Brock University

Department of Computer Science

Gentropy: Evolutionary 2D Texture Generation

Andrea L. Wiens and Brian J. Ross
Technical Report # CS-00-02
May 2000

Brock University
Department of Computer Science
St. Catharines, Ontario
Canada L2S 3A1
www.cosc.brocku.ca

Gentropy: Evolutionary 2D Texture Generation

Andrea L. Wiens

50 Lakeshore Road, Unit #94
St. Catharines, Ontario,
Canada L2N 6P8
janet.wiens@sympatico.ca

Brian J. Ross

Department of Computer Science
Brock University
St.Catharines, Ontario,
Canada L2S 3A1
bross@cosc.brocku.ca

Abstract

Gentropy is a genetic programming system that evolves two-dimensional textures. It creates textures by combining common mathematical and image manipulation functions into formulas, which are normally impossible for a person to devise on his/her own. An entire texture spans two-dimensional space and is represented by a relatively small formula. At any time, the formula can be re-evaluated with arbitrary texture-space coordinates, to generate a new portion of the texture. Most evolutionary art programs require the user to repeatedly choose the best images from a displayed generation. Adversely, *Gentropy* uses an unsupervised approach, where a target image represents the user's desired features, such as colour, shape and smoothness (contrast). Then, *Gentropy* evolves textures independent of any further user involvement. As this is an art application, an exact match to the target is *not* desirable. A set of feature tests may be combined and given different priorities in the fitness function. It is possible to use several target images, each with its own fitness function, set of desirable features. Moreover, subpopulations may have unique fitness functions. A multiple subpopulation approach is used where several divisions of the population exchange textures.

1 INTRODUCTION

More and more, computers are being used for artistic purposes, such as the generation of interesting images and textures for desktops, webpages, documents and animation programs. While a computer program has no concept of creativity, the computer is a powerful tool to be used by an artist for creating fascinating textures. Simple random texture generation is not practical. Consider this: of all the billions of gene configurations possible, only a small portion will be compatible with life. Similarly, of all the possible textures a computer program may produce, only a small portion will be pleasing to the human user. Genetic algorithms (GA) offer a better solution, by simulating the process of evolution (Koza, 1992). It is an effective search scheme for finding a texture with certain visual features, such as colour and shape.

GA has already been applied to this problem. In the majority of evolutionary art (evoart) applications, the user is presented with a small population of images using a real-time display and asked to choose several for breeding (Rowbottom, 1999). In this situation, the human is the fitness function and, since he/she directs the evolution, supervision of the system is required. Some examples are Karl Sims' LISP-based system for evolving pictures and textures, Juan Luis Abadia's *Gaia* and Steven Rooke's program (Sims, 1993; Abadia, 1997; Rooke, 1993). Alaa Eldin Ibrahim's *Genshade* includes an unsupervised option, where target images are used to direct the evolution of high-level Renderman shaders, defined as connected graphs (Ibrahim, 1998). Gentropy takes a similar approach but with a low-level, mathematical function set, combined to define texture formulas.

Why use algorithmic texture formulas? They are capable of an infinite variety of complex patterns and their mathematical nature means that they vary across texture space. Furthermore, a formula is more compact than a bitmap. You need only store the formula, and a program to evaluate it, in order to reproduce any section of the texture. This saves storage space, as image files are large and cannot possibly hold an entire texture; it is a form of compression.

Section 2 discusses the mathematical nature of textures, the function set and the evaluation of texture formulas. Section 3 outlines image analysis techniques used in the fitness function. Section 4 describes the genetic programming system used, including parameters used and strategies attempted. Section 5 shows several results of experiments conducted. Finally, section 6 states some conclusions reached and future extensions to *Gentropy*.

2 COMPUTER GRAPHICS AND TEXTURES

Computer images are usually represented as a collection of many small *pixels*, each with its own colour, arranged in a two-dimensional grid. *Gentropy* uses the RGB format to acquire and produce images. Any colour is divided into three real values called *channels*. These channels represent the amounts of red, green and blue in the colour, hence the name RGB.

2.1 TEXTURE FUNCTION SET

The texture function set consists of various mathematical and image processing functions that take an RGB colour (vector) or channel (scalar) as arguments and return either a vector or scalar. A vector is represented by three floating-point values, while a scalar is a single floating-point number, defining a grey colour.

```
X Y ERC VERC RADIAL COLGRAD + - diff * / sin cos
mod log pow atan not avg max min if noise turb
turbflow cloud marble warprel warpabs kaleid
tile tilerad lum rgb hsv yiq forv chn ichn
```

These functions are combined into a *texture formula* in such a way that the output is always an RGB colour vector. Variables are added to the function set, such as `X` and `Y`, which give the position within two-dimensional space (corresponding to the current pixel position), so that the formula may produce a different result (colour) for each pixel when it is evaluated. Tables 1 and 2 show several simple texture examples, followed by a few more complex ones. Two texture representations are used: an image and the texture formula. Many of the images are colourful.

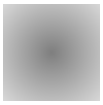
The majority of the functions are inspired by Karl Sims' LISP functions used in his texture evolution experiment (Sims, 1993). Terminals (constants, and functions that take no arguments) have upper-case names. `ERC` returns a random scalar and `VERC`, a random vector. They appear in a texture formula as either a floating point number or a set of three floating-point numbers within parentheses. Circular effects are generated by `RADIAL`, which gives the distance to the origin, is taken from *Gaia* (Abadia, 1997). In `COLGRAD`, which creates a colour gradient, the red and green values are the current x and y positions and the blue value is the distance to the origin. The `noise`, `turb`, `turbflow`, `cloud` and `marble` functions are more complex, generating shaded bumps, curves, gases and other natural-looking effects (Ebert et al, 1994).

Several warp and tiling operations, which alter the positions of or duplicate pixel values, are parts of the function set. In addition to relative (`warprel`) and absolute (`warpabs`) warps and a kaleidoscope effect (`kaleid`), there are tiling operations, which create repeating patterns, both rectangular (`tile`) and circular (`tilerad`).

The `lum` operation finds the luminance, average of all channels, which is a grey colour. Also, there are several colour space conversion functions (`rgb`, `hsv` and `yiq`), which treat their vector argument as a colour in their particular space and then convert it to RGB (Foley et al, 1994).

The final set of three operations performs an iteration over a finite colour vector, which is an automatically defined iteration (ADI) (Koza, 1999). The function `forv` performs the actual iteration, like a `for` loop in C or `dotimes` in the LISP programming language, with the current channel as the looping variable. Operations `chn` and `ichn` either return the appropriate scalar value (one channel of a colour vector) based on the current channel or the current channel plus some increment value, respectively. Before the ADI was added, there were mathematical and noise functions for both scalars and colour vectors. Afterward, the number of these functions was cut in half! The vector (colour) versions were eliminated because the ADI is able to take scalar functions as a subtree (part of the overall texture function tree) and evaluate it three times to produce a colour or a grey. If there is a `chn` or `ichn` in the subtree, different values will likely be returned for each iteration, as the channel varies, and `forv` returns a (non-grey) colour. Otherwise, each iteration will have the same result and a grey colour will be produced. For example, the first texture in Table 2 is green, as opposed to grey, because there is a `chn` in the `forv`'s subtree.

Table 1: Simple Texture Examples

IMAGE	TEXTURE FORMULA
	`X`
	`RADIAL`

	`(rgb RADIAL`
	`(turbflow X RADIAL RADIAL)`
	`1.00)`

	`(rgb (- RADIAL -0.99)`
	`(turbflow 0.07 -0.25 X)`
	`RADIAL)`

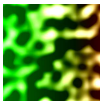
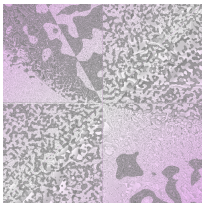
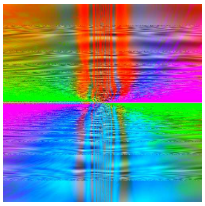
	`(yiq`
	`(cloud 0.25 X -0.65 0.26)`
	`X`
	`(+ X -0.60))`

Table 2: More Complex Texture Examples

IMAGE	TEXTURE FORMULA
	`(forv (diff (noise (min (min Y RADIAL) RADIAL) (mod (diff RADIAL RADIAL) (max RADIAL Y))) (chn (-0.34,-0.76,-0.33))))`
	`(hsv (sin (max Y 0.68)) (/ (diff X Y) (/ X X)) (/ (/ (diff (/ X RADIAL) (/ (pow Y X) (cloud Y -0.14 -0.88 0.70))) (cloud 0.72 Y Y 0.40)) (noise Y RADIAL)))`
	`(yiq (/ (noise (pow -0.44 -0.37) (mod (ichn RADIAL (0.79,0.92,-0.31)) Y)) (- -0.37 RADIAL)) (+ (cos (/ (avg (turb (noise X -0.62) (atan RADIAL)) 0.90) X)) (/ (mod Y 0.58) RADIAL)) (* (/ X Y) (turb (tilerad RADIAL (turbflow (ichn (+ RADIAL X) (rgb Y X RADIAL)) (min (sin X) (cos Y)) (log (turb RADIAL RADIAL)))) (/ (* (tilerad X RADIAL) (avg RADIAL X)) (cos Y)))))`

2.2 EVALUATION

One of the evoart applications described on the Web is *Gaia*. It stores both a formula and a domain, which determines where the formula is evaluated, for each individual in order to generate an image (Abadia, 1997). In contrast, the domain in *Gentropy* is fixed for all individuals during evolution and is adjustable afterward to allow any portion of the final texture to be generated.

Gentropy uses a domain of -1 to 1 (about the origin) for the x and y positions but the texture formula may be evaluated anywhere in the two-dimensional plane, for example from 0 to 100. Note that evaluating about the origin tends to create symmetric patterns. For each pixel position, x and y coordinate, the texture formula is evaluated to produce a colour. The result of the evaluation process is an *image*, which shows a small portion of the texture.

3 IMAGE ANALYSIS

Selection of individual textures, for crossover and mutation, depends on the fitness of the individuals. *Gentropy*'s fitness function assigns every individual texture a fitness measure between 0 and 1, with 1 indicating a perfect score, which is used for tests like selection and run termination.

3.1 IMAGE COMPARISON

Quantization is used to ease matching, particularly the shape aspect. It slots the colours so that very similar colours are represented by the same colour. This gets rid of minute colour differences that the human eye normally cannot judge anyway. However, this does introduce some error to the matching process. For instance, the fitness function may deem two images identical, even though their colouring or shape is slightly different. The colour space — or number of colours involved — is reduced so that histograms are smaller.

Histograms represent the colour distribution of an image by storing the frequency of each quantized colour (Smith, 1997). They are used by many of the feature tests in *Gentropy*, including those that match directly and those that match histograms. *Direct matching* involves the comparison of two images, pixel by pixel, while *histogram matching* calculates a histogram for the images and then measures the distance between the histograms. Position is irrelevant in the latter but not the former. In other words, direct matching includes shape information in its analysis.

3.2 FEATURE TESTS

Gentropy uses image comparison methods, mainly adapted from *query by image content* (QBIC) applications. These programs take a target image and search a database for similar images. They often use histograms and wavelet analysis to match images. Wavelets and image query techniques are also used by *Genshade* (Ibrahim, 1998). Table 3 shows a few of the feature tests the user may choose from as part of the fitness function.

Table 3: Several Feature Tests

TEST	DESCRIPTION
Colour Direct (`CDIR`)	Matches colour similarities pixel by pixel
Colour Histogram Quadratic (`CHISTQ`)	Matches similar colours, position irrelevant
Wavelet (`WAV`)	Matches shape using wavelet theory
Smoothness Histogram (`SHIST`)	Matches colour smoothness (contrast), position irrelevant

It is extremely difficult to do an accurate automatic evaluation of an image. *Gentropy* uses various image analyses, which evaluate basic image attributes, such as colour, shape and smoothness. Complex analyses, from computer vision research for instance, would result in more accurate evaluations of images, but a serious price would be paid in run-time speed.

Colour direct (`CDIR`) is the simplest matching operation. It compares two quantized images by adding the colour similarities of each pixel in the two images, where colour similarity is the distance between the colours in RGB space. Unlike `CDIR`, colour histogram quadratic (`CHISTQ`) is not concerned with the colour's position within the image. It creates colour histograms from the two quantized images being compared and then finds the quadratic histogram distance. This method is used in QBIC systems like VisualSEEk (Smith and Chang, 1996).

Let `i` and `j` be indexes into the two histograms (`hist1` and `hist2`), representing quantized colours whose frequencies the histograms measure. The similarity between the colours, represented by the histogram indexes of `i` and `j`, is given by `sim(i,j)` and `abs` indicates the absolute value. The distance between these two indexes is given by:

```
dist(i,j) = abs(hist1[i]-hist2[i]) * sim(i,j) *
            abs(hist1[j]-hist2[j])
```

This distance is summed for all of the possible combinations of `i` and `j` (depending on the histogram size), in order to obtain an overall measure of histogram difference: the colour histogram quadratic distance. The computation is expensive but it allows the testing of the closeness between colours, as opposed to merely considering exact colour matches.

The wavelet (`WAV`) test compares the shape of two images. It creates a frequency image using the colour histogram of the quantized image. Colours with low frequency, as determined by the histogram, are replaced with dark greys, while frequent colours are assigned lighter greys. The result is a grey image with the same shape as the original colour image, but the specific colours are removed. This frequency image can now be used to match shape, without regard to the actual colours involved.

Wavelet analysis is useful because its shape matching precision is adjustable by specifying the number of coefficients to be kept after truncation. `WAV` uses a basic Haar wavelet decomposition, where each row and then each column in the image is recursively compressed using an averaging and differencing operation and the resulting coefficients of least magnitude are truncated (Stollnitz, Derose and Salesin, 1996). The result of this procedure is a wavelet image of the most significant coefficients, either positive or negative, represented as white or black pixels, respectively. The final step is the comparison of two wavelet images by matching the positions of the white and black pixels.

Notice that the two example wavelet images in Table 4 share a few coefficients (white or black dots in the same position) but they are quite different.

Table 4: Example Quantized, Frequency and Wavelet Images Used in Shape Comparison

QUANTIZED	FREQUENCY	WAVELET

Smoothness histogram (`SHIST`) matches smoothness or contrast by generating a smoothness image, consisting of grey colours indicating the deviation of each pixel from its neighbours. Colour similarities between the particular pixel and its surrounding pixels are averaged to find a local deviation measure. These values make up a smoothness image, for which `SHIST` creates a histogram. Then, the distance between two such histograms provides the smoothness measure.

4 GENETIC PROGRAMMING SYSTEM

Gentropy uses the strongly typed *lil-gp* kernel, a genetic programming (GP) system written in the C programming language (Luke, 1997). Table 5 shows several GP parameters and Table 6 lists a few settings specific to *Gentropy*.

Subpopulations in *Gentropy* may be given their own fitness functions with unique feature tests and weights. Different subpopulation topologies (arrangements) are possible. There is always a main subpopulation, which contains the final image. Evolution stops with the best fitness of this subpopulation is greater than some threshold value (0.99) or has not improved for a while (25 generations). If there are multiple subpopulations, each evolves separately but exchanges five individuals with the main subpopulation every ten generations. This provides the main subpopulation with a regular injection of fresh, hopefully useful, textures. Complex topologies may be designed where several subpopulations, each with different but related fitness functions, exchange individuals with each other.

There are problems with simply adding several feature tests together. For example, textures that have a high colour fitness may not necessarily have a good shape score. Evolution may try to find the highest possible colour score, without improving the shape fitness. Adding these fitness scores together equally — trying to search in various directions at once — may result in the cancelling of their effects. Hence subpopulations are useful in permitting different populations to concentrate on various features, which are then combined elsewhere. This attempts to find as many good examples of different features as possible. One subpopulation can look for textures with good colour, while a another subpopulation

hunts for textures with good shape. These individuals are then exchanged with the main subpopulation, which measures both colour and shape. Then, some of the exchanged textures should be useful to the search for a texture with good colour and shape.

Table 5: GP Parameters

PARAMETER	VALUE
Maximum nodes	100
Maximum depth	50
Initialization method	half and half
Initial depth	2 to 6
Crossover selection	tournament, size=5
Crossover rate	0.9
Mutation selection	tournament, size=5
Mutation rate	0.1
Subpopulation size	1000
Number of subpopulations	1, 3, 4 or 7
Number of exchanges	0, 2, 3 or 9
Subpopulation exchange interval (in generations)	10
Exchange count (number of individuals)	5
Exchange selection methods	best and random
Maximum generations	250

Table 6: Application Parameters

PARAMETER	VALUE
Threshold (stop when fitness >)	0.99
Generation wait (when no progress, stop after this many generations)	25
Number of quantized greys	25
Number of quantized colours	1000
Number of wavelet coefficients used	25
Evaluation/Target width and height	50
Final image width and height	500
Number of target images	1 or 2

Colour and shape matching precision is adjustable by specifying how many greys, colours and wavelet coefficients are to be kept after quantization or truncation. Recognize that setting these values too low will likely cause very different images to be treated as similar. On the other hand, retaining too much colour and shape information, setting values too high, causes evaluation time to be increased, as the fitness function looks for a near exact match to the target image.

Although the evaluation size of the generated images is equal to the target size (or the minimum of all target sizes), the final image size may be much larger. The target images are all 50 by 50 pixels, which means that 2500 pixels are evaluated by the fitness function for each individual. The tables in section 5 show two image sizes: the evaluation size, to show what the fitness function uses, and the final size (500 by 500 pixels, reduced here). Notice that the left image is a small central portion of the right image.

Gentropy allows the use of multiple target images. It is up to the user to specify which features are desirable in each target and the weight they are given. This removes the need for the user to find one target image that contains all desired criteria in the correct proportions. Furthermore, this allows the results of previous runs of the program to be used together as targets, seeding subsequent runs with appealing textures. This should result in a combination of features from the interesting textures in the final image.

5 RESULTS

Gentropy runs on a 16-CPU Silicon Graphics Origin server. Each CPU is a 250 MHz MIPS R10000. Typically, all runs for a given experiment were executed concurrently on separate processors.

The average run time of the results presented is about 20 hours and the mean generation of the best individual texture is 96. Runs that last for more generations tend to have more nodes and a greater depth. The average number of nodes is 82 and the mean depth is 10.

Table 7 shows several fitness functions and subpopulation topologies for the results reported in this paper. The first is the simplest possible: a single subpopulation using one feature test (so its weight is 100% of the fitness score). Recall that `CDIR` simply matches colour similarities pixel by pixel.

Next are two fitness functions that vary only in the weights of their main subpopulation (top). It is a multiple subpopulation and target topology, where there are three subpopulations and two target images. The lower two subpopulations exchange textures with the main one. Every ten generations, the five best textures of each of the lower subpopulations go up the arrow and the main subpopulation sends five random textures back. The main subpopulation compares the generated image to the colour of target 1 (`CHISTQ`) and the shape of target 2 (`WAV`), either with equal weight or with shape determining the majority

(70%) of the fitness score. The other subpopulations measure the colour of target 1 or the shape of target 2.

Tables 8, 9 and 10 display the target image(s) involved, the fitness score of the best image and the fitness breakdown of the best fitness score (if applicable), showing the contribution of each of the feature tests. The final two columns show the best image and a larger portion of the corresponding texture: the final image. The fitness function compares images ("Eval. Image") to the target images ("Targets") to get a fitness score ("Fit"). Each row shows the results of a particular run of the program.

Notice that the best fitness scores are in the area of 70 to 80%. This fits with the images generated and the priorities of the particular fitness functions. Notice that a near perfect score is not necessary in order to gain the visual effects desired.

The feature test named `CDIR` works better than expected but it tends to find generalized versions of the target. Notice that the mottling is missing or less obvious. The top image has most the colours of the target but not the gradient. In some applications it may be desirable to find a simplified version of the target, such as for a background.

The multiple subpopulation results below do not involve `CDIR`. Their fitness functions are more complex, adjustable and the fitness breakdown may be analyzed.

Although colour and shape are given equal weight in Table 9, in the main subpopulation, the Fitness Breakdown column shows that colour is dominant. Clearly, the shape score (`WAV`) makes up the least portion of the overall fitness. Fortunately, the influence of shape on the fitness may be improved by increasing its weight as compared to other feature tests.

In Table 10, notice the many stripes in the generated images like target 2. Also, colours not found in target 1 are permitted in the generated images. This is useful when the user is not concerned with particular colours.

Table 7: Multiple Target Results

TABLE	FITNESS FUNCTION
6	`CDIR=1.0`

7	`CHISTQ=0.5,WAV=0.5`
	`CHISTQ=1.0 (target1)`
	`WAV=1.0 (target 2)`

8	`CHISTQ=0.3,WAV=0.7`
	`CHISTQ=1.0 (target1)`
	`WAV=1.0 (target 2)`

Table 8: Single Subpopulation Results

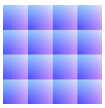
TARGET	FIT	EVAL. IMAGE	FINAL IMAGE
	0.79		
	0.80		

Table 9: Multiple Subpopulation and Target Results

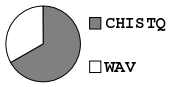
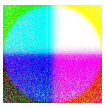
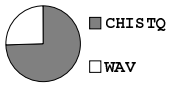
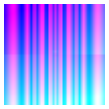
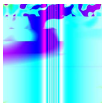
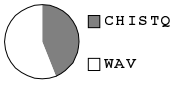
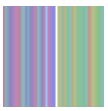
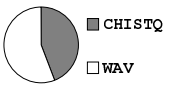
TARGETS	FIT	FITNESS BREAK-DOWN	EVAL. IMAGE	FINAL IMAGE
 Target 1 Target 2	0.77	 CHISTQ WAV		
	0.73	 CHISTQ WAV		

Table 10: Multiple Subpopulation and Target Results (Shape Emphasis)

TARGETS	FIT	FITNESS BREAK-DOWN	EVAL. IMAGE	FINAL IMAGE
 Target 1 Target 2	0.70	 CHISTQ WAV		
	0.70	 CHISTQ WAV		

6 CONCLUSIONS

This research successfully used genetic programming to perform automatic texture generation. Given the difficulty of automating “aesthetic sensibilities”, Gentropy’s use of a user-selectable set of image analysis fitness functions was shown to be useful. Shape matching is the most difficult analysis, but it is possible to compensate for this by increasing its weight in the fitness function. Give `WAV` a higher priority and the evolution will concentrate on shape and allow more colour variation to the target.

Although several feature tests and subpopulation topologies were tried, many more are possible. For example, edge and texture analysis tests may prove useful. Also, alternatives to the use of weights may be investigated, such as the use of dynamic weights that change during evolution and the averaging of different feature test scores.

One method of increasing *Gentropy*'s speed is through the use of multi-threaded GP. Each subpopulation could run on one or more processors and truly evolve in parallel. Should the program be able to run in real-time, more user interaction would be possible. A faster, user-friendly *Gentropy* could be made into an evoart program, where the artist is free to stop the evolution when an interesting texture develops. This may then be stored in a library, like Steven Rooke's "digital amber", for use as target images later (Rooke, 1993). Several textures from the library may be combined in multiple target runs. In this way, the artist adds his/her creativity to the process.

Gentropy's image representation and texture function set are based on Karl Sims' LISP-based texture generation system (Sims, 1993). Juan Luis Abadia's *Gaia* has useful functions too, as well as the concept of an adjustable domain in the image evaluation process (Abadia, 1997). At the present time, *Gentropy* evaluates a small portion of the texture. Often the rest of the texture shares features with this portion. The fitness function may operate by adjusting the domain, evaluating the texture at different places in two-dimensional space or at different resolutions (dimensions) to see how it matches the target.

Like *Genshade* (Ibrahim, 1998), *Gentropy* takes an unsupervised approach, where a fitness function judges the similarity of generated images to one or more targets. However, *Gentropy* has a lower-level function set and allows any image to be used as targets, including both images generated by the program and images from other sources (scanned, drawn or generated by another program). Furthermore, *Gentropy* allows the user to specify desired features — as well as the weight — of each target and every generated texture is compared with the targets in the same way. In this way, the user has control over how the targets affect the fitness.

Genshade does not appear to use subpopulations with different fitness functions. There are many subpopulation and fitness topologies possible in *Gentropy*. In some cases, the simplest strategy, with a single subpopulation and feature test, is useful. However, more complex strategies, such as those that use several weighted feature tests, allow the user to better specify his/her preferences in the final texture. The multiple target approach works far better than expected and is able to effectively combine different features from different targets in the generated image!

Acknowledgments

This research is supported through NSERC grant 138467-1998.

References

J. L. Abadia (1997). *Gaia: Generating New Images using Genetic Algorithms*. <http://www.iua.upf.es/~jlabadia/gaia/>

D. S. Ebert et al (1994). *Texturing and Modeling: A Procedural Approach*. Toronto: Academic Press. p.86-203.

J. Foley et al (1994). *Introduction to Computer Graphics*. New York: Addison-Wesley Publishing. p.410-416.

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

J. Koza (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, Mass.: The MIT Press.

J. Koza (1999). *Genetic Programming III: Darwinian Invention and Problem Solving*. San Francisco, CA: Morgan Kaufmann Publishers. p.121.

S. Luke (1997). *Patched lil-gp Kernel*. Computer Software. University of Maryland at College Park. <http://www.cs.umd.edu/users/seanl/gp/patched-gp/>

S. Rooke (1993). *The Genetic-Evolutionary Art Process of Steven Rooke*. <http://www.azstarnet.com/~srooke/process.html>

A. Rowbottom (1999). Evolutionary Art and Form. *Evolutionary Design By Computers*. Ed. Peter Bentley. San Francisco: Morgan Kaufmann Publishers.

K. Sims (1993). Interactive Evolution of Equations for Procedural Models. *The Visual Computer* 9, 466-476.

J. R. Smith (1997). *Integrated Spatial and Feature Image Systems: Retrieval, Analysis and Compression*. Ph.D thesis, Graduate School of Arts and Sciences, Columbia University (JRS97). <http://www.ctr.columbia.edu/~jrsmith/html/publications.html>

J. R. Smith and S-F. Chang (1996). *VisualSEEk: A Fully Automated Content-based Image Query System*. <http://www.ctr.columbia.edu/~jrsmith/html/pubs/acmmm96/acmfin.html>

E. Stollnitz, T. Derose, and D. Salesin (1996). *Wavelets for Computer Graphics: Theory and Applications*. San Francisco: Morgan Kaufmann Publishers. p.44-45.