitness functions were tested, the *Nettleton* fitness, and the *overlap* fitness. The *Nettleton fitness* was also used in [Sarafopoulos, 1999][Sarafopoulos, 2001], and was originally proposed by Nettleton, and Garigliano [Nettleton, Garigliano, 1994]. The *overlap fitness* combines the Nettleton fitness with a calculation of the pixel overlap of affine transformations of an IFS.

Here we deal with (fractal) *shapes* that are black-on-white drawings. Strictly speaking fractal shapes are defined as compact subsets of 2D Euclidean space [Barnsley, 1993]. However, in

terms of the fitness function the universal set *U* is the set of pixels of a 128x128 bitmap. A *shape* is defined as a subset of the set pixels of a 128x128 bitmap. We are searching though the set *H* of all subsets of *U*. The *target shape* or *target* for this experiment is the *shape* of the Barnsley Fern. Let *N(a)* be the number of pixels of *shape a*. Let *collage*, for a given IFS *wn*, be the *shape* created by applying transformation *W* (see Equation 2) to the *target shape*.

We calculate the standardized fitness of individuals, in the Nettleton case by:

$$error = N(target) + N(collage \cap target^{-1}) - N(collage \cap target), \quad \text{(Equation 4)}$$

where a^{-1} is the complement of set *a*. That is, each individual in the population strives to produce a collage that covers pixels of the target shape. The more pixels of the *target* an individual covers and the less of the rest of the image, the greater the reward. Nettleton fitness can lead to the evolution of "opportunistic" transformations. That is, transformations that cover a large portion of the target will tend to spread quickly among the population. However, such transformations are not necessarily optimal, nor do they necessarily lead to a solution. One way to reduce this effect is to introduce a term in the fitness function which punishes individuals that contain transformations that overlap, thus reducing the spread of a transformation that just happens to cover many pixels of the target and therefore encouraging transformations that cover a small number of pixels.

We calculate the standardized fitness in the *overlap* case as *error* + *sharing*, where *error* is defined in Equation 4, and *sharing* is defined by:

$$sharing = \sum_{j=1}^{n} \sum_{\substack{i=1 \\ i \neq j}}^{n} N(w_j(target) \cap w_i(tagret)), \quad \text{(Equation 5)}$$

where *n* is the number of affine transformations of a given IFS.

In order to avoid non-contractive IFS individuals, any IFS individual with contractivity greater than 0.85 was not selected for reproduction.

4.2.6 Control parameters

In the case of the HES coding, the selection scheme was a (μ, λ)-ES type elitist selection with μ=9, and λ=16000. Affine transformations for individuals in the initial population were generated by ES mutation of an affine transformation with {a, b, c, d, e, f} = {0.2, 0.0, 0.0, 0.2, 0.0, 0.0}. Standard deviations were initialized to 0.4. The initial values for correlation coefficients were set to 0. ES style mutation operation was allowed to help fine-tune the parameters of transformations within a given IFS. The ES mutation operation was applied to a randomly selected ES individual of an S-expression. The affine transformations were not directly constrained, and therefore were

not forced to be contractive after mutation. However, in order to reduce mutations that produce non-contractive maps we repeated the mutation operation until a contractive individual was found or a maximum number of iterations were reached. The population number was set to 16000, the generation number was set to 30, and the number of runs to 6.

4.2.7 Results

The results of a successful run are shown in the following figure and they depict the evolution of the fern shape starting with noise, i.e. a random initial population. Each section/image of figure 16 represents the best individual in a generation. Generations increment form top to bottom, and left to right. The only exception to this rule is the image at the top left corner of figure 16, which depicts the target shape. The attractors of the evolved IFSs have been rendered and then posted-processed using a chamfer distance mask [Borgefors, 1984] which creates the shading effect for visualizing the distance of points in an image from the IFS attractor.

5 Conclusions

We demonstrated how evolutionary algorithms could be used in the context of computer graphics as an artistic meta-tool. Only still images are presented here, however the techniques can be extended to generate animation sequences (this could be achieved by interpolation between texture representations or IFS codes). Currently there are two major pathways being explored. The first is the use of EA as a tool for interactive exploration of procedural models, such as textures. The second path involves automatic exploration of features important to computer animation using EA as an optimization tool. In this case strict criteria need to be applied to evolve new forms. This is usually considered problematic if these forms are to claim some aesthetic visual quality. We propose a method that avoids this problem; that is “visual representation”. The aesthetic judgment is made indirectly before starting the process of evolving new forms by deciding on a subject matter and a method of (symbolic) representation that is aesthetically pleasing. We demonstrated this approach by generating a sequence of images that gradually approximate the shape of a fern. This provides a proof of principle that our method can be applied in practice. Currently we are working on techniques that will allow us to improve the efficiently of the EA so shapes and images of greater complexity can be depicted.

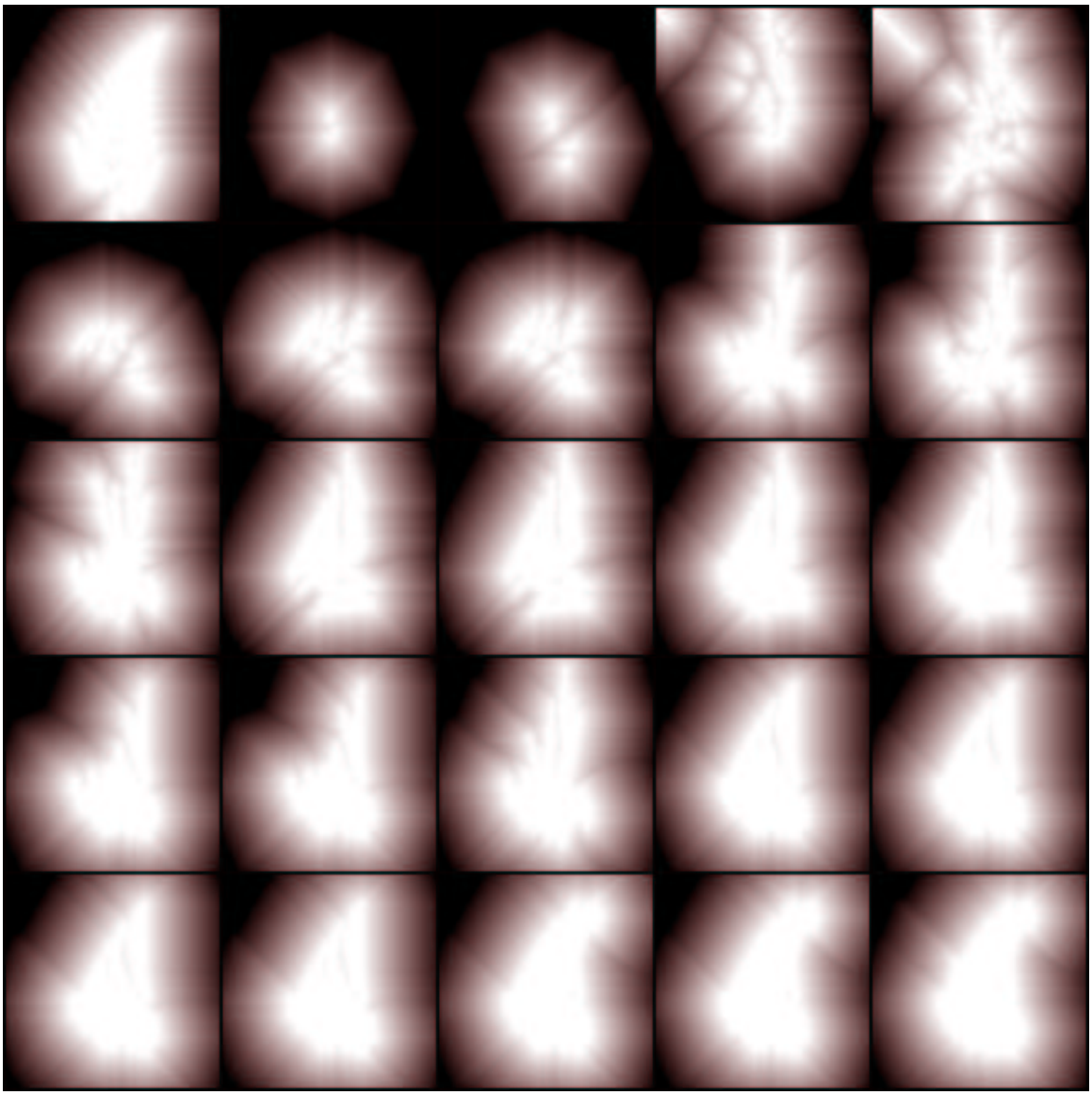

Figure 16. The figure shows the attractors of the best individuals for 24 generations. With the exception of the image at the top left corner, which depicts the target shape, generations increment form top to bottom, and left to right.

Bibliography

[Angeline, 1996] Angeline, J. P. Evolving Fractal Movies, Genetic Programming 1996 Proceedings of the first Annual Conference, pages 503-511, 1996, MIT Press

[Ashlok, D., and Golden, 2000] Ashlok, D., and Golden, J. Iterated Function Systems Fractals for the Detection and Display of DNA Reading Frame, Proceedings of the 2000 Congress on Evolutionary Computation, pages 1160-1167, 2000

[Back, 1996] Back, T. Evolutionary Algorithms in Theory and Practice, 1996, Oxford University Press

[Baluja, Pomerleau, Jochem, 1994] Baluja, S., Pomerleau, D., and Jochem, T. Towards Automated Artificial Evolution for Computer-generated Images, Connection Science, Vol. 6, Number 2 and 3, pages 325-354, 1994

[Banzhaf, Nordin, Keller, Francone, 1998] Banzhaf, W., Nordin, P., Keller, R., and Francone, F. D. Genetic Programming an Introduction, 1998, Morgan Kaufmann

[Barnsley, Jacquin, Mallassenet, Rueter, Sloan 1988] Barnsley, M. F., Jacquin, A., Mallassenet, F., Rueter, L., and Sloan, A. D. Harnessing chaos for image synthesis, Computer Graphics 22(4), pages 131-140, 1988

[Barnsley, 1993] Barnsley, M. F. Fractals Everywhere: 2nd edition, 1993, Academic Press

[Barnsley, Hurd, 1993] Barnsley, M. F., and Hurd, L. P. Fractal Image Compression, 1993, AK Peters

[Bentley, 1999] Bentley, P.J. (ed.) Evolutionary Design by Computers, 1999, Morgan Kaufmann

[Bentley, Corne, 2001] Bentley, P. J. and Corne, D. W. (eds.) Creative Evolutionary Systems, 2001, Morgan Kaufmann

[Bethke, 1981] Bethke, A. D. Genetic Algorithms as Function Optimizers, Ph.D. Dissertation, University of Michigan, Ann Arbor, 1981

[Beyer, 1995] Beyer, H. -G. Toward a theory of evolution strategies: On the benefit of sex - the $(\mu/\mu, \lambda)$ - theory, Evolutionary Computation, 3(1), pages 81-111, 1995

[Beyer, 1996] Beyer, H. -G. Toward a theory of evolution strategies: Self-adaptation, Evolutionary Computation, 3(3), pages 311-347, 1996

[Beyer, 2001] Beyer, H. -G.: The Theory of Evolution Strategies, Natural Computing Series, 2001, Springer

[Borgefors, 1984] Borgefors, G. Distance Transformation in arbitrary dimension, Computer Vision, Graphics, and Image Processing 27, pages 231-345, 1984

[Caruana, Schaffer, 1988] Caruana, R. A., and Schaffer, J. D. Representation and Hidden Bias: Gray vs. Binary Coding for Genetic Algorithms, Proceedings of the 5th International Conference on Machine Learning, 1988, Morgan Kaufmann

[Collet, Lutton, Raynal, Schoenauer, 1999a] Collet, P., Lutton, E., Raynal, F., and Schoenauer, M. Individual GP: an alternative viewpoint for the resolution of complex problems, Proceedings of the Genetic and Evolutionary Computation Conference, volume 2, pages 974-981, 1999, Morgan Kaufmann

[Collet, Lutton, Raynal, Schoenauer, 1999b] Collet, P., Lutton, E., Raynal, F., and Schoenauer, M. Polar IFS + Individual Genetic Programming = Efficient IFS Inverse Problem Solving, Rapport de Recherche INRIA No 3849, December 1993

[Collet, Lutton, Raynal, Schoenauer, 2000] Collet, P., Lutton, E., Raynal, F., and Schoenauer, M. Polar IFS + Parisian Genetic Programming = Efficient IFS Inverse Problem Solving, Genetic Programming and Evolvable Machines Journal, Volume 1, Issue 4, pages 339-361, October 2000

[Collins, Jefferson, 1991] Collins, R., and Jefferson, D. Ant farm: toward simulated evolution, Artificial Life II, Santa Fe Institute studies in the Sciences of the Complexity, Langton C. et al. (eds.), 1991, Addison Wesley

[Cramer, 1985] Cramer, N. L. A representation for the Adaptive Generation of Simple Sequential Programs, Proceedings of an International Conference on Genetic Algorithms and the Applications, pages 183-187, 1985

[Cretin, Lutton, Levy-Vehel, Glevarec, Roll, 1996] Cretin, G., Lutton, E., Levy-Vehel, J., Glevarec, P. and Roll, C. Mixed IFS: Resolution of the inverse problem using genetic programming, Artificial Evolution, volume 1063 of LNCS, pages 247-258, 1996, Springer Verlag

[Darwin, 1895] Darwin, C. The Origin of Species, New American Library, 1859, Mentor paperback

[Dawkins, 1986] Dawkins, R. The Blind Watchmaker, 1986, Harlow Logman

[Dawkins, 1987] Dawkins, R. The Evolution of Evolvability, Artificial Life Proceedings, pages 201-220, 1987

[De Jong, 1975] De Jong, K. A. An analysis of the behavior of a class of genetic adaptive systems, Ph.D. thesis, University of Michigan, Ann Arbor, 1975

[Desmond, 1994] Desmond, S. T. N. An introduction to genetic engineering, 1994, Cambridge University Press

[D'haeseleer, 1994] D'haeseleer, P. Context preserving crossover in genetic programming, Proceedings of the 1994 IEEE World Congress on Computational Intelligence, pages 256-261, vol. 1, 1994, IEEE Press

[Ebert, Musgrave, Peachey, Perlin, Worley, 1998] Ebert, D. S., Musgrave, F. K., Peachey, D., Perlin, K., Worley, S. Texturing and Modeling: A Procedural Approach, Second Edition, 1998, AP Professional

[Fisher, 1995] Fisher, Y. (ed.), Fractal Image Compression: Theory and Application to Digital Images, 1995, Springer Verlag

[Fogel, 1998] Fogel, D. B. Evolutionary Computation, Second edition, 1998, IEEE Press

[Fogel, 1999] Fogel, L. J. Intelligence through Simulated Evolution, 1999, John Wiley & Sons

[Foley, van Dam, Feiner, Huges, 1990] Foley, J. D., van Dam, A., Feiner, S. K., and Huges, J. F. Computer Graphics Principle and Practice, 1990, Addison-Wesley

[Furuta, Maeda, Watanabe, 1995] Furuta, H., Maeda, K. and Watanabe, W. Apllication of Genetic Algorithm to Aesthetic Design of Bridge Structures, In Microcomputers in Civil Engineering, pages 415-421, 1995, Blackwell Publishers, MA, USA

[Graf, Banzhaf, 1995] Graf, J., and Banzhaf, W. Interactive Evolution of Images, Proceedings of Int. Conference on Evolutionary Programming, San Diego, 1995

[Graf, Banzhaf, 1996] Graf, J., and Banzhaf, W. Interactive Evolution for Simulated Natural Evolution, Artificial Evolution, Alliot, J. -M., Lutton, E., Ronald, E., Schoenauer, M., Snyers, D. (eds.), LNCS, Vol. 1063, pages 259-272, 1996, Springer Verlag

[Garigliano, R., Purvis, A., Giles, P. A., and Nettleton, 1993] Garigliano, R., Purvis, A., Giles, P. A., and Nettleton, D. J. Genetic algorithms and shape representation, In D. B. Fogel and W. Atmar, editors, Proceedings of the 2nd Annual Conference on Evolutionary Programming, pages 40-47, Evolutionary Programming Society, 1993

[Gen, Cheng, 1999] Gen, M., and Cheng, R. Genetic Algorithms and Engineering Optimization, 1999, John Wiley & Sons

[Goldberg, 1989] Goldberg, D. E. Genetic Algorithms in Search, Optimization, & Machine Learning, 1989, Addison Wesley

[Goertzel, Miyamoto, Awata, 1994] Goertzel, B., Miyamoto, H., Awata, Y. Fractal Image Compression with the Genetic Algorithm, Complexity International, 1:25-28, 1994, http://www.csu.edu.au/ci/vol1/goertzel.html

[Griffiths, Sarafopoulos, 1999] Griffiths, D. and Sarafopoulos, A. Evolving Behavioural Animation Systems, Artificial Evolution, volume 1829 of LNCS, pages 217-230, 1999, Springer Verlag

[Gritz, Hahn, 1995] Gritz, L. and J. K., Hahn, J. K. Genetic Programming for Articulated Figure Motion, Journal of Visualization and Computer Animation, 6(3), pages 129-142, 1995

[Gritz, Hahn, 1997] Gritz, L. and James K. Hahn, J. K. Genetic Programming Evolution of Controllers for 3-D Character Animation, Genetic Programming 1997: Proceedings of the Second Annual Conference, pages. 139-146, 1997, Morgan Kaufmann

[Gritz, 1999] Gritz, L. Evolutionary Controller Synthesis for 3-D Character Animation, Ph.D. Thesis, The George Washington University, 1999

[Gruau, 1996a] Gruau, F. Modular Genetic Neural Networks for Six-Legged Locomotion, Artificial Evolution, Alliot, J. -M., Lutton, E., Ronald, E., Schoenauer, M., Snyers, D. (eds.), LNCS, Vol. 1063, pages 201-219, 1996, Springer Verlag

[Gruau, 1996b] Gruau, F. On using Syntactic Constraints with Genetic Programming, Advances in Genetic Programming 2, pages 377-394, 1996, MIT Press

[Hamda, Jouve, Lutton, Schoenauer, Sebag, 2000] Hamda, H., Jouve, F., Lutton, E., Schoenauer, M., and Sebag, M. Unstructured Representations in Evolutionary Topological Optimum Design, IJAI, The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies, Special Issue on Creative Evolutionary Systems, Bentley, P. and Corne, D. W. (eds.), 2000

[Hansen, Ostermeir, Gawelczyk, 1995] Hansen, N., Ostermeir, A., and Gawelczyk, A. On the adaptation of arbitrary normal mutation distributions in evolution strategies: the generating set adaptation, In Eshelman, L.J. (ed.) Proceedings of the Six International Conference on Genetic algorithms, pages 57-64, 1995

[Hansen, Ostermeir, 1996] Hansen, N., and Ostermeir, A. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation, In Proceedings of IEEE 1996 International Conference on Evolutionary Computation, pages 312-317, 1996

[Herdy, 2001] Herdy, M. Optimization of a Two-Phase Nozzle with an ES, EvoNet Flying Circus Demo, http://www.wi.leidenuniv.nl/~gusz/Flying_Circus/3.Demos/Movies/Duese/index.html, 2001

[Holland, 1973] Holland, J. Genetic algorithms and the optimal allocation of trials, SIAM Journal on Computation, vol. 2, pages 88-105, 1973

[Holland, 1992] Holland, J. H.: Adaptation in Natural and Artificial Systems, Second edition, 1992, MIT Press

[Hollstien, 1971] Hollstien, R. B. Artificial genetic adaptation in computer control systems, Ph.D. thesis, University of Michigan, Ann Arbor, 1971

[Hoskins, Vagners, 1992] Hoskins, D. A., and Vagners, J. Image Compression Using Iterated Function Systems and Evolutionary Programming: Image Compression without Image Metrics, Proceedings of the 26th Asilomar Conference on Signals, Systems, and Computers, Vol. 2, pages 705-711, 1992, IEEE Press

[Hutchinson 1981] Hutchinson, J.E. Fractals and Self-Similarity, Indiana University Journal, Vol. 35, No. 5, 1981

[Ibrahim, 1998] Ibrahim, A. E., Genshade: An Evolutionary Approach to Automatic and Interactive Procedural Texture Generation, Doctoral Thesis, Office of Graduate Studies of Texas A&M University, 1998

[Jacob, 1996] Jacob C. Evolving Evolution Programs: Genetic Programming and L-Systems, Genetic Programming 1996: Proceedings of the first Annual Conference, pages 107-115, 1996, MIT Press

[Kang, Cho, Lee, 1999] Kang, Y. -M., Cho, H. –G., and Lee, E –T. An efficient control over human running animation with extension of planar hopper model, The Journal of Visualization and Computer Animation, 10(4), pages 215-224, 1999

[Keller, Banzhaf, Mehnen, Weinert, 1999] Keller, R.E., Banzhaf, W., Mehnen, J., and Weinert, K. CAD surface reconstruction from digitized 3D point data with a genetic programming/evolution strategy hybrid, Advances in Genetic Programming 3, pages 41-65, 1999, MIT Press

[Kimura, 1983] Kimura, M. The Neutral Theory of Molecular Evolution, 1983, Cambridge Univ. Press

[Kirpatrick, Gelatt, Vecchi, 1983] Kirpatrick, S., Gelatt, C. D., Vecchi, M. P. Optimization by simulated annealing, Science 220, pages 671-680, 1983

[Klockgether, Schwefel, 1970] Klockgether, J. and Schwefel, H.-P. Two-phase nozzle and hollow core jet experiments, In D. G. Elliott, (ed.), Proc. 11th Symp. Engineering Aspects of Magnetohydrodynamics, California Inst. of Technology, Pasadena CA, pages 141-148, March 1970

[Koza, 1992] Koza, J.R. Genetic Programming: On the Programming of Computers by Means of Natural Selection, 1992, MIT Press

[Koza 1994] Koza, J.R. Genetic Programming II: Automatic Discovery of Reusable programs, 1994, MIT Press

[Koza, Bennett, Andre, Keane, 1999] Koza, J.R., Bennett III, F. H., Andre, D., and Keane, M. A. Genetic Programming III: Darwinian Invention and Problem Solving, 1999, MIT Press

[Langdon, 1998] Langdon, W. B. Genetic Programming and Data Structures, 1998, Kluwer Academic Publishers

[Langdon, Poli, 2001] Langdon, W. B., and Poli, R. Foundations of Genetic Programming, 2001, Springer

[Lankhorst, 1996] Lankhorst, M. M. Genetic Algorithms in Data Analysis, Ph.D. thesis, University of Groningen, 1996

[Levy-Vehel, Lutton, 1993] Levy-Vehel, J., and Lutton, E. Optimization of Fractal Functions using Genetic Algorithms, Rapport de Recherche INRIA No 1941, June1993

[Levy-Vehel, Lutton, 1994] Levy-Vehel, J., and Lutton, E. Optimization of Fractal Functions using Genetic Algorithms, Proceedings of Fractal'93, London, Sept 1993, Fractals in the Natural and Applied Science (A-41), Novak, M. M. (ed.), pages 275-285, 1994, Elsevier Science

[Lewis, 2000] Lewis, M. Aesthetic Evolutionary Design with Data Flow Networks, presented at 4th International Conference and Exhibition on Generative Art 2000, http://www.accad.ohio-state.edu/~mlewis/

[Lewis, 2001] Lewis, Visual Aesthetic Evolutionary Design Links, http://www.accad.ohio-state.edu/~mlewis/aed.html, 2001

[Lim, Thalmann, 1999] Lim, I. S., and Thalmann, D. How Not to Be a Black-Box: Evolution and Genetic Engineering of High-Level Behaviours, Proceedings of the Genetic and Evolutionary Computation Conference, Vol. 2, pages 1329-1335, 1999, Morgan Kaufmann

[Lu, 1997] Lu, N. Fractal Imaging, 1997, Academic Press

[Lutton, Cretin, Levy-Vehel, Glevarec, Roll, 1995] Lutton, E., Cretin, G., Levy-Vehel, J., Glevarec, P., Roll, C. Mixed IFS: resolution of the inverse problem using Genetic Programming, Complex Systems, Vol. 9, No 5, pages 375-398, 1995

[Lutton, 1999a] Lutton, E. Genetic Algorithms and Fractals - Algorithmes Génétiques et Fractales, Dossier d'Habilitation à diriger des recherches, Université Paris XI Orsay, Spécialité Informatique, 11 February 1999

[Lutton, 1999b] Lutton, E. Genetic Algorithms and Fractals, In Evolutionary Algorithms in Engineering and Computer Science, K. Mietttinen, M. M. Mäkelä, P. Neittaanmaki, J. Périaux, (eds, 1999), John Wiley & Sons

[Lund, Pagliarini,Miglino, 1995] Lund, H., Pagliarini, L., and Miglino, O. Artistic Design with GA and NN, Proc. of the 1st Nordic Workshop on Genetic Algorithms and Their Applications (1NWGA), Uni. Vaasa, Finland, xiii+417 pages 97-105, 1995

[Magurran, 1988] Magurran, A. E. Ecological diversity and its measurement, 1988, Princeton University Press

[Man, Tang, Kwong, 1999] Man, K. F., Tang, K. S., and Kwong, S. Genetic Algorithms, 1999, Springer

[Martin, Hine, 2000] Martin, E. and Hine, R. S. (eds.), A Dictionary of Biology, 2000, Oxford University Press Market House Books

[Mitchell, 1996] Mitchell, M. An introduction to Genetic Algorithms, 1996, MIT Press

[Montana, 1995] Montana, J. D. Strongly Typed Genetic Programming, Evolutionary Computation, 3(2), pages 199-230, 1995

[Nettleton, Garigliano, 1994] Nettleton, D. J. and Garigliano, R. Evolutionary algorithms and the construction of fractals: solution of the inverse problem, Biosystems (33), pages 221-231, 1994, Elsevier Science

[Nettleton, 1995] Nettleton, D. J. Evolutionary Algorithms in Artificial Intelligence: A Comparative Study through Applications, Ph.D. thesis, University of Durham, Department of Computer Science, UK, 1995

[Nettleton, Garigliano, 1995] Nettleton, D. J. and Garigliano, R. Evolving fractals, Jour. of Computers and Graphics, Vol. 19, No. 5, pages 779-782, 1995, Pergamon Press

[Nettleton, Garigliano, 1996] Nettleton, D. J. and Garigliano, R. Reductions in the search space for deriving a fractal set of an arbitrary shape, Jour. of Mathematical Imaging and Vision, Vol. 6, No. 4, pp. 379-393,1996, Kluwer

[O'Reilly, Oppacher, 1995] O'Reilly, U-.M., and Oppacher, F. The Troubling Aspects of a Building Block Hypothesis for Genetic Programming, In L. D. Whitley, and M. D. Vose, (eds), Foundations of Genetic Algorithms 3, pages 73-88, 1995, Morgan Kaufmann

[Peachey, 1985] Peachey, D. Solid Texturing of Complex Surfaces, Computer Graphics Vol.19, No.3, pages 279-286, 1985

[Pelikan, Goldberg, Tsutsui, 2001] Pelikan, M., Goldberg, D. E. and Tsutsui, S. Combining the Strengths of the Bayesian Optimization Algorithm and Adaptive Evolution Strategies, IlliGAL Report No. 2001023, University of Illinois, 2001.7

[Perlin, 1985] Perlin, K. An Image Synthesizer, Computer Graphics, Vol.19, No.3, pages 287-296, 1985

[Poli, R., and Cagnoni, 1997] Poli, R., and Cagnoni, S. Evolution of Psuedo-colouring Algorithms for Image Enhancement with Interactive Genetic Programming, Proceedings of the Second International Conference on Genetic Programming, GP'97, pages 269-277, 1997, Morgan Kaufmann

[Poli, Langdon, 1997] Poli, R., and Langdon, W. B. Genetic Programming with One-Point Crossover, In P. K. Chawdhry and R. Roy and R. K. Pan, (eds), Soft Computing in Engineering Design and Manufacturing, pages 180-189, 1997, Springer Verlag London

[Provine, 1986] Provine, W. B. Sewall Wright and Evolutionary Biology, 1986, The University of Chicago Press

[Racine, Hamida, Schoenauer, 1999] Racine, A., Hamida, S.B., and Schoenauer, M. Parametric coding vs genetic programming: A case study, In W. B. Langdon, Riccardo Poli, Peter Nordin, and Terry Fogarty (eds.), Late-Breaking Papers of EuroGP-99, pages 13-22, Goteborg, Sweeden, 1999

[Raynal, Lutton, Collet, Schoenauer, 1999] Raynal, F., Lutton, E., Collet, P., Schoenauer, M. Manipulation of Non-Linear IFS attractors using Genetic Programming, CEC99, Proceedings of the Congress on Evolutionary Computation, Washington DC, USA, Vol. 2, pages 1171-1177, 1999, IEEE Press

[Rechenberg, 1965] Rechenberg, I. Cybernetic solution path of an experimental problem, Royal Aircraft Establishment, Library translation No. 1122, Farnborough, Hants., UK, August 1965

[Rechenberg, 1973] Rechenberg, I., Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, 1973, Frommann-Holzboog, Stuttgart

[Redmill, Bull, Martin, 1996] Redmill, D. W. Bull, D. R. and Martin, R. R. Genetic algorithms for fast search in fractal image coding, In R. Ansari and M. J. Smith, editors, Visual Communications and Image Processing '96, volume 2727, pages 1367-1376, SPIE Proceedings, 1996

[Reynolds 1992] Reynolds, C. W. An Evolved, Vision-Based Behavioral Model of Coordinated Group Motion, From Animals to Animats (Proceedings of Simulation of Adaptive Behaviour), 1992, MIT Press

[Reynolds 1994a] Reynolds, C. W. An Evolved Vision-Based Behavioral Model of Obstacle Avoidance Behaviour, Artificial Life III, SFI Studies in the Sciences of Complexity, Vol. XVII, pages 327-346, 1994, Addison-Wesley

[Reynolds 1994b] Reynolds, C. W. Evolution of Obstacle Avoidance Behaviour: Using Noise to Promote Robust Solutions, Advances in Genetic Programming, pages 221-241, 1994, MIT Press

[Reynolds 1994c] Reynolds, C. W. The difficulty of roving eyes Proceedings of the 1994 IEEE World Congress on Computational Intelligence, pages 262-267,1994, IEEE Press

[Reynolds 1994d] Reynolds, C. W. Competition, Coevolution and the Game of Tag, Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, pages 59-69, 1994, MIT Press

[Reynolds 1994e] Reynolds, C. W. Evolution of Corridor Following Behavior in a Noisy World Simulation of Adaptive Behaviour (SAB-94), 1994

[Ridley, 1996] Ridley, M. Evolution, Second edition, 1996, Blackwell Science

[Russell, Norvig, 1995] Russell, S., Norvig, P. Artificial Intelligence a Modern Approach, 1995, Prentice Hall International

[Sarafopoulos, 1995] Sarafopoulos, A. Textures, Animation sequence shown at the “Cabaret électronique” during the Sixth International Symposium on Electronic Art in Montreal, Canada, 1995

[Sarafopoulos, 1999] Sarafopoulos, A. Automatic generation of affine IFS and strongly typed genetic programming, Genetic Programming Proceedings of EuroGP1999, volume 1598 of LNCS, pages 149-160, 1999, Springer-Verlag

[Sarafopoulos, 2001] Sarafopoulos, A. Evolution of Affine Transformations and Iterated Function Systems Using Hierarchical Evolution Strategy, Genetic Programming Proceedings of EuroGP2001, volume 2038 of LNCS, pages 176-191, 2001, Springer-Verlag

[Saupe, Ruhl, 1996] Saupe, D. and Ruhl, M. Evolutionary fractal image compression, In Proceedings ICIP-96 (IEEE International Conference on Image Processing), volume I, pages129-132, Lausanne, Switzerland, September 1996

[Schoenauer, Lamy, Jouve, 1995] Schoenauer, M., Lamy, B., and Jouve, F. Identification of mechanical behaviour by genetic programming part II: Energy formulation, Technical report, Ecole Polytechnique, 91128 Palaiseau, France, 1995

[Schoenauer, Sebag, Jouve, Lamy, Maitournam, 1996] Schoenauer, M., Sebag, M., Jouve, F., Lamy, B., and Maitournam, H. Evolutionary identification of macro-mechanical models, Advances in Genetic Programming 2, pages 467-488, 1996, MIT Press

[Schwefel, 68] Schwefel, H.-P. Experimentelle Optimierung einer Zweiphasend\"use Teil I, AEG Research Institute, Berlin, Technical Report No.~35 of the Project MHD-Staustrahlrohr, No.11.034/68, 1968

[Schwefel, 1995] Schwefel, P. H. Evolution and Optimum Seeking, 1995, John Wiley & Sons

[Sedivy, Joyner, 1992] Sedivy, J. M., and Joyner, L. A. Gene Targeting, 1992, Oxford University Press

[Shonkwiler, Mendivil, Deliu, 1991] Shonkwiler, R., Mendivil, F., Deliu, A. Genetic Algorithms for the 1-D Fractal Inverse Problem, Proceedings of the Fourth International Conference on Genetic Algorithms, San Diego, pages 495-501, 1991

[Sims, 1991] Sims, K. Artificial Evolution for Computer Graphics, Computer Graphics Siggraph '91 proceedings, pages 319-328, 1991

[Sims, 1992] Sims, K. Interactive Evolution of Dynamical Systems, Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life, pages 171-178, 1992, MIT Press

[Sims, 1993] Sims, K. Interactive Evolution of Equations for Procedural Models, The Visual Computer, pages 466-476, 1993

[Sims, 1993] Sims, K. Genetic Images, Media installation allowing the interactive evolution of abstract still images, Exhibited at the Centre Georges Pompidou in Paris, Ars Electronica in Linz, Austria, and the Interactive Media Festvial in Los Angeles, 1993

[Sims, 1994a] Sims, K. Evolving Virtual Creatures, Computer Graphics Siggraph '94 Proceedings, pages 15-22, 1994

[Sims, 1994b] Sims, K. Evolving 3D Morphology and Behavior by Competition, Artificial Life IV Proceedings, Brooks and Maes (eds.), pages 28-39, 1994, MIT Press

[Sims, 1997] Sims, K. Galápagos, Media installation allowing museum visitors to interactively evolve 3D animated forms, Exhibited at the ICC in Tokyo and the DeCordova Museum in Lincoln Mass, 1997

[Sharman, Esparcia-Alcazar, Li, 1995] Sharman, K. C., Esparcia-Alcazar, A. I., and Li, Y. Evolving signal processing algorithms by genetic programming, First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA, IEE, volume 414, pages 473-480, 1995

[Syswerda, 1991] Syswerda, G. A Study of Reproduction in Generational and Steady-State Genetic Algorithms, in *Foundations of Genetic Algorithm*s, Rawlins G. J. E. (ed.), pages 94-101, 1991, Morgan Kaufmann

[Teller, 1994] Teller, A. Turing Completeness in the Language of Genetic Programming with Indexed Memory, Proceedings of the 1994 IEEE World Congress on Computational Intelligence, Vol. 1, pages 136-141, 1994, IEEE Press

[Todd, Latham, 1991] Todd, S., and Latham, W. Mutator, a Subjective Human Interface for Evolution of Computer Sculptures, IBM United Kingdom Scientific Centre Report 248, 1991

[Todd, Latham, 1992] Todd, S. and Latham, W. Evolutionary Art and Computers, 1992, Academic Press

[Vences, Rudomin, 1994] Vences, L. and Rudomin, I. Fractal Compression of single images and image sequences using genetic algorithms, The Eurographics Association, 1994

[Vrscay, 1990] Vrscay, E. Moment and Collage Methods of the Inverse Problem of Factal Construction with Iterated Function Systems, Proceedings of 1st IFIP Conference on Fractals, FRACTAL 90, Lisbon 1990

[Vrscay, 1991] Vrscay, E. Iterated Function Systems: Theory, Applications and the Inverse Problem, in Fractal Geometry and Analysis, Belair, J. Dubuc, S. (eds.), pages 405-468, 1991, Kluwer Academic

[Watt, Watt, 1992] Watt, A., and Watt, M. Advanced Animation and Rendering Techniques Theory and Practice, 1992, Addison-Wesley

[Whigham, 1996] Whigham, P. A., Grammatical Bias for Evolutionary Learning, Ph.D. Thesis, School of Computer Science, University College University of New South Wales, Australian Defence Force Academy, 1996

[Wiens, Ross, 2000] Wiens, A. L., and Ross, B., J. Gentropy: Evolutionary 2D Texture Generation, late breaking papers at the 2000 Genetic and Evolutionary Computation Conference, pages 418-424, 2000

[Wiens, Ross, 2001] Wiens, A. L., and Ross, B., J.Gentropy: Evolutionary 2D Texture Generation, Computers and Graphics Journal (in press), 2001

[Wright, 1931] Wright, S. Evolution in Mendelian populations, Genetics, vol. 16, pages 97-159, 1931

[Xuan, Dequn, 1996] Xuan, Y., and Dequn, L. An improved genetic algorithm of solving IFS code of fractal image, In Proceedings of the 3rd International Conference on Signal Processing, volume 2, pages 1405-1408, Beijing (China), October 1996, IEEE, New York

[Zhao, Wang, 1998] Zhao, K., and Wang, J. Path Planning in Computer Animation Employing Chromosome-Protein Scheme, Genetic Programming 1998, Proceedings of the Third Annual Conference, pages 439-447, 1998, Morgan Kaufmann