THE USE OF GENETIC PROGRAMMING IN EXPLORING 3D DESIGN WORLDS

A REPORT OF TWO PROJECTS BY MSC STUDENTS AT CECA UEL

T.Broughton, A.Tan, P S Coates
Centre for Environment and Computing in Architecture,
University of East London.

Abstract

Genetic algorithms are used to evolve rule systems for a generative process, in one case a shape grammar,which uses the "Dawkins Biomorph" paradigm of user driven choices to perform artificial selection, in the other a CA/Lindenmeyer system using the Hausdorff dimension of the resultant configuration to drive natural selection.

1) Using Genetic Programming in an interactive 3d shape grammar (AmyTan &P.S..Coates) A report of a generative system combining genetic programming(GP) and 3D shape grammars. The reasoning that backs up the basis for this work depends on the interpretation of design as search In this system, a 3D form is a computer program made up of functions (transformations) & terminals (building blocks). Each program evaluates into a structure. Hence, in this instance a program is synonymous with form. Building blocks of form are platonic solids (box, cylinder....etc.). A Variety of combinations of the simple affine transformations of translation, scaling, rotation together with Boolean operations of union, subtraction and intersection performed on the building blocks generate different configurations of 3D forms. Using to the methodology of genetic programming, an initial population of such programs are randomly generated,subjected to a test for fitness (the eyeball test). Individual programs that have passed the test are selected to be parents for reproducing the next generation of programs via the process of recombination.

2) Using a GAto evolve rule sets to achieve a goal configuration(T.Broughton & P.Coates) . The aim of these experiments was to build a framework in which a structure's form could be defined by a set of instructions encoded into its genetic make-up. This was achieved by combining a generative rule system commonly used to model biological growth with a genetic algorithm simulating the evolutionary process of selection to evolve an adaptive rule system capable of replicating any preselected 3-D shape. The generative modelling technique used is a string rewriting Lindenmayer system the genes of the emergent structures are the production rules of the L-system, and the spatial representation of the structures uses the geometry of iso-spatial dense-packed spheres

1) Using Genetic Programming in an interactive 3d shape grammar

This paper attempts to show how evolving creative forms can be cast as a problem of induction. And more importantly, that there is a way to solve the problem of induction -- by genetic programming. The genetic programming paradigm described in the following provides a way to do program induction. The basic idea of program induction is the inductive discovery of a computer program from a space of possible computer programs. In our case the computer program is a sequence of operations or a composition of functions that evaluates to a geometric output.

Given that the computers are more effective in generating proposals according to rules and control strategies, the human critic is used to inspect and test them. There is no question that there are many analysis programs that will do the task of evaluating the performance of each proposal. As the main purpose of this paper is to explore and create imaginative and beautiful forms, the proposals must pass the ‘eye’test. Criterion for high performance is that it must appeal to the critic enough for it to be selected to evolve further. The final structure is the form selected that will not be bred any further.

The structure of the program is equivalent to the form because it contains all the information that will be manifested in the geometric properties of the form. When reference is made to the computer program, it implies the structure.

Mapping of the code to a virtual model is simple and direct. Three dimensional geometrical shapes pose no problem to programs that use the constructive solid geometry paradigm, such as AUTOCAD and a host of others too numerous to mention. What better way is there to represent properties of geometrical objects than to use the in-built functions of CAD software. Because AUTOLISP, a version of LISP unique to AutoCad, has been employed to implement the genetic programming paradigm, the result or emergent computer program that has evolved can be easily translated into its visual geometric form using the EVAL function in AutoLISP. The process of genetic programming takes on a search for highly fit individual computer program in a space of computer programs. In particular, the search space is the space of all possible computer programs composed of functions and terminals appropriate to the problem domain. Breeding using reproduction of the fittest along with genetic recombination (crossover) operation appropriate for mating computer program applies. A computer program that solves, or approximately solves a given problem (or meets a target) may emerge from this combination of Darwinian natural selection and genetic operations.(Koza, Genetic programming - on the programming of computers by natural selection, MIT, 1992)

The set of possible structures in genetic programming is the set of all possible compositions of functions that can be composed recursively from the set of Nfunc functions from F = { f1, f2, ..., ffunc} and the set of Nterm terminals from T = { a1, a2, ..., afunc }. Each particular function fi in the function set F takes a specified number of z(fi) of arguments z(f1), z(f2), ..., z(fNfunc).

The function in the function set may include

* arithmetic operations (+, -, *, etc.),

* mathematical functions (such as cos, cos, exp, and log),

* Boolean operations (such as AND, OR, NOT),

* conditional operators (such as IF-THEN-ELSE),

* functions causing iteration (such as DO-UNTIL),

* function causing recursion, and

* any other domain-specific functions that may be defined.

We are interested in the last category of functions because the functions applicable to this paper are uniquely defined so as to give desirable results - generating interesting forms. Bearing in mind that genetic programming was developed to solve a wide range of problems, it is probably more widespread and easier to cast problems of a mathematical nature as a computer program. However, this important aspect of the versatility of genetic programming has been harnessed to bridge the gap between creativity in design on the one hand and the mathematical nature of geometrical forms on the other.

Consider the function set

F = { UNION, INTERSECT, SUBTRACT }

and the terminal set

T = { S0, S1}

where S0 and S1 are Boolean variable sets that serve as arguments for the functions.

A combined set of functions and terminals is as follows:

C = F U T = { UNION, INTERSECT, SUBTRACT, S0, S1 }

The terminals in the combined set C can be considered as functions that does not require any arguments.

Take for example the INTERSECT function with two arguments. A set that occurs both in the two arguments is returned if there is an overlap in arguments sets (i.e., S0 and S1) and a NIL if there is none. A typical Boolean function can be expressed in disjunctive normal form (DNF) by the following LISP S-expression;

(UNION (INTERSECT (SUBTRACT S0 S1) (SUBTRACT S0 S1)) (INTERSECT S0 S1))

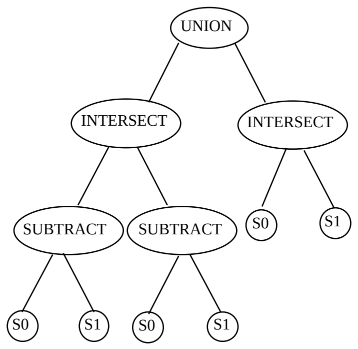


Fig. 1: S-expression depicted as a rooted, point-labeled tree with ordered branches

The above LISP S-expressions can be graphically shown as a rooted, point labelled tree with branches. The leaves (external points) of the trees are labelled with terminals S0, S1, S0, S1, S0 and S1 respectively. The internal points are labelled with functions UNION, INTERSECT, SUBTRACT, SUBTRACT and INTERSECT. The root of the tree (the UNION) is the first function just inside the outermost left parenthesis of the LISP S-expression. This tree is equivalent to the parse tree which most compilers construct internally to represent a given computer program.

All possible trees of this description that can be generated from any possible recursive combinations of the available functions and terminals appropriate to the problem at hand is the search space to be explored by genetic programming. This is equivalent to searching through all LISP S-expressions consisting of the available functions and terminals.

This is the distinction between the conventional genetic algorithm and genetic programming. While one-dimensional strings, be they finite or variable length, are the structures that undergo adaptation in genetic algorithms, the structures that undergo adaptation in genetic programming are hierarchical structures (rooted point labelled trees with ordered branches).

Closure of the Function set and Terminal Set

Due to the hierarchical nature of the trees, one has to take care that the functions in the function set should be well defined and closed such that each function is able to accept any value or data type that may be returned by any function in the function set and any value and data type that may possibly be assumed by any terminals in the terminal set. This is especially relevant where ordinary computer programs contain arithmetic operations, conditional comparative operators, and conditional branching operators. Computation breaks down when the operators are given an undefined variable (eg. division by zero, logarithm of zero) or unacceptable data (e.g. NIL, square-root of a negative number).

Closure of the function set in such cases can be dealt with by defining a protected function that does not evaluate division by zero or returning an absolute value when a nonpositive argument is encountered or avoiding non-numerical logic operators.

Sufficiency of the Function Set and Terminal Set

In order to use genetic programming effectively, the set of terminals and the set of primitive functions selected should be capable of expressing a solution to the problem. It is up to the designer to identify the set of function and terminals that has sufficient explanatory power for the problem at hand. Depending on the problem, this may be obvious or may require considerable insight. No doubt, design knowledge and experience comes into play where a good choice is concerned. This step of identifying the right variable to solve a particular problem is common to virtually every problem in science. In the sphere of design, personal sets of strategies which designers adapt to particular design circumstances can be expressed in the set of functions and terminals. It is not unknown that many architects already have strategies that are often pronounced and consistent to the point where their individual projects are instantly recognisable.

This project also attempts to show that the designer has complete control over how he/she would like to express his/her design concepts. Designers are free to define their own functions set and terminal set. The results of the experiments that I have performed are solely based on the operations implied in the functions that I had written and the operations are performed on the terminals which I have selected from the domain of all possible terminals. Terminals can be any 3 dimensional shape. They can be simple primitive geometrical objects (eg. cube, wedge, etc.) or a composite of them or even any 3 dimensional form that can be mathematically generated by the computer.

The Initial Structur es

The initial structures are made up of individual S-expressions that form the initial population in the first instant. This first population is completely generated at random. For each individual S-expression, the root of the tree is labelled by randomly selecting one function from the set of functions F . Selection of the root function is confined to functions in the function set. The purpose is to generate a program tree with branches extending profusely to form a hierarchical structure rather than creating a depraved tree with a single terminal.

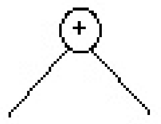


Fig. 2 Beginning of the creation of a random program tree, with the function + with two arguments chosen for the root of the tree

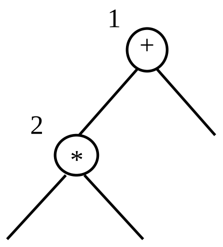


*Fig. 3: Continuation of the creation of a random program tree with the function * with two arguments chosen for point 2.*

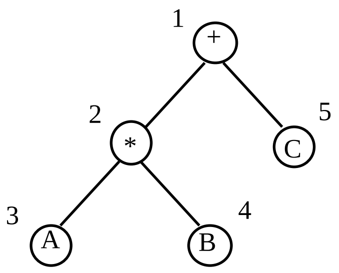


Fig. 4: Completion of the creation of a random program tree, with terminals A, B & C chosen

Fig 2 shows the first node of the random program tree. As the root of the tree, the function chosen (+) has two arguments. Two lines where two is the number of arguments of the function radiate out from the node. For each function, F with z(f) arguments, there are z(f) number of radiating lines. Then, for each such radiating line, an element from the combined set C = F U T of the functions and terminals is randomly selected to be the label for the endpoint of that radiating line. In the case where a function happens to be selected for the endpoint of a radiating line, then the process is repeated as described above. A recursive generative process continues if functions are repeatedly selected In Fig 2, the function * was selected for the second point (internal node) from the combined set C = F U T. The function * has two arguments which is represented by two lines radiating from node 2. Suppose then, that a terminal, A was chosen to be the label of an endpoint, for example at node 3. At this point, the generating process will terminate. Similarly, if terminals B and C were selected to be the labels of the other two radiating lines, the generating process is complete and a fully labelled tree is created, as shown in Fig 4.

Although the example shown has a tree of depth of 3, due to the random nature of selecting functions and terminals, resulting trees may be of variable shapes and sizes. The depth of the tree is defined as the length of the longest non backtracking path from the root to an endpoint. There are several ways to achieve some control over the size of a tree.

One method of generating the initial random population is to define a fixed maximum depth. The length of every non backtracking path between the root and endpoint is restricted to a specified maximum length. This requires that for points at less than the maximum depth, the random selection of labels is taken from the combined set C = F U T. However, once the depth has reached the maximum specified, selection of labels is restricted to the terminal set T. The number of functions in the function set F and the number of terminals in the terminal set T influence the expected size of the tree somewhat. This method of generating initial trees has been termed as the "grow" method by Koza(1994).

Other methods of generating initial trees are the "full" method and "ramped half-and-half" method described by Koza (1994). In the "full" method, selection of labels for all internal nodes at less than specified maximum depth are taken from the function set and only from the terminal set T at maximum depth. The resulting trees tend to have the same size. The "ramped half-and-half" method is a combination of the "grow" and "full" methods. For example if the maximum specified depth is 6, 20% of the trees will have depth 2, 20% of the trees will have depth 3 and so forth up to depth 6. Then for each value of each depth, 50% of the trees are created via the full method and 50% of the trees are produced via the grow method. The "ramped half-and-half" method produces a wide variety of trees of various shapes and sizes, but for simplicity sake, the "grow" method was chosen for this experiment.

For reasons of creating genetic diversity and to avoid wasting computational resources, I have taken steps to eliminate duplicate trees in the initial population of generated trees, although this is not necessary. Each newly created S-expression was subject to a check for uniqueness before it is inserted into the initial population, otherwise it is discarded and the process is repeated until a unique S-expression has been created. Variety of the population is maintained at 100% for the initial population. This may be not be the case in later generations but should be expected as an inherent part of the genetic processes.

Fitness

The rule of Darwininan natural selection pivots about a measure of fitness. In nature, fitness is the measure of a living thing to survive an propagate itself . This is adapted to the sphere of computer algorithms in genetic algorithms and genetic programming, where it takes on the form of some control over the process of when and how artificial reproduction is determined. In other words, the individuals in a population is evaluated by some procedure and rank in order of performance according to the procedure. This procedure can be made explicit, as in many applications of a mathematical nature, or it may be implicit. In my case, the fitness can be called the

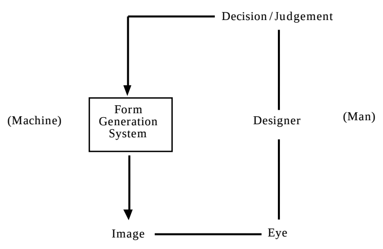


Fig. 5 : Interaction between man and machine, the 'see and decide' relationship

‘eye test’. Individuals in a population will be scrutinised for its aesthetic appeal and the best looking one(s) selected to live on. The user is given the opportunity to make subjective judgements before moving on. The selected individual(s) are the chosen parents of the next generation.

Fitness can also, in this context, be understood as a steering mechanism. The act of choosing parents with desirable characteristics is to lead to a preferred direction in the evolutionary path. Since we do not have a million years to spare, an evolutionary process is accelerated by purely manoeuvring through the shortest path to arrive at a possible solution. Anything else not in favour is omitted.

Reproduction.

The individuals in each generation of the genetic programming undergo adaptation via the choice of three operations

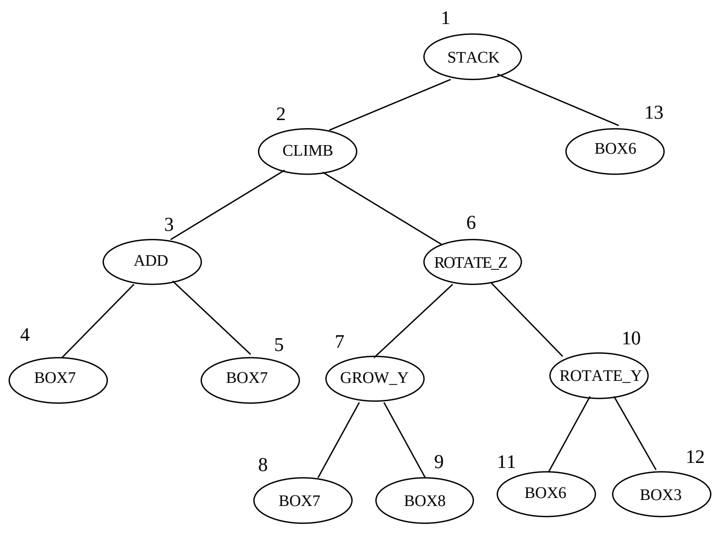
- crossover (sexual recombination)
- asexual reproduction
- mutation

CROSSOVER

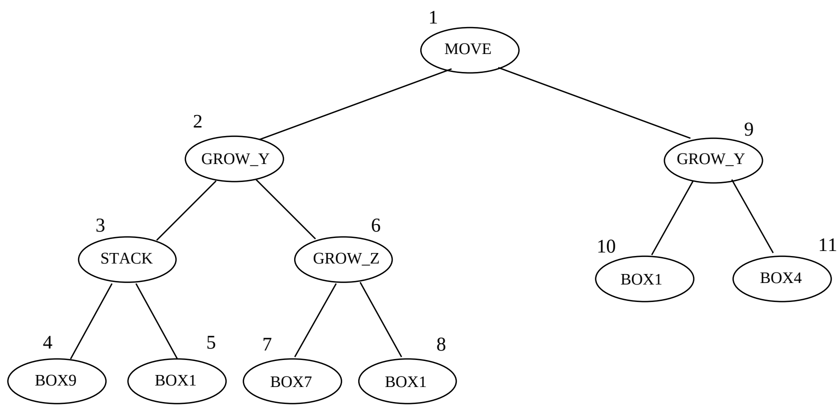
New offspring are produced from the parent(s) that has been selected from the population according to the ‘eye test’. In the case where crossover (sexual recombination) is chosen to be the breeding process, two parents are selected. The crossover operations begins by randomly selecting a point or node in each parent. The nodes serve as crossover points for the parents. Offspring are produced like this: The fragment of one tree lying below the crossover point in one tree will be exchanged for the sub-tree fragment of the second tree. Offspring 1 keeps everything else above the crossover point of the parent 1 but will have the sub-tree fragments of parent 2 attached to it at the crossover point. Similarly for offspring 2, parts of parent 2 is combined with the subtree fragment of parent 1. From the point of LISP S-expressions, crossover swaps the sublists starting at the crossover point. This will always produces legal LISP S-expressions as offspring irrespective of parents or crossover points.

There are several cases of crossover worth discussing here due to the random nature of selecting crossover points.

If the crossover point of a parental tree is chosen to be the root, the crossover operation produces an offspring 1 by replacing the entire tree of the first parent with the sub-tree fragment of the second parent while offspring 2 will include parent 2 and the whole tree of parent 1 at the point of crossover point. In the case where the roots of



(STACK (CLIMB (ADD (BOX7 BOX7) (ROTATE_Z (GROW_Y BOX7 BOX8) (ROTATE_Y BOX6 BOX3))) BOX6)



(MOVE (GROW_Y (STACK BOX9 BOX1) (GROW_Z BOX7 BOX1)(GROW_Y BOX1 BOX4))

Fig.s 6 & 7: Two parental computer programs. (S-expression shown below each parse tree)

both parents are chosen to be crossover points, the resulting offspring are repeats of their parents.

Crossover produces trees of considerable variety. Shapes and sizes of trees tend to be influenced by whether a function or a terminal occupies the node chosen as crossover point. In the event where the node of parent 1 selected is a terminal, then the terminal will be be inserted at the location of the subtree of the second parent while this subtree

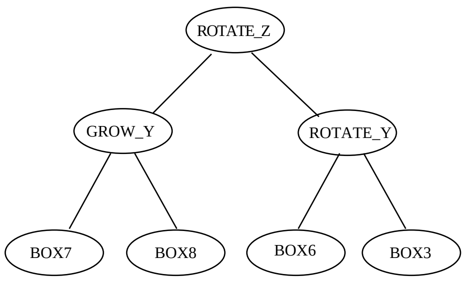


Fig. 8: The crossover fragments resulting from the selec - tion of crossover point of point 6 of the first parent .

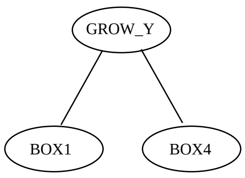
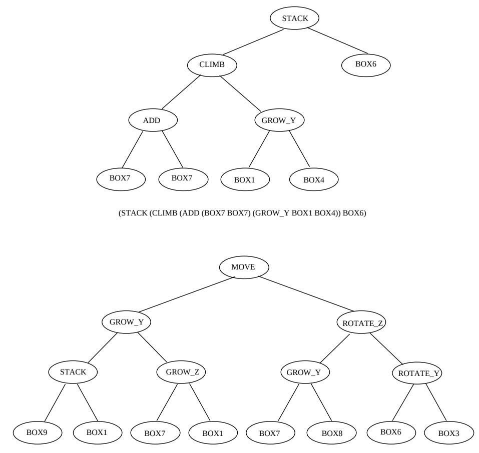


Fig. 9 : The crossover fragments resulting from the selection of crossover point of point 9 of the sec -



```
(STACK (CLIMB (ADD (BOX7 BOX7) (GROW_Y BOX1 BOX4)) BOX6)
```

```
(MOVE (GROW_Y (STACK BOX9 BOX1) (GROW_Z BOX7 BOX1) (ROTATE_Z (GROW_Y BOX7 BOX8) (ROTATE_Y BOX6 BOX3)))
```

Fig. 10: The two offspring produced by the crossover

now occupies the position of the terminal. The first offspring has 'grown'and the second offspring is a shorter tree. If the crossover points are both terminals, the crossover operation has the effect of just swapping terminal from tree to tree.

ASEXUAL RECOMBINATION

This is the situation where one individual mates with itself. Practically, only one parent is chosen and the other is a copy. The operations are the same as for crossover except that now the two parents are identical. The two resulting offspring are likely to be different because the crossover points selected are going to be different anyhow. However, I am not ruling out that the choosing the same crossover points will not happen.

This is not the case with genetic algorithms. What is different between genetic algorithm and genetic programming is that in conventional genetic algorithms, only one crossover point is selected and applied to both parents. Two similar fixed length strings crossing over at the same point cannot produce dissimilar offspring. The implication of this is that premature convergence occurs in conventional genetic algorithms and the population is directed towards over production of similar offspring.

In 'The Selfish Gene'(Richard Dawkins, 1974) writes; " ...Darwinian adaptation precludes that selection cannot produce adaptations unless there is a hereditary difference among which to select." Darwinian selection has to have gene variation to work on.

MUTATION

The third method of reproduction, mutation introduces variation in the gene pool. The aim is to cause random changes in the structures in the population. The use of mutation most useful on two aspects in this project. The first is to do a quick random walk (which in principle is what mutation alone achieves) to search for a suitable form (or forms) and thereafter to perform crossover or asexual reproduction on the selected forms to hone in on desirable characteristics and features. Secondly, mutation brings about leaps from hill to hill. Occasionally, this might be beneficial when the need to add diversity to the population arises.

The operation of mutation is asexual and only one parental S-expression is involved. One offspring S-expression is produced in a mutation. Once again, a node is selected at random. The subtree below this point is discarded and a new randomly generated subtree is inserted in its place. A high rate of mutation is said to have taken place if the node is chosen is higher up the tree nearer the root and a low degree if chosen node is near the bottom of the tree. Obviously an offspring produced at a lower rate of mutation has near all the characteristics of the parent and more besides. But the degree of similarity depends very much on the new subtree. It may be possible that an offspring with a low rate of mutation resembles its parents in very few places, especially if the subtree is of huge size or contain features that overshadows the original.

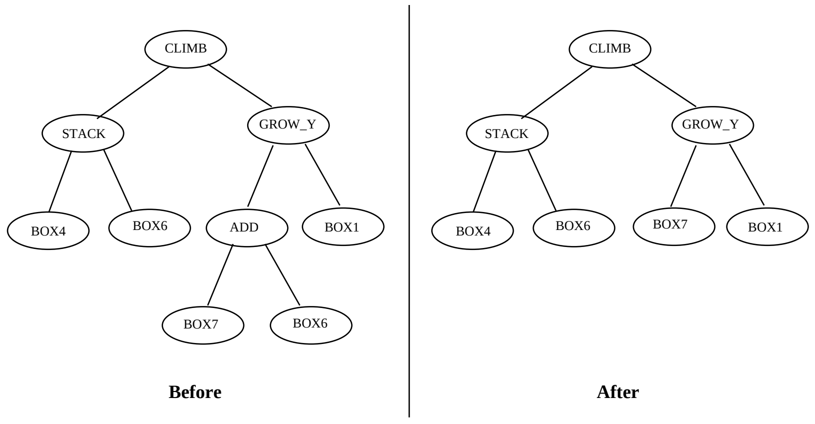


Fig. 11: A computer program before and after the mutation operation

The mutation operation in genetic programming is quite different from that of genetic algorithms. In conventional genetic algorithms, mutation helps to prevent against premature loss of potentially useful genetic material. 1's and 0's at particular locations may be lost occasionally. Mutation is the random alteration of the value of a string position, i.e. flipping a 1 to a 0 or vice versa. Mutation is generally considered a minor operation in genetic algorithms.

RESULT DESIGNATION AND THE TERMINATION

The genetic program paradigm adopted in this project aims to imitate nature and that foreordains a never-ending process. In principle, the evolutionary process continues on and on. Practically, it is usually terminated by either one of the following whichever comes first; a desirable structure(s) has been discovered and designated as the result of the experiment, or the processing power of the computer does not permit any further processing.

Using Genetic Pr ogramming

There are five major steps in preparing to use the genetic programming paradigm to solve a problem:

• determining the set of terminals

• determining the set of functions

• determining the fitness measure

• determining the parameters and variables for controlling the run, and

• determining the method of designating the result and the criterion for terminating a run.

Note that for each problem, solutions can be found on numerous runs. However, since the genetic programming paradigm is a probabilistic method, different runs almost never yield precisely the same S-expression. No one particular run and no one particular result is typical or representative of all the others.

The terminals can be viewed as the input to the computer program being sought by genetic programming. In turn, the output of the computer program consists of the value(s) returned by the program.

FORM GENERATION AND SEARCH

In my problem of form search, the information we want to process is the transformations that can be applied to solids and shapes to generate interesting forms. Thus the functions set for the problem consists of three dimensional transformation operators. They can be pure affine transformations or composites of them. The terminal set for this problem should then contain the objects or shapes on which the transformations are to be acted upon. Thus, the terminal set selected is :

T = { box, cone, cylinder, sphere, torus, wedge }

primitives of solid shapes available in AutoCad.

The function set F selected is:

F = { (grow_x), (grow_y), (grow_z), (move), (stack), (rotate_x), (rotate_y) (rotate_z), (climb), (add), (subtract), (intersect) }

Each of these two functions has an two arguments. For reasons of computer processing power, the function arguments have been limited to two. Theoretically, it can be any number of arguments. However, the resulting LISP tree would be too complicated for more than 3 arguments.

The function (grow_x) grows the second argument by a random factor and attaches itself to the first argument along the x-axis. (grow_y) and (grow_z) is similar to (grow_x) except that the attachments are along the y and z- axis receptively. Functions (move), (stack) and (climb) translates the second argument along the x, y, and z-axis by the extents of the first. Functions (rotate_x) (rotate_y), (rotate_z) rotates the second argument with respect to the first. Functions (add), (intersect), (subtract) are standard Boolean operations performed on both arguments simultaneously.

The third major step in preparing to use genetic programming is to identify the fitness measure. This may be considered as the test mechanism at the local level, i.e., at the solution generating level. On a global level, the form will be subject to more formal analysis often associated with the function of the structure.

Visual feedback and interaction are an important part of this experiment. The key to whether an individual form survives depends on its appeal to the eye. The fitness can be known as aesthetic fitness. No number crunching is involved in the fitness measure. The best-of-run individual(s) in the population is the individual(s) chosen as the parent(s) of the next generation.

Breeding can take place using 3 methods, by crossover reproduction, by asexual reproduction, or by mutation. In crossover reproduction, two individual of the population are selected from whom 16 other offspring are bred. One parent is chosen for both asexual reproduction and mutation. The parents are not copied to the next generation. Every new generation consists of 16 new re-combinations of the chosen parent(s). This cuts down on having to re-evaluate the same parent(s) twice and since I am interested in exploring a wider expanse of search space, no effort should be spared to produce as many possible forms as possible in each generation. 16 forms per generation has been tested to be generally suitable.

At every generation, the 16 S-expressions for each individual is evaluated and its form presented to be tested for aesthetic fitness.

Any number of generations can be tested, there is no limit to the number of generations. Once a form has been identified as the final one with no further transformations required, the run can be terminated. The resultant satisfactory structure would be deemed as having emerged from the search of possible forms defined by the functions and terminals.

As the populations are bred from generation to generation, a history of the evolution is created. This is duly recorded and they form a gene bank. Each structure is a potential choice as a new form to be adapted. An interesting forms from the gene bank can be added to the set of terminals. The starting point of the evolution process using a terminal set consisting of forms taken from the gene bank has a head start many generations from the primitive shapes from which they were made up.

Results of the experiments illustrated in Appendix A. based on representation set, R1 where the set of objects is the set of nine possible pattern of wedges, terminal set T =

{wedge1 wedge2 wedge3 .... wedge9} and the operators, function set F = {grow_x grow_y grow_z rotate_x rotate_y rotate_z climb move stack add intersect subtract}.

Thus by varying the sets F and T, a different search space is created every time. It is up to the user to discover what to define in representation set in order to create his/her modelling world.

A constant seed was chosen while the reproduction methods were varied. Two different evolutionary paths taken for each type of reproduction demonstrates the power of selection in steering the evolution of form. Each different type of reproduction chosen produces it own family of structures.

Asexual and crossover reproduction breed trees that bears resemblance to each other only to the extent that no extra information is added to the first population that is initially generated. Outcomes are a recombination of information amongst the individuals of each population. For each generation, information is limited to that contained in one tree selected for asexual reproduction while crossover reproduction has twice the number.

Clearly by changing the seed, the consequence is a new set of initial representations, I. Starting from a different point in the search space and choosing different paths allows one to visit virtually every state in the search space. Due to the randomness of traversing paths one cannot rule out the chance of a fortuitous occurrence of the same structure and should therefore not be piqued by it. In most cases, similar looking structures can have entirely different S-expressions (hierarchical trees), one may be of a depth of 8 and the other a considerable depth of 17. It goes to prove that there are more ways than one to achieve a ‘solution’, but the paradigm does not take into account the ‘economical efficiency’of arriving at it. This is also found in nature. Up till today, no one has been able to completely explain the extraneous human genetic material that apparently cannot be accrued to any feature . (Dawkins)

As the populations are bred from generation to generation, a history of the evolution is created. This is duly recorded and they form a gene bank. Each structure is a potential choice as a new form to be adapted. An interesting forms from the gene bank can be added to the set of terminals. The starting point of the evolution process using a terminal set consisting of forms taken from the gene bank has a head start many generations from the primitive shapes from which they were made up.

Fig. 12 traces the genealogy of an emergent form. .History of evolution of the emerging structures can be referred to in Appendix A.

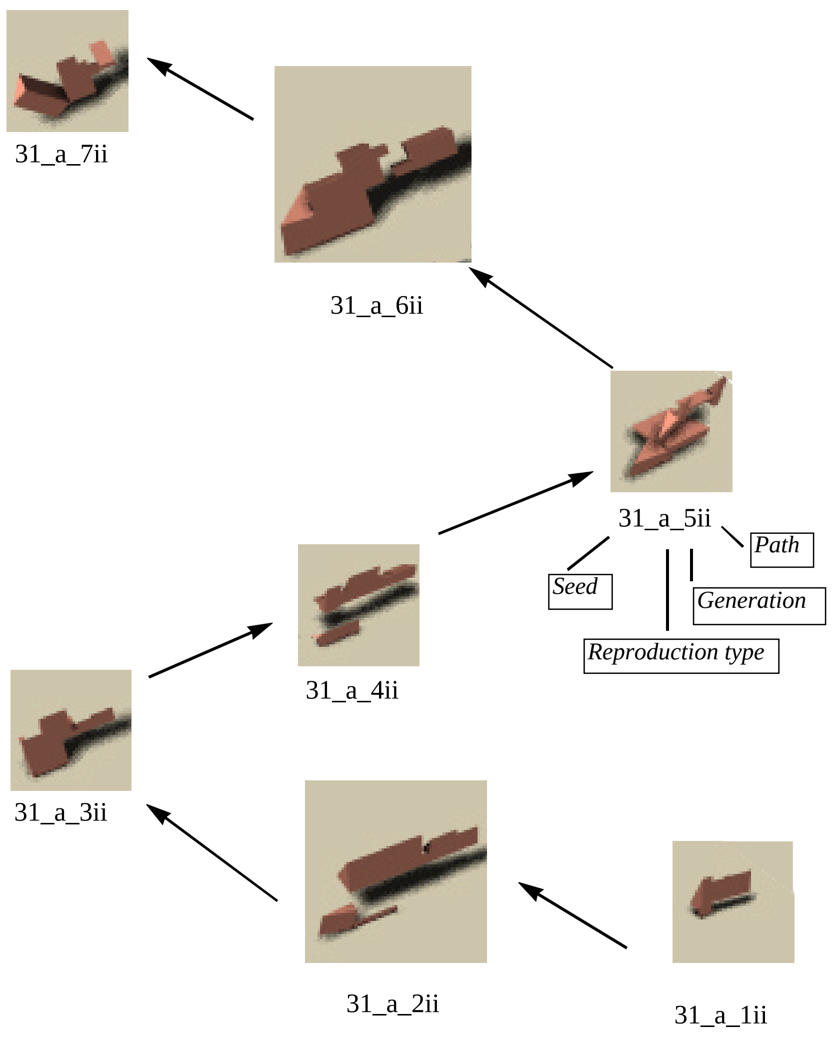


Fig. 12 : Evolving tree of forms, parent forms only

Conclusion

A device to execute design as search necessitates three subclasses of jobs. The first, the representation of the problem through structuring and restructuring the problem space, also known as the design representation. The control strategy (design concepts) that the architect chooses to produce a final solution has to be translated into the description of a three-dimensional space. The second, a solution generation mechanism applying the evolutionary concepts of Darwinian principle of natural selection and reproduction is set up. Structure is defined by its 'genetic' code scripted as S-expressions (hierarchical trees). Structure undergoes adaptation via the process of reproduction and crossover of individual structures, the vehicle for creation of new population structure. A test mechanism that evaluates the 'aesthetic fitness'of a structure is put to use. This drives the evolutionary path to generate interesting and appealing structure. Third, the designation of a candidate solution (or solutions).

The search paradigm stands in harmony with, and perhaps partially motivates, the present recurrence of compositional ideas in architecture. At its core, the model views search as the action of a set of operators on a representation all being guided by a search strategy. When compared to a current understanding of human problem solving, the model suggests that a complementary relationship exists between human designers and computer-based search systems.

The search paradigm owes intellectual debts to many sources outside architecture. These include formal language theory and computational linguistics for grammatical concepts, set theory for representations and proof procedures, cognitive psychology for models of human problem solving, and artificial intelligence for representation and search. However, the paradigm is more than an amalgam of these gleanings, it is enlivened by its own logic and firmly anchored in the field of design.

Design exploration through search will improve the ability to create in fundamental ways, the generation of visual alternatives in a short period of time allows for making selections and new associations possible. A fast feedback of results and the immediate visual control enhances imagination. For intelligent users, the proposed model provides more opportunities for creativity.

Design intelligence located in the generation mechanism assumes that an acceptable solution based on the control strategy will be quickly produced; the evaluation has nothing to do. This simulates the existence of God, a smart designer with no need for a critic. Designer intelligence located in the test mechanism indiscriminately produces alternatives and it is up to the evaluation mechanism to sort out the acceptable ones by bringing a knowledge to bear, drawing inferences and exploring entailments of alternatives. This is evolution; indiscrimate generation but deadly effective criticism.

2) Using a GA to evolve rule sets to achieve a goal configuration

PRIOR WORK

Prior work involving the use of the essential techniques for form generation and analysis in the experiments - L-systems, genetic algorithms and the Hausdorff distance equation.

Various computer-generated models of morphogenesis have been used to help understand the emergence of complex forms in living organisms since Turing proposed the reaction-diffusion process in 1952. Diffusion limited aggregation models have been used to simulate crystal formation in super-saturated solutions and J. Kaandorp in "Fractal Modeling:Growth and form in Biology uses an aggregation model to investigate the growth of corals. L-systems were developed by Astrid Lindenmayer in 1968 to model the morphology of organisms using string re-writing techniques. These techniques have been applied in a variety of studies to the production of abstract models of biological forms as an aid to interspecial comparison and classification see (deBoer, Fracchia and Prusinkiewicz 1992) . An L-system form generation engine is the main ingredient in the following experiments.

L-SYSTEMS

An L-system model starts with an initial axiom and one or more production rules. Axioms and production rules consist of symbol strings whereby individual symbols in the axiom are replaced by a string of symbols designated by the production rule. This process of character recognition and string substitution is carried out iteratively with each successive iteration producing a symbol string of greater complexity.

The resultant symbol string is interpreted as a series of drawing instructions which produce an abstract representation of the desired organism. The success of the L-system model lies in the self similarity of cell structure that many biological systems exhibit which is mirrored in the grammar based string re-writing process.

The L-system method of modeling developmental processes has been the subject of considerable research. Our approach is to use the L-system biological model in conjunction with an evolutionary algorithm and is an area which has seen relatively little investigation, recent work is reported in (Jacob 96)

A standard Genetic algorithm and the associated Genetic Programming strategy are models which utilise the processes involved in the evolution of biological organisms

The Hausdorff distance measures how close each point comprising a given shape is to a point in a second shape and vice versa and is used photographic image recognition .

Visualisation of the growth model was carried out in Autocad, a 3-D modelling appli-

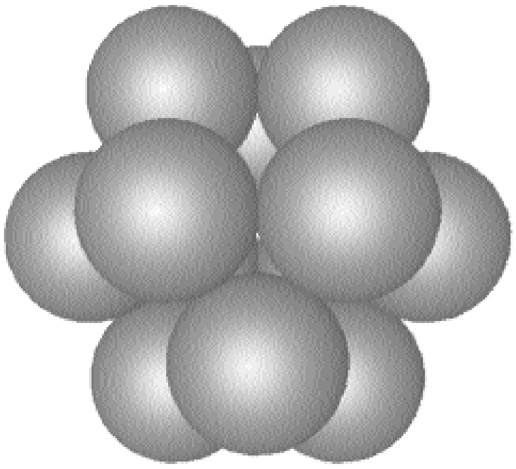


Fig 13 : the 12 spheres of the Iso-spatial dodecahedral array

cation using the Autolisp programming language.

Due to the constraints imposed by using Autocad we kept to the tri-axial Cartesian co-ordinate system rather than the alternative , more ‘elegant’four axial system preferred by Frazer. The twelve neighbours of any given point-’*apoint*’- are given by the autolisp function ‘*getneighbours*’:

```
(defun getneighbours ( apoint / neighbours offsets p np )
   (setq neighbours ‘()
                    offsets(list ‘(1 1 0)
                                  ‘( 1 -1 0)
                                  ‘( -1 1 0)
                                  ‘(-1 -1 0)
                                  ‘(1 0 1)
                                  ‘(0 1 1)
                                  ‘(-1 0 1)
                                  ‘(0 -1 1)
                                  ‘(1 0 -1)
                                  ‘(0 1 -1)
                                  ‘(-1 0 -1)
                                  ‘(0 -1 -1)))
      (foreach p offsets
      (setq np (mapcar ‘+ a point p)
            neighbours(cons np neighbours))
      )
```

Fig 14: the offset list for con - structing the 12 neighbours of a point

The method of representing form will be through the insertion of spheres drawn and saved in separate drawing files. Different coloured spheres represent different levels of recursive branching. The geometry of the 3-D space into which the spheres are inserted is an iso-spatial grid where the co-ordinates of the grid are the vertices of close packed cuboctahedrons as used by J. Frazer in "Data Structures for Rule-based and Genetic Design" . Spheres can only be inserted at these vertices and each sphere has 12 equally spaced neighbours. A variation on this geometry is the one used by Carter Bays' in "Patterns for Simple Cellular Automata in a Universe of Dense-Packed Spheres" which has a four axial system where 12 neighbouring points are the vertices of a dodecahedron.

There are three primary genetic operators used in this experiment, initialisation, crossover and mutation. The first stage of the program is initialisation where a function is called to produce an initial gene pool. The genes are a randomly generated series of nested brackets containing three types of symbols;

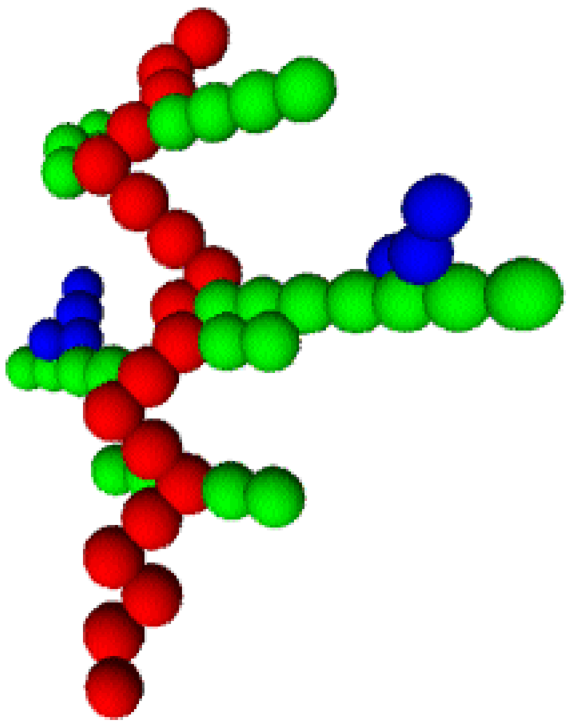


fig 15 : example of a 3D branching morphology using the Iso-spatial array & L system

INITIALISATION

°An instruction to insert a sphere;
this is represented by the symbol **F** (for “forward”)
°A positional variable which refers to which of the 12 neighbouring positions the next sphere inserted will occupy;
these are given in the symbology as **POS1** .. **POS12** for each of the 12 possible orientations from any sphere
°A bracket
an open one **(** indicates a branching point,
a closed one **)** is an instruction to return to a position on a lower 'limb' where it last branched.

These symbol strings which make up the gene pool are the production rules of the L-system.

PRODUCTION RULE

The next stage of the program is to iteratively apply a production rule to the initial axiom - in this experiment a simple 'f symbol. It only requires 3 or 4 iterations to generate a symbol string of considerable length as every instance of 'f is substituted by the production rule which itself is made up largely of 'f's.

```
F  ->

     (F(F POS12(F F POS10)F)F) ->

           (F(F POS12(F F POS10)F)F)((F(F POS12(F F POS10)F)F) POS12
           (F(F POS12(F F POS10)F)F)(F(F POS12(F F POS10)F)F)POS10)
           (F(F POS12(F F POS10)F)F)(F(F POS12(F F POS10)F)F))
```

The re-written symbol string is passed to a function for interpretation of the symbols, ie its genetic code, and the realisation of the artificial organism in the drawing database. Every symbol in the string is evaluated and the corresponding function is called which carries the instruction for either;

(1) changing the positional variable or
(2) inserting a sphere at a point on the grid.

Before a sphere is inserted clash detection is carried out and if a collision is imminent the insertion command is ignored and the next symbol evaluated.

MUTATION

The mutation genetic operator works by counting the levels of recursion or nested brackets in an individual's genetic code and selects, at random, a self-contained balanced chunk of code to be replaced by a similar, randomly generated, chunk. The chunk of code selected always starts with a branching bracket symbol and ends with closed bracket. The symbol string is not of fixed length so mutated organisms can have varying lengths of genetic material.

CROSSOVER

Crossover works by choosing two organisms, selecting suitably balanced sections of code and swapping them. As in mutation the amount of genetic material an individual has can change, producing differences in generations ranging from slight to radical.

OBJECTIVE FUNCTION

After choosing a datastructure and method of representation and defining the type of genetic operators to be used, the third element of the algorithm is defining the objective function or 'fitness' function. This was carried out by designing a series of structured experiments which build on the results of each experiments precursor.
Initial experiments were carried out to test the crossover function. In the figures below, the left hand and centre configurations are two objects generated from a random production rule, the right hand object is generated from the resulting rule after crossover of the other two rule sets.

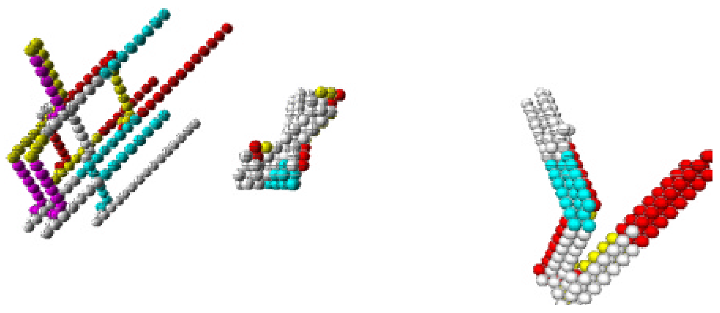


fig 16: Crossover test 1

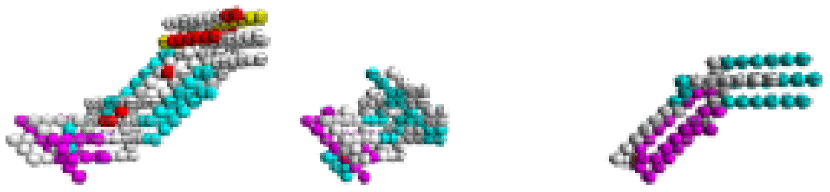


fig 17 Crossover test 2

Our immediate goal was to develop a model which would respond to the user's selection of the characteristics of two individuals which would survive and combine and become accentuated over successive generations - the 'eyeball' test as used in Dawkins' Biomorph program. Figs 18 -21 show screen shots of two experiments.

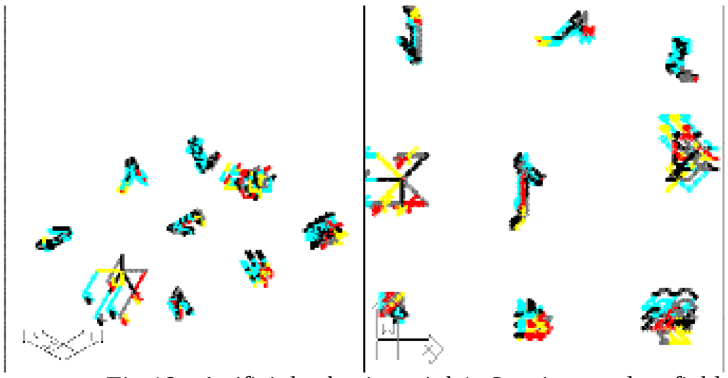


Fig 18 : Artificial selection trial 1- Starting random field of 9 objects

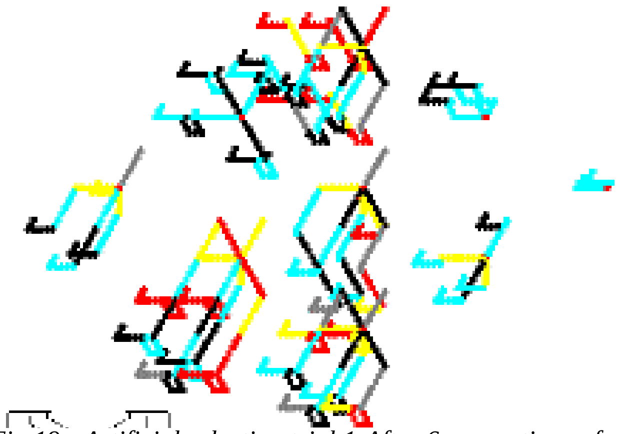


Fig 19 : Artificial selection trial 1-After 6 generations of artificially selecting for "spi - deryness" the field consists of predominantly spidery objects

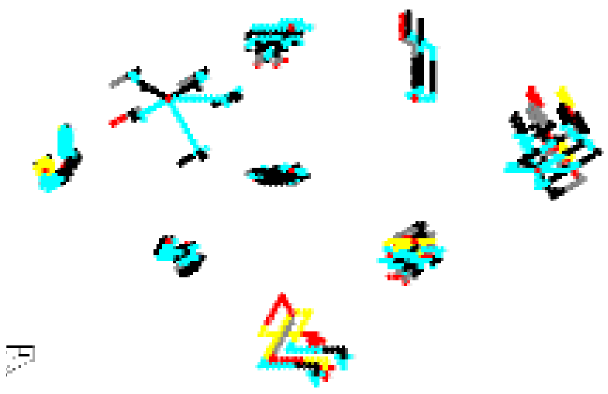


Fig 20 Artificial selection trial 2, the initial field of 9 random genotypes

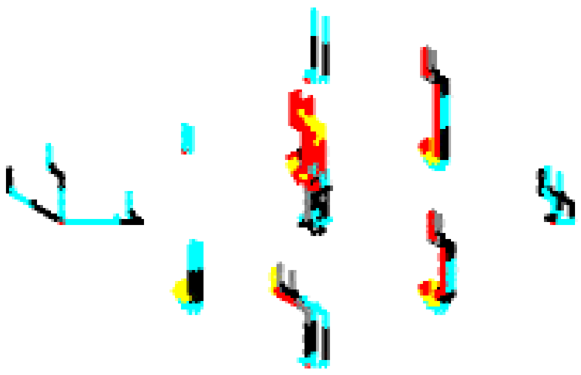


Fig 21 Artificial selection trial 2, after selecting for long thin genotypes

This was completed successfully. The mechanism used here is similar to that employed in section one of this paper, except that whereas there a range of breeding options were allowed (mutation, crossover and asexual reproduction) here the operation is restricted to crossover. At each generation 9 phenotypes are displayed and the user chooses two parents from which to breed the next generation. These two candidates' production rules (genetic material) are them crossed over, 9 times to ,create the 9 genotypes for the next generation, which are then developed into the 9 phenotypes for display.

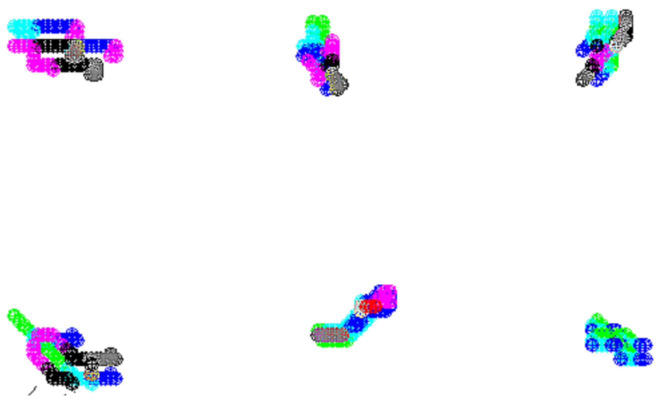


Fig 22 : starting phenotypes for the aspect ratio natural selection experiment

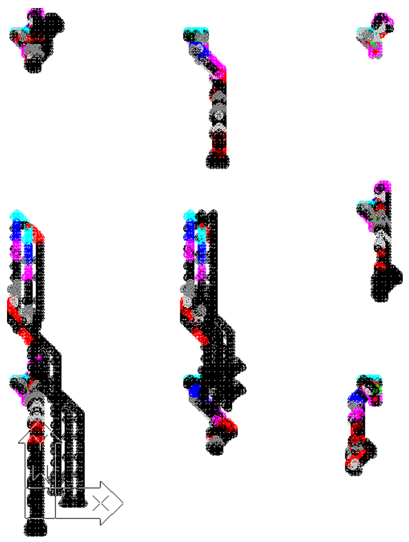


Fig 23 : result of setting a 1:10 X-Y aspect ratio after 20 generations

The next stage was to define an ideal form in terms of a desirable length over width ratio and run the algorithm until it produced forms matching the target within a given tolerance Figures 22 and 23 illustrate a typical run, showing convergence to the target (ie the initial random morphologies slowly adapt to assume more and more etiolated forms).

More complex forms can now be defined as a target. By using the Hausdorff distance equation we can analyse how close an individual has come to matching the pre-defined shape. Successful individuals are used to breed the next generation. These experiments are still continuing, and will be reported to the conference.

FUTURE WORK

The work reported here is at an early stage, and there are two main areas which will be developed next. Firstly the basic function set of **F** and **P** can be enriched to provide a wider range of behaviours; secondly the problem of adopting arbitrary fitness functions can be addressed by using the "arms race" paradigm where a competitive scenario can be introduced by growing two forms simultaneously where each form is its opponent's environment and the fitness function becomes pure survival.

References

Bays C.(1988): Classification of semi-totalistic cellular automata in 3-D, Complex Systems 2

Bays C (1987).Patterns for Simple Cellular Automata in a Universe of Dense-Packed Spheres, Complex Systems 2

DeBoer,Fracchia,Prusinkiewicz (1992), "Analysis and Simulation of the Development of Cellular Layers" Artificial Life II ed. C Langton

Das, S., Franguiadakis, T., Papka, M., DeFanti, T. A., Sandin, D. J.(1994):Agenetic programming application in virtual reality. IEEE Computational Intelligence Evolutionary Computation Conference Proceedings, June .

D'arcy Wentworth, Thomson. (1917/61). On growth and form. Abridged ed. (Tyler Bonner, John ed.) Cambridge University Press.

Dawkins, Richard (1991). The Blind Watchmaker. London: Penguin.

Dawkins, Richard (1972). The Selfish Gene. OxfordUniversity Press.

Fraser, John (1995). An evolutionary architecture. London: Architectectural Association.

Frazer J Datastructures for rule-based & Genetic Design

Graves, Michael (1977). "The Necessity of Drawing: Tangible Speculation." Architectural Design, 47, no 6, pp. 384 - 394.

Gero, John S. & Schnier, Thorsten. Evolving representations of designcases and their use in creative design.

Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley.

Horling B Implementation of a context-sensitive Lindenmayer-system modeler Dept Engineering and Computer Science Trinity College Hartford USA

D.P.Huttenlocher, GA Klanderman, W J Rucklidge. (1992): "Comparing images using the Hausdorff distance under translation" Computer Vision and Pattern Recognition, pages 654-656 Champaign-Urbana Ilinois,

Holland, John (1975). Adaptation in Natural and Artificial Systems. Cambridge, Massachusetts: MITPress.

Jacob, C.,(1996) "Evolving Evolution Programs:Genetic Programs and L-Systems". Proceedings of first annual conference on genetic programming, Stanford USA MIT Press pp 107-115.

Jo, H. Jo & Gero, John (1995) Representation and use of Design Knowledge in evolutionary design. CAAD Futures '95. Singapore.

Kaandorp J.(1994)Fractal Modeling: Growth and Form in Biology, Springer Verlag,

Langton C. G. (ed.) (1990).Artificial Life II. Proceedings of the workshop on Artificial Life. Santa Fe. Feb Addison-Wesley.

Koza, John R. (1992). Genetic Programming, on the programming of computers by means of natural selection. Cambridge, Massachusetts: MIT Press.

Langton C. G. (Ed)Artificial Life I II & III Addison-Wesley Publishing Company

Lawson, Bryan (1990). How designers think? 2nd ed. Butterworth Architecture.

Lawson, Bryan (1994). Design in mind. Butterworth-Heinemann.

Lindenmayer A. and Prusinkiewicz P.The Algorithmic Beauty of Plants, Springer Verlag, 1988

Lionel March (ed.) (1976). The Architecture of Form. Cambridge: Cambridge University Press.

March, Lionel & Steadman, Philip (1971). The Geometry of Environment. London : RIBA
Mitchell, William J. (1990). The Logic of Architecture, design, computation & cognition. Cambridge, Massachusetts: MIT Press.

Neutra, Richard (1969). Survival Through Design. London : Oxford University Press.

Pearce, Peter (1978). Structure in Nature as a Design Strategy. Cambridge, Massachusetts: MITPress.

Pask, Gordon (1972). An approach to cybernetics. London : Hutchinson & Co Ltd

Rowe, Peter G. (1987). Design Thinking. Cambridge, Massachusetts: MIT Press.

Schimitt, Gerhad (1988). Microcomputer aided design for architects and designers. New York : John Wiley & Sons, Inc.

Schnier, Thorston & Gero, John. “Learning genetic representations as alternative to hand-coded shape grammars.”Artificial Intelligence in Design '96. pp. 35-57.

Schnier, Thorston & Gero, John.(1995) “Learning representations for evolutionary computation.” 8th Australian Joint Conferenc on Artificial Intelligence. AI '95. pp387-394.

Steadman, Philip (1979). The Evolution of Designs. London: Cambridge University Press.

Stiny, G(1980). Introduction to shape and shape grammars. Environmental and Planning B 16, pp 253-287.

Todd, Stephen & Latham, William (1992). Evolutionary Art and Computers. London : Academic Press Ltd.

Tschumi, Bernard (1987). Cinegram Folie Le ParcDe La Villette.Butterworth Architecture.

Walker, Miles (1993). Digital evolutions. Dissertation for MSc Computing and Design, University of East London.

Watt, Alan (1993). 3D Computer graphics. Addison-Wesley. pp 1-13

Appendix A

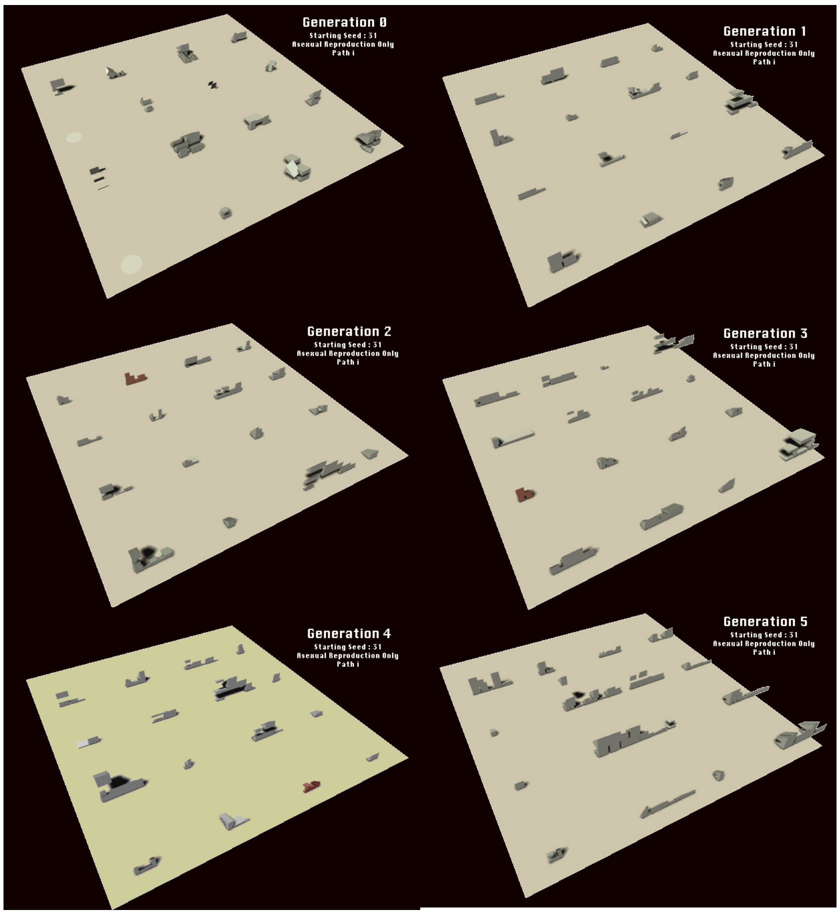


Fig 24 Asexual reproduction only

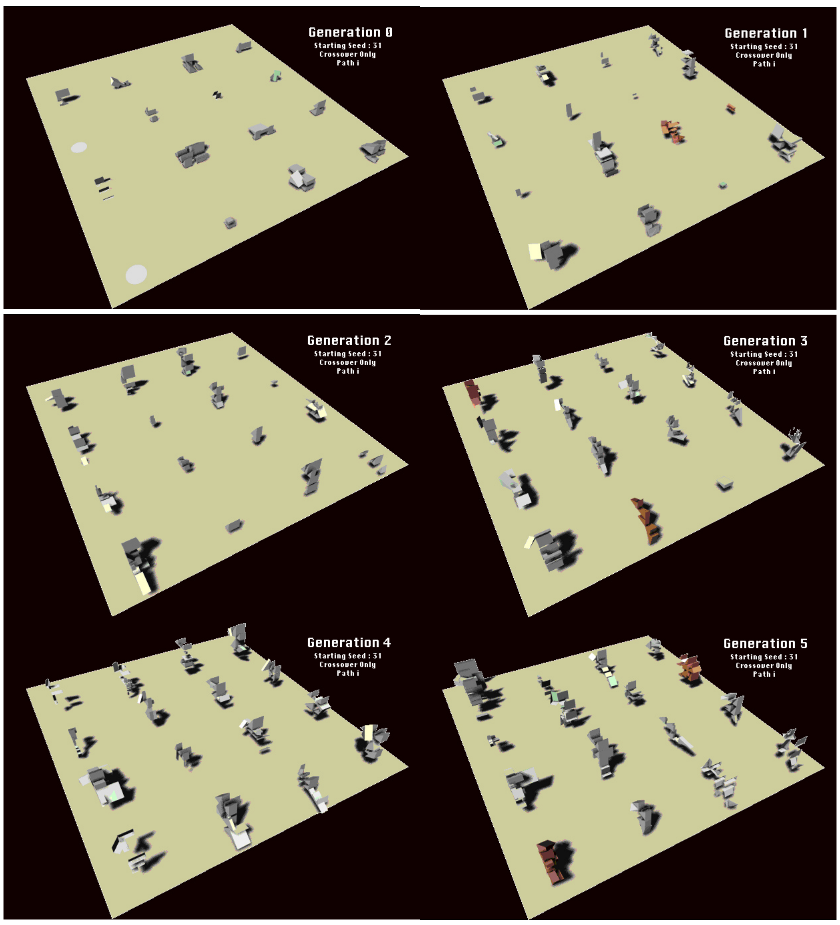


Fig 25Crossover only

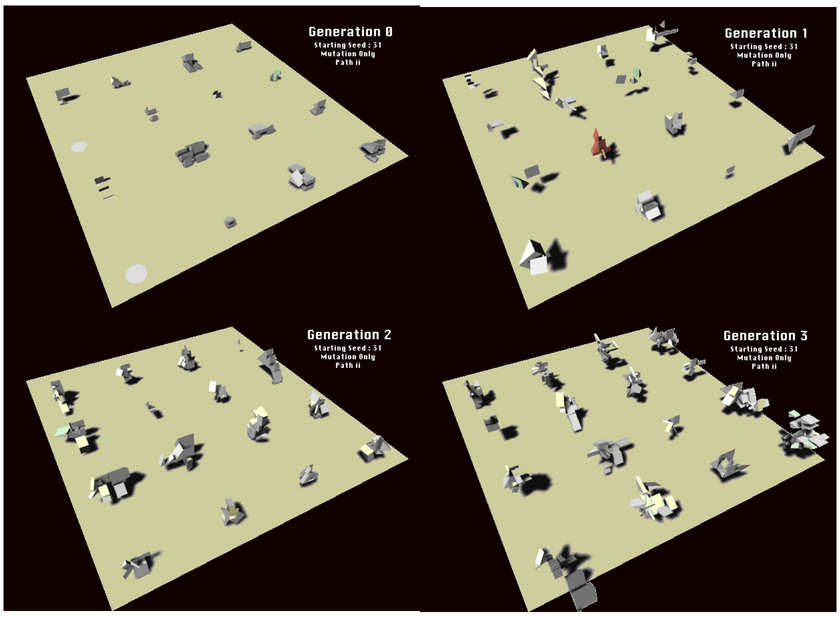


Fig 26 Mutation only

Address for inclusion in directory

P.S.Coates
CECA
University of East London
Holbrook Centre
Stratford
London E15 3EA
UK
Web page:
www.uel.ac.uk/faculties/arch/info.html
Email:
P.S.Coates@uel.ac.uk