BRIDGES
Mathematical Connections
in Art, Music, and Science

Mathematical Building Blocks for Evolving Expressions

Gary R. Greenfield
Department of Mathematics & Computer Science
University of Richmond
Richmond, VA 23173, U.S.A.
E-mail: ggreenfi@richmond.edu

Abstract

We consider the problem of analyzing visual imagery obtained using Sims' method of evolving expressions from the perspective of the mathematical building blocks, or function primitives, that are used to form such expressions. We survey previously used sets of building blocks and then give the design principles and motivation for our latest set of such building blocks. This includes a new construction of building blocks starting from arbitrary continuous functions. We conclude by incorporating our building blocks into a coevolutionary system and show examples of images that emerged from the "primordial ooze" defined by our building blocks.

1. Introduction

Richard Dawkins formulated the principle of interactive evolution for use with populations of primitive drawing routines in his *Biomorphs* program [2]. Dawkins' key concept was "fitness by aesthetics," which meant that the survivability of an image was determined by the user. Karl Sims expanded upon the idea by turning interactive evolution into an art *medium* [12]. The new feature Sims introduced was to free image generation from procedural drawing routines by encoding images as expression trees. Methods for organizing and maintaining expression trees as evolving populations had been developed previously by Hillis [6] and Koza [7]. Therefore, perhaps Sims' most significant contribution was his recognition of the importance of the mathematical building blocks that needed to be incorporated into such a system in order to create imagery using the principles of artificial evolution and fitness by aesthetics. Those following in Sims' footsteps were forced to design and implement their own systems based on evolving expressions entirely from scratch due to the computational limitations underlying Sims' original methods. Because there are no standards and only fragmentary descriptions of how the essential mathematical building blocks are defined — they often constitute the proprietary component of the system — comparison of the art that has been produced, as well as evaluation of the capabilities of the image generating systems, has been difficult. In this paper we will review what is known about previous efforts and we will discuss the mathematical building blocks we designed for a new system to generate images using coevolution *i.e.* where aesthetic fitness decisions are algorithmic thanks to *two* populations, a population of hosts which produce images and a population of parasites which consume images.

This paper is organized as follows. In section two we recall the technical details for producing images from expressions. In section three we discuss the building blocks used by previous designers of systems of evolving expressions. In section four we describe how we designed our latest set of building blocks. In section five we discuss some coevolution experiments using our set-up.

2. Images from Expressions

A (symbolic) expression is a rooted tree. The nodes of the tree contain the function primitives that are used to build the expression. We use the terms function primitive, basis function, building block, or operator interchangeably. Function primitives are distinguished by their *arity*, the number of arguments the function requires. Typically, one limits the arity to at most three. The leaves of the tree contain the inputs to the expression so they must be chosen from the set of basis functions of arity zero. Any function $F(u, v)$ from, say, the unit square to the unit interval can be written as an expression tree in one of three ways: using prefix notation ($F\ u\ v$), using infix notation ($u\ F\ v$), or using postfix notation ($u\ v\ F$). Adopting the familiar prefix notation, any expression E can therefore be thought of as a function from the unit square to the unit interval that is formed by functional composition as, for example, $G(H(v), K(v, u))$. The external representation of the expression E that we choose to adopt does not affect its value $E(u, v)$. By resolving the unit square into a grid of points (u_i, v_j), calculating the expression $E(u_i, v_j)$ over all points on the grid, and then mapping those values to colors, we obtain an image from the expression. In artificial life terminology the genotype (expression) yields the phenotype (image). The above example clearly demonstrates why the set of primitives wholly dictates the type of imagery that it will be possible to discover when exploring the resulting image space of expressions. The richer the set of building blocks, the more interesting the images. There is, however, one further complication. Images from expressions may be based on either intrinsic color or extrinsic color. For intrinsic color the expression's value is either a vector of colors in a standard color space such as HSV space or RGB space, while for extrinsic color an expression's value is used as a look-up index into a color table for either one channel in the color space or the full palette of colors.

3. Previous Examples

Sims [12] actually describes *three* sytems for producing images: one for 2d imagery, one for 3d imagery, and one for animation. We will consider only his 2d system based on extrinsic color. We denote by A_i the set of primitives of arity i. It appears that the functional primitives Sims used are:

$$
\begin{aligned}
A_0 &= \{X, Y, \text{together with various constants}\},\\
A_1 &= \{\textit{round, expt, log, atan, sin, cos, invert(?), if, ifs}\},\\
A_2 &= \{+, -, *, /, \textit{mod, min, max, and, or, xor, bw-noise, color-noise}\},\\
A_3 &= \{\textit{hsv-to-rgb, vector, transform-vector, warped-color-noise, blur, band-pass, grad-mag}\\
&\quad \textit{grad-dir, color-grad(?), bump, warped-ifs, warp-abs, warp-rel, warp-by-grad}\}.
\end{aligned}
$$

The reason we are uncertain about some of the placements in this classification is that because Sims' system is extrinsic it is not always clear which functions are defined componentwise and which are not. To cite one example, since *ifs* stands for iterated function system, one presumes *if* stands for iterated function, and in either case these could be true vector functions or vector functions induced by being defined componentwise. Also, some primitives only appear in Sims' examples. (E.g. *invert* and *color-grad* only appear in conjunction with Figure 9 on page 325 of [12].) Because of the sophisticated use of blurring, warping and other image processing functions, the implementation details of which are not given, it is not surprising that Sims' successors have seldom matched his image making prowess.

The next system about which we have some information is a bare-bones implementation by Baluja et al [1] which was used as an experimental testbed for attempting to train an artificial neural net to guide the interactive evolution of the images. Their function set is given as

$$\begin{aligned} A_0 &= \{x, y\}, \\ A_1 &= \{\textit{reciprocal, natural log, common log, exponent, square, square root, sine, hyperbolic sine, cosine, hyperbolic cosine}\}, \\ A_2 &= \{\textit{average, minimum, maximum, addition, subtraction, multiplication, division, modulo, random}\}. \end{aligned}$$

They remark that *random* is a function for randomly selecting either the first or second argument of the function and is not a function for generating random values. Here is a convenient spot to point out that designers using the *divide* function use the aptly named "protected divide" to prevent division by zero.

In the user's manual for an X-windows system available on the web, Unemi lists his functional primitives [13] as

$$\begin{aligned} A_0 &= \{XY0, YX0, \text{together with various constants}\}, \\ A_1 &= \{-, \textit{abs, sign, sin, cos, log, exp, sqrt, image}\}, \\ A_2 &= \{+, -, *, /, \textit{pow, hypot, max, min, and, mdist, mix}\}. \end{aligned}$$

The *image* function signifies that one must use a source image provided at run time. Presumably *mix* is a blending operator. One interesting feature of the system is that one can set on/off switches to delimit the set of primitives that will be available for use in the genotypes.

In [4] we gave a complete listing of the function primitives for a variant of a Sims' style system. We reproduce the complete listing here although without the primitive's definitions.

$$\begin{aligned} A_0 &= \{u, v, c, e\} \\ A_1 &= \{\textit{sin, cos, exp, log, abs, sqrt, square, cube, not}\} \\ A_2 &= \{\textit{and, add, multiply, mod, min, max, pwr, vee, cir}\} \end{aligned}$$

The leaf node symbol c signifies that a constant is stored at the node, while e, instead of accessing a source image, is a re-direction operator to a source expression (see [4] for details). Most of the binary primitive functions will be discussed further in the next section. [Note: For future reference *cir* will become *cone* and *vee* is a variant of *zabs*.]

For 2d systems these are the only complete listings of building blocks that we are aware of. Though we keep an archive of sample genomes from systems that have been implemented, they do not yield the complete function primitive sets, so our subsequent discussion has many gaps. One system, which for reasons easily explained, produces imagery that most would describe as "fractal" is the system found on the web `http://www.cs.cmu.edu/~jmount/g3.html`) by Mount

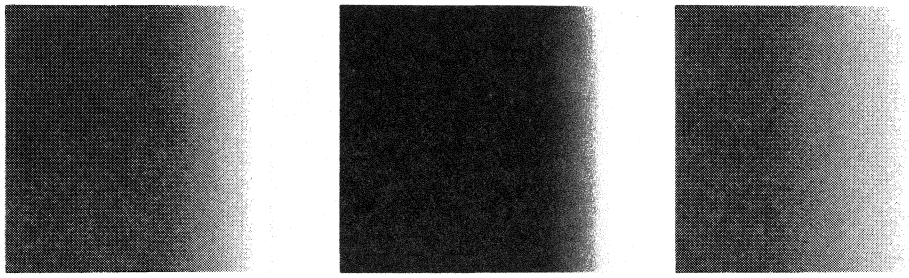

Figure 1: (a) A linear black-to-white color ramp. "Smoothing" operators for the ramp are defined using (b) the squaring function and (c) the square root function.

and Witbrock [14]. It uses quaternion operators including rotations, inversions, and conjugation together with the usual arithmetic operators. Similar remarks apply to the genotypes built from function primitives defined on the complex plane in a system that appears on the web by Yoshiaki [15]. Steven Rooke has for many years been producing images from his evolving expressions system. A sample description of one of his genomes reveals heavy use of both complex and quaternionic primitives, iterated functions, and of course the usual complement of arithmetic, boolean, and trigonometric functions [11]. Rooke also makes significant use of random constants. On the other hand, Musgrave, in constructing a prototype Sims' style system, followed Sims' original image processing theme more closely by including many noise, turbulence, and wave image processing functions but also added significant image generating capability by including fractal Brownian motion primitives [10]. Finally, Ibrahim [8] adopts a design where all internal nodes have arity four, but the primitive functions are Renderman shaders, so comparison of the primitives used is not justified.

4. On Designing a Set of Building Blocks

We will restrict ourselves to the problem of designing primitives for 2d image generating systems based on extrinsic color. We remind the reader that for us this implies an expression E maps the unit square to the unit interval. The set of terminals of arity zero must include variables for the coordinate axes. To avoid confusion with systems we have discussed earlier we shall use generic coordinates (V_0, V_1) instead of the usual choices of (x, y) or (u, v). The next decision to be made is whether or not to allow constants. We will include distinct generic constants $C_1, \ldots, C_n$ lying in the unit interval. The problem of introducing background imagery, source imagery, or secondary imagery is a thorny one with many factors to consider. It is intimately connected with image resolution, language dependencies, and choice of direct or indirect access to the source imagery. To avoid unnecessary complexity, we will not consider it here. Thus we will fix our arity zero set as:

$$A_0 = \{V_0, V_1, C_1, \ldots, C_n\}.$$

We now turn to unary operators of arity one. These operators are perhaps the most misunderstood ones. What is their purpose? If these basis functions are continuous, then we think of them as either smoothing operators or redistribution operators. Why? Let us suppose we are using a simple color *ramp* from black (zero) to white (one) having 255 increments such as $F(V_0) = V_0$ (Fig.

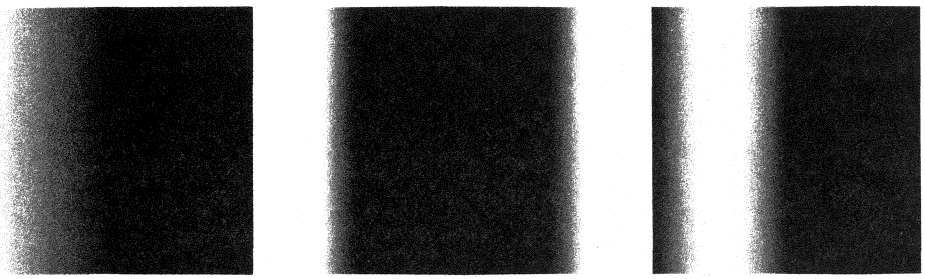

Figure 2: Redistributing the linear color ramp using (a) an inversion operator, (b) a parabolic map, and (c) a sine function.

1a). Then, $F(V_0) = V_0 * V_0$ (Fig. 1b) and $F(V_0) = \sqrt{V_0}$ (Fig. 1c) smooth the ramp towards black and white respectively. On the other hand, the inversion operator $F(V_0) = 1 - V_0$ (Fig. 2a), the parabolic operator $F(V_0) = 4*(V_0-0.5)^2$ (Fig. 2b), and the sine operator $F(V_0) = 0.5+0.5\sin(2\pi V_0)$ (Fig. 2c) redistribute the ramp. Some authors report considerable success with unary magnification operators [14], however, in our experience a preponderance of unary operators does not help image generation; it clogs image space. The principal danger in using too many unary operators is that through iteration a chain of unary operators can bleed to a constant or, worse yet, if an operator is its own inverse a chain can create an identity operator. In either case, the difficulty is that chains of unary operators fill the expression with dead weight. Thus we favor a very limited set of unary operators

$$A_1 = \{sqr,\ sqrt,\ not,\ parab,\ sin\},$$

and we avoid clogging expressions with unary operators by limiting the number that can appear in an expression.

We now turn to the set of binary primitive functions of arity two. Arithmetic operators are efficient and easy to implement, but because we require values to lie in the unit interval, normalization is often necessary. Thus, under the heading of addition, we include operators

$$avg\text{-}sum(V_0, V_1) = (V_0 + V_1)/2 \text{ (Fig. 3a)}$$

and

$$mod\text{-}sum(V_0, V_1) = (V_0 + V_1) \bmod 1 \text{ (Fig. 3b).}$$

For subtraction, our choice is

$$sub(V_0, V_1) = |V_0 - V_1| \text{ (Fig. 3c).}$$

The *multiply* and the *power* functions borrowed from the standard mathematical library implementations require no discussion. The *min* (Fig. 4a) and *max* operators are straightforward, and although the method is somewhat implementation dependent, the bitwise logical operators can be used to produce fractals (Fig. 4b). The cone over the unit square (Fig. 4c) is included for its radial diffusion characteristics.

Another inspiration for binary primitives comes from using drawing routines as primitives. We adapted three from Maeda's book [9], two of which were based on the vector drawing algorithms

Figure 3: (a) An addition primitive based on averaging; (b) an addition primitive obtained by using the fractional part of the sum; and (c) a subtraction primitive.

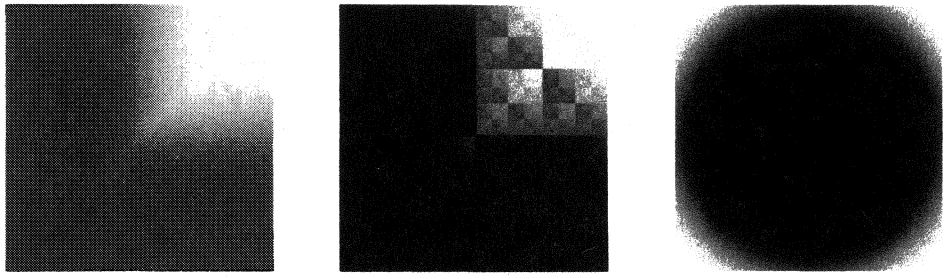

Figure 4: (a) The *min* function; (b) the "fractal" bitwise *and* operator; and (c) the cone over the unit square.

written in his design language. They depend heavily on his use of unusual metrics. We give the results of converting Maeda's primitives [9, page 129 and page 242] to coordinate form.

$$maeda1(V_0, V_1) = \begin{cases} 1 - V_1 & \text{if } V_0 < V_1 \\ V_0 & \text{if } V_0 > V_1 \end{cases} \quad \text{(Fig 5a)}$$

$$maeda2(V_0, V_1) = \begin{cases} 0.5 * (1 + (V_1 - 0.5)^2/(V_0 - 0.5)^2) & \text{if } V_1 < V_0 \\ 0.5 * (1 + (V_0 - 0.5)^2/(V_1 - 0.5)^2) & \text{if } V_1 > V_0 \end{cases} \quad \text{(Fig 5b)}$$

$$maeda3(V_0, V_1) = \begin{cases} 0.5 * (1 + V_1^2/(1 - V_0)^2) & \text{if } V_1 < 1 - V_0 \\ 0.5 * (1 + (1 + (V_0 - 1)/V_1)^2) & \text{if } V_1 > 1 - V_0 \end{cases} \quad \text{(Fig 5c)}$$

Finally, we give our construction for converting *any* continuous function $F(X)$ defined on the interval $[-0.5, 0.5]$ into a binary primitive function. Our goal is to define a function

$$B(V_0, V_1) : [0, 1] \times [0, 1] \longrightarrow [0, 1].$$

By setting $V_i' = V_i - 0.5$, and letting $B(V_0, V_1) = B'(V_0', V_1')$, thanks to simple translation of axes, it suffices to construct $B'(V_0', V_1') : [-0.5, 0.5] \times [-0.5, 0.5] \longrightarrow [0, 1]$. We start with the continuous

function $V_1' = F(V_0')$ defined on $[-0.5, 0.5]$. Parameterizing by c, the family of vertical translations $\{V_1' = F(V_0') + c\}$, foliate the strip $[-0.5, 0.5] \times \mathbb{R}$. Choose $c_{\max}$ and $c_{\min}$ such that over the interval $[-0.5, 0.5]$ the minimum of $V_1' = F(V_0') + c_{\max}$ is 0.5 while the maximum of $V_1' = F(V_0') + c_{\max}$ is -0.5. The reason for doing so is that now

$$c_{\min} \leq V_1' - F(V_0') \leq c_{\max}$$

on the translated unit square, $[-0.5, 0.5] \times [-0.5, 0.5]$. Now choose a "height map"

$$Z : [c_{\min}, c_{\max}] \longrightarrow [0, 1]$$

and set

$$B'(V_0', V_1') = Z(V_1' - F(V_0')).$$

The height map can be used to intensify the primitive's contrast.

EXAMPLE 4.1 Let $F(X) = |X|$. Then $V_1' = |V_0'|$ and $c_{\max} = 1/2$ while $c_{\min} = -1$. Since $Z(c) = \frac{2}{3}(c + 1)$ maps $[-1, 1/2]$ onto $[0, 1]$ we obtain the primitive

$$\mathit{zabs}(V_0, V_1) = 0.666667 * ((V_1 - 0.5 - |V_0 - 0.5|)) + 1) \text{ (Fig. 6a).}$$

EXAMPLE 4.2 Let $F(X) = 0.5 - X^2$. Then $V_1' = 0.5 - (V_0')^2$. Again $c_{\max} = 1/2$ and $c_{\min} = -1$, but this time we use $Z(c) = \frac{2}{3}(1 - c)$ to obtain

$$\mathit{zparab}(V_0, V_1) = 0.666667 * (1.5 - V_1 + (V_0 - 0.5)^2) \text{ (Fig. 6b).}$$

EXAMPLE 4.3 Let $F(X) = 0.5 + \sin(2\pi(x - 0.5))$. Then $V_1' = 0.5 * \sin(2\pi(V_0' - 0.5))$ yields $c_{\max} = 1$ and $c_{\min} = -1$. We intensify the sine curve by setting $Z(c) = 1 - c^2$. This gives

$$\mathit{zsin}(V_0, V_1) = 1 - (V_1 - 0.5 - 0.5 * \sin(2\pi(V_0 - 1)))^2 \text{ (Fig. 6c).}$$

This completes the set of binary primitives that will be used in our next section. They are

A_2 = {*multiply, subtract, avg-add, mod-add, power, min, max, and, cone, maeda1, maeda2, maeda3, zabs, zprb, zsin*}.

5. Coevolutionary Images

The primitives that were described in the previous section have been incorporated into a coevolutionary system. Details will appear elsewhere [5], so we give only a cursory description in order to provide some context for our examples. The rationale for making the system coevolutionary is to test ideas for automating the user-guided fitness by aesthetics technique for evolving images that are defined by expressions. The coevolutionary model we will consider is that of a predator-prey model using hosts and parasites. The purpose of parasites is to digitally filter the image in order to detect structures that look significantly different after filtering, and thus whose spatial organization might suggest that the image is visually interesting.

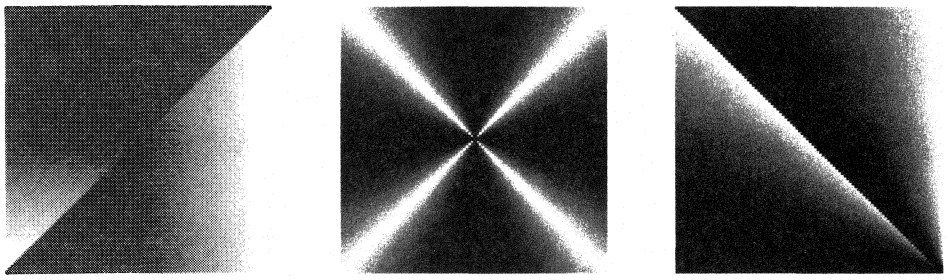

Figure 5: Vector graphic images designed by Maeda which we converted to coordinate form for use as binary primitives: (a) *maeda1*, (b) *maeda2*, (c) *maeda3*.

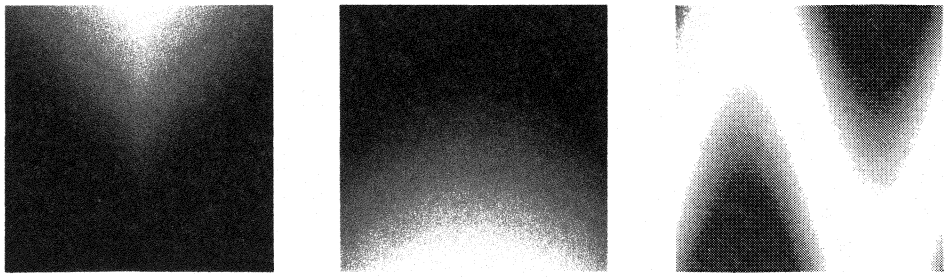

Figure 6: Conversions of (a) an absolute value function, (b) a parabola, and (c) a sine curve to binary primitives using the new construction.

The expressions, or rather the phenotypes of the expressions, serve as hosts to parasites. Parasites are 3×3 digital filters defined by scaling a 3×3 array of integers, with each entry in the interval $[-15, 15]$, by the reciprocal of the absolute value of the array sum. For convenience, we assume the resolution of the hosts is 100×100. At a fixed number of sites on each host, parasites are attached to the host. The host-parasite interaction is quantified by convolving the digital filter with a small 10×10 patch of the host. The patch is therefore simply a neighborhood of the location where the parasite is attached. In general, the parasite is successful at preying upon the host if the convolved image matches the underlying patch to within a specified tolerance, while the host repels the parasite if the convolved patch is significantly different from the underlying patch (thereby exposing the parasite). To quantify this, the comparison is made on a pixel by pixel basis within the patch so that the host is assigned a fitness between zero and one hundred. Of course, when a host has multiple parasites, each attached at a different site, the host fitness is the fitness averaged over all parasites.

The host population evolves in the usual way: Randomly selected pairs of hosts chosen from the pool of fittest hosts produce progeny by swapping subexpressions. Mutation of the host progeny occurs by mutating the building blocks within the progeny genomes on a primitive by primitive basis with arity conserved. We devised our own artificial genetics for parasites based on cloning and mutation. More precisely, we clone (*i.e.* copy) parasites selected from the pool of fittest parasites to replace the weakest parasites, preserve only the very best parasites (elitism), and then subject all the remaining parasites to a suite of mutation operators which, for example, could cause rows or columns in their arrays to be swapped, replace randomly selected array entries, etc. The motivation

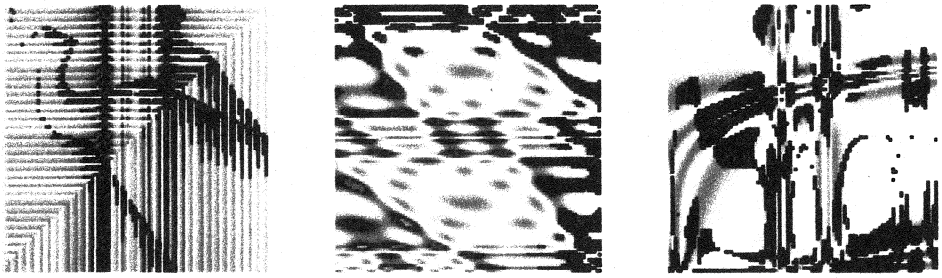

Figure 7: Three images from one run of the coevolutionary simulation using the building block primitives developed in the text. Their diversity is striking. Shown left to right are the most fit images after 5500, 6500, and 7000 time steps respectively. A maximum of fifty primitives is allowed per image.

Figure 8: Images from three separate coevolutionary runs. The number of time steps is limited to 1500, however the upper bound on the number of primitives allowed per image has been increased to one hundred.

for specifying computational aesthetics using our set-up is that when the hosts repel the parasites they are alerting our digital filters — our eyes — that there might be something visually interesting taking place at the parasite's locations. Since parasites attack the hosts *locally*, while hosts, whose only counter measure is new genotypes whose primitives are defined on the entire unit square, can only respond *globally*, evolutionary pressure exerted by the parasites "chases" the host population phenotypes into new regions of image space. The sample coevolved hosts of Figure 7 were coevolved starting from small random populations of hosts and parasites using very small host genomes. Thus they represent images that evolved from the primordial ooze. More typically an evolutionary run uses thirty hosts whose genome expressions each consist of approximately 75 primitives. Each host has three parasites attached, and coevolution lasts for 1500 generations with the most fit host being culled every 200 generations. Examples obtained under these conditions are shown in Figure 8. We are in the early stages of investigating what kind of imagery will result from our coevolutionary setup. Further experimentation needs to be done before drawing any meaningful conclusions about the limitations of coevolved images based on computational aesthetics.

References

[1] Shumeet Baluja, Dean Pommerleau & Todd Jochem, Towards automated artificial evolution for computer-generated images, *Connection Science*, Volume 6, Numbers 2 & 3, 1994, 325–354.

[2] Richard Dawkins, The evolution of evolvability, *Artificial Life*, Christopher Langton (ed.), Addison Wesley, Reading, MA, 1989, 201–220.

[3] Gary Greenfield, New directions for evolving expressions, *Bridges: Mathematical Connections in Art, Music, and Science; Conference Proceedings 1998* (ed. R. Sarhangi), Gilliland Printing, 1998, 29–36.

[4] Gary Greenfield, On understanding the search problem for image spaces, *Bridges: Mathematical Connections in Art, Music, and Science; Conference Proceedings 1999* (ed. R. Sarhangi), Gilliland Printing, 1999, 41–54.

[5] Gary Greenfield, Art and artificial life — a coevolutionary approach, *Artificial Life VII*, to appear.

[6] Danny Hillis, Co-evolving parasites improves simulated evolution as an optimization procedure, *Artificial Life II*, C. Langton et al (eds.), Addison-Wesley, Reading, MA, 1991, 313–324.

[7] John Koza, *Genetic Programming III : Darwinian Invention and Problem Solving*, Morgan Kaufmann, San Francisco, CA, 1999.

[8] Aladin Ibrahim, GenShade, *Ph.D. Dissertation*, Texas A&M University, 1998.

[9] John Maeda, *Design by Numbers*, MIT Press, Cambridge, MA, 1999.

[10] F. Kenton Musgrave, *personal communication.*

[11] Steven Rooke, *personal communication.*

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

[13] Tatsuo Unemi, sbart, *User's manual.*

[14] Miichael Witbrock & Scott Neil-Reilly, Evolving genetic art, in *Evolutionary Design by Computers*, P. Bentley (ed.), Morgan Kaufmann, San Francisco, CA, 1999, 251–259.

[15] Ishihama Yoshiaki, `http://www.bekkoame.ne.jp/~ishmn/gallery/`.