A Toolkit for Visual Genetic Programming

Timothy Rowley, University of Minnesota
The Geometry Center, University of Minnesota
email: trowley@geom.umn.edu

August 1994

1 Introduction

Genetic programming is an area of research in computer science directed towards finding "optimal" solutions in a large search space [Gol89]. The method is based on what happens during evolution in the natural world. Natural evolution proceeds because certain members of a species are better fitted for the environment and survive better. With a better survival rate they are more likely to reproduce and pass on positive genes to their offspring. This Darwinistic survival of the fittest improves the gene pool and results in creatures closer to the "optimum" for their environment.

The general idea of genetic algorithms is that there exists a "genome" — the analog of DNA — which can be interpreted as a possible solution to the problem under investigation. It could represent a formula, algorithm, or program. In "traditional" genetic programming, the problem being studied has a fitness function which acts as nature's grim reaper. The fitness function gives a measure of how well the genome performs. The typical genetic programming run consists of the following steps:

1. Generate a gene pool of random genomes
2. Apply the fitness function to all genomes and pick the best to be the parents of the next generation
3. Generate a new gene pool by crossbreeding and mutating the parents from step #2
4. Repeat from step #2

This summer project applied techniques from genetic programming to create a toolkit for exploring problems which don't have a natural fitness function and need human intuition to select the "best of the breed." Examples include images, fractals, music, three dimensional objects, and many others.

2 Tools

To aid in the exploration of these visual genetic programming problems, two general tools were written. The first is a toolkit which provides facilities for displaying the output of the genomes, keeping a history of the problem space investigated, and allowing the user to investigate problems. The toolkit uses a separate module for each problem, allowing an investigator to easily add a new problem for the program to explore. The second piece of software is a general mutator and crossbreeder for mathematical equations, which are considered a valuable genome representation for many types of problems.

2.1 Toolkit

2.1.1 User Interface

A screen snapshot of a typical session with the toolkit is shown in Figure 1. The three main windows can be seen: the selection window, the inspector window, and the history window.

The selection window displays nine images – the current gene pool. The user can go on the next generation in two ways. First, the user can double-click on an image. In this case, the selected genome is moved to the upper left corner for reference and then eight new genomes are created by mutation from the selection and displayed. The second way of getting a new generation is by selecting two parents to be crossbred. When this happens, the two parents are moved to the upper left and seven children are generated by crossover of genes and mutation.

The history window provides the user with a visual history of the problem space they have investigated. It allows the user to go back to a previous choice if they have run into a genetic "deadend". It also gives the user an additional gene pool that can be used when crossbreeding genomes.

The contents of the inspector window vary with the module that is in use. It controls various parameters specific to the selected module. Typically

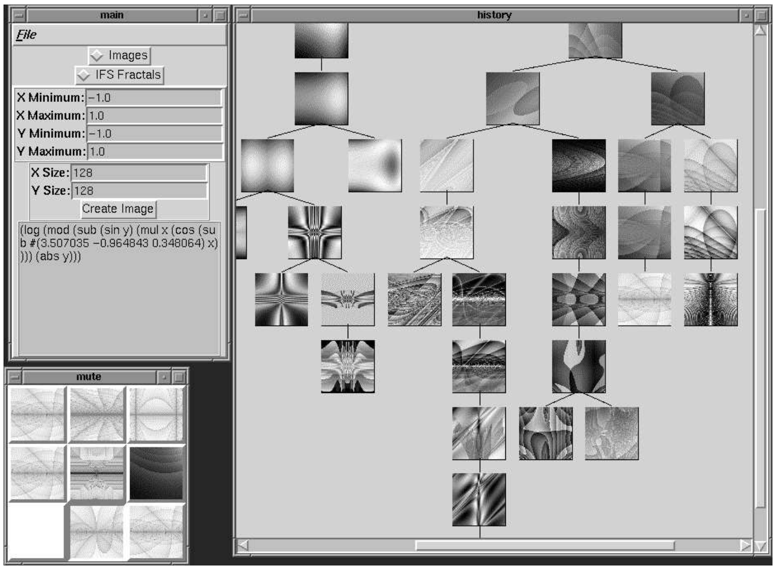

Figure 1: The three main windows of the toolkit: selection window (lower-left), inspector window (upper-left), and history window (right)

it also displays the currently selected genome and has an option to create a larger view of a the selected image.

2.1.2 Module Interface

The various genetic programming problems are treated as plug-in modules to the toolkit. An interface between modules and the program was defined so that adding a new module consists of writing four procedures:

domutate_*module* Takes an genome and performs a mutation

docross_*module* Takes two genome and crossbreeds them

dopicture_*module* Takes an genome and generates an image

doinspect_*module* (Optional) Creates an inspector for the object type image

2.2 Expression Mutator

Expressions are a useful genome representaton for genetic algorithms because they can be used to generate many interesting objects, such as images,

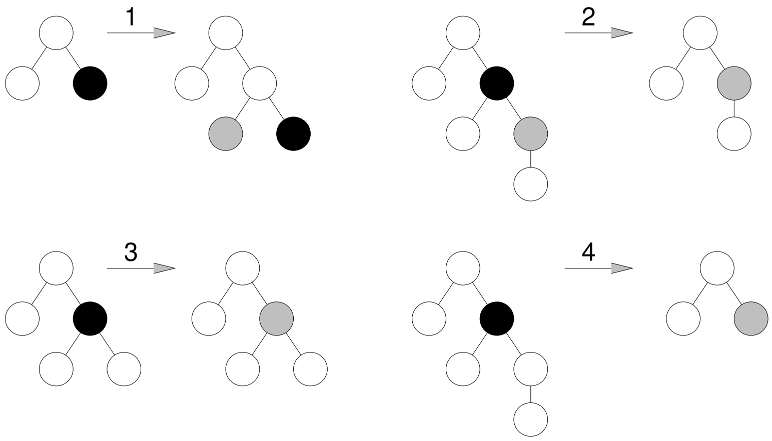

Figure 2: Various types of mutatations performed on trees

sound, parametric surfaces, and implicit objects.

With this in mind, a general mutator and crossbreeder for expressions was written. It takes as input a list of allowable variables and function names, making it adaptable for many purposes.

The expression is represented internally as a tree, with functions as non-leaf nodes and numbers, variables, or vectors stored at leaves. A mutation is done by recursively walking down the tree and deciding at each node whether to mutate it. If the node is a non-leaf, then there are four possibilities for what can happen when the node is mutated (based on weighted probability). See Figure 2 for a pictorial view of the possible mutations.

1. Push the current node down the tree one level, adding a new node to replace the current one

2. Move a node from one level down up to replace the current node

3. Change the function (also changing children as appropriate for the new function)

4. Replace the node with a leaf node

For leaf nodes, the possible mutations are to push the node down the tree (method one from the above list) or to randomly change the value of the leaf.

Two expressions are crossbred by descending the two expression trees simultaneously. At each step, there is a probability that the node of the host expression will be replaced by the corresponding doner expression node.

3 Example Modules

While the tools were written, they were tested by implementing modules to explore images created by expressions and IFS fractals. Audio was briefly experimented with, but set aside for a time when more effort could be invested in it.

3.1 Images

Like the work of Karl Sims [Sim91], images are generated by evaluating a function $f(x, y)$ on the rectangular (x, y) grid. This function is vector valued with red, green, and blue components. The grid of vectors is normalized so all the components are in the range $[0 : 1]$. This function $f(x, y)$ is composed of many standard and custom mathematical functions. The complete list of functions used is: *avg, max, min, add, mul, sub, div, sin, cos, tan, pow, log, exp, abs, inv, rnd, mod.*

The mutation and crossbreeding of the image expressions is performed by the general expression mutator and crossbreeder discussed earlier. The starting equations are fairly easy to understand, but after only a few generations of mutation, they become non-intuitive.

Two images generated from somewhat similar expressions can be animated evolving from one to the other by crossfading the generating expressions [Sim91]. This is done by traveling down the expression tree until differences are discovered. Linear interpolation between the different subexpressions is used when evaluating of expression during the crossfade. An example is shown in Figure 3.

3.2 IFS Fractals

Iterated Function System fractals are generated from a set of affine transformations of the form

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} d \\ e \end{pmatrix}.$$

These transformations have associated probabilities which sum to one. To produce a fractal image, a point is transformed randomly based on the

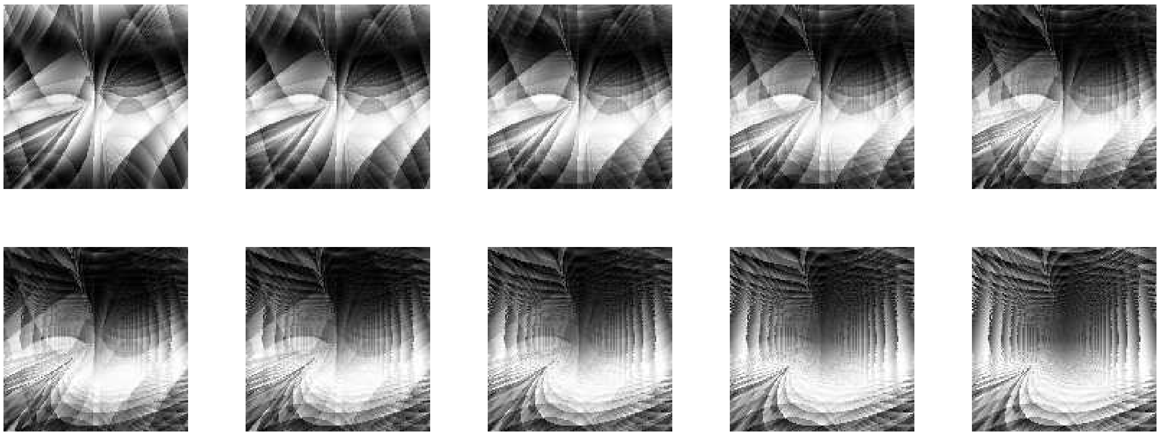

Figure 3: Example of a genetic crossfade between images

probablility. The transformed point is fed back in to be transformed again. Repeating thousands of times give a fractal IFS picture.

These fractals are mutated by randomly adding small amounts to the elements of the transformations. A reasonable method for crossbreeding IFS fractals has yet to be found.

The current implementation of the IFS fractal module keeps the number of tranformations constant, so a genetic crossfade can be performed by linearly interpolating the elements of the transformations over time. An example is shown in Figure 4.

4 Implementation

The expression crossbreeder and mutator was written in Scheme. The image function evaluator was written by extending libscheme, an imbeddable Scheme interpreter [Ben], to add a grid data type and operations to manipulate grids. The mutation toolkit was written in Tk/Tcl [Ous] with the photo [Mac] and tree [Bri] extensions.

The entire toolkit is portable to any UNIX[1] platform running the X Window System[2] with an ANSI C compiler.

[1] UNIX is a trademark of Unix Systems Laboratories, Inc.

[2] X Window System is a trademark of the Massachusetts Institute of Technology

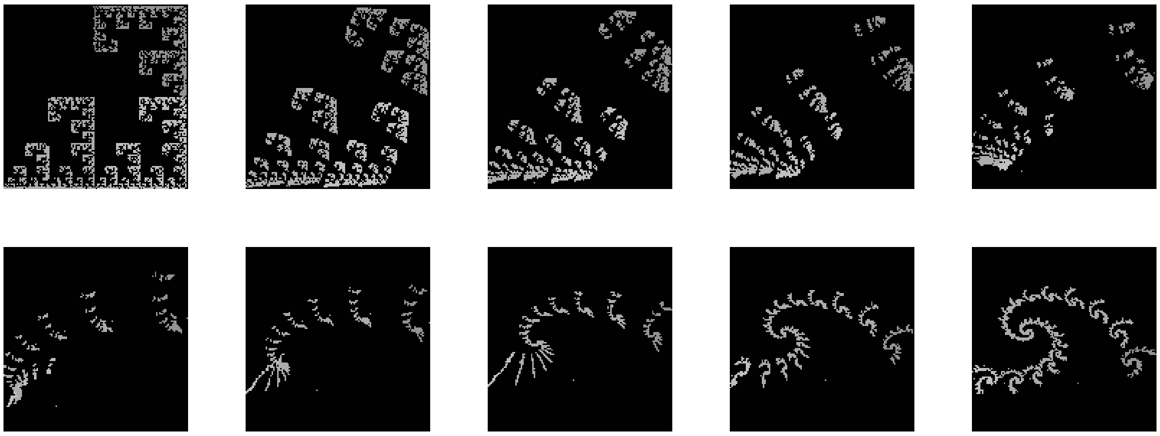

Figure 4: Example of a genetic crossfade between IFS fractals

5 Conclusion

I believe I have succeeded in making a general toolkit for exploring visual genetic programming. While other people have yet to write modules for it, I believe the interface between the modules and the toolkit is flexible enough to implement many problems. Judging from the reactions of users at the Geometry Center, it is an easily understood, useable, and addicting system for exploring visual genetic programming.

6 Future Work

There are many additions or changes to the toolkit which would make interesting projects. One is to make a World Wide Web version of the toolkit. Another is to expand the current modules and to write others for such things as cellular automata, L-systems, implicit and parametric surfaces, and music. An automatic selector could be added to the toolkit to aid the user in their search. This "helper" could be based on a neural network [BPJ], or a more analytic fitness function. A useful addition would be support for more traditional genetic algorithm programming.

7 Acknowledgements

I would like to thank Tony Phillips, the coach, for being supportive during my long search for a project. Bing Cheng, Li-Tien Cheng, Nils McCarthy, and Jovan Popović served as the first users of the system and provided many useful comments and suggestions. Nils was intrigued enough by the genetically produced images to write the cross-fader for them. I would also like to thank the staff and administrators of the Geometry Center for providing such a positive and creative work environment.

References

[Ben] Brent Benson. libscheme. `ftp://ftp.cs.indiana.edu:/pub/sunfish/scheme-repository/imp/libscheme-0.3.tar.Z`.

[BPJ] Shumeet Baluja, Dean Pomerleau, and Todd Jochem. Towards automated artificial evolution for computer generated images. Submitted to *Connection Science*.

[Bri] Allan Brighton. Tree widget. `ftp://harbor.ecn.purdue.edu/pub/tcl/extension/tree3.6.tar.gz`.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[Mac] Paul Mackerras. Photo widget. `ftp://harbor.ecn.purdue.edu/pub/tcl/extensions/photo-2.4.tar.gz`.

[Ous] John Ousterhout. Tk/Tcl. `ftp://ftp.cs.berkeley.edu/ucb/tcl/`.

[Sim91] Karl Sims. Artificial evolution for computer graphics. *Computer Graphics*, 25(4):319–328, 1991.