BRIDGES
Mathematical Connections
in Art, Music, and Science

On Understanding the Search Problem for Image Spaces

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 the visual imagery that can be found in a neighborhood of just one image selected from a visual image space. The image space we shall explore is a variant of the space of "expressions" as formulated by Karl Sims. Sims' expressions are rooted trees that can be manipulated and searched using the art-by-choice technique known as "evolving expressions." We present a search methodology and give examples from the catalog of imagery we obtained in searching a neighborhood of a fixed image which contained in excess of 10^{20} instances of possible images. We draw conclusions about what this portends for future work on search engines developed for exploring artistically non-linear spaces.

1. Introduction

One approach to harnessing the computational resources of the computer for assisting in generating images for aesthetic purposes is to work in a setting where images are described completely in mathematical terms. When each possible image has a brief mathematical description and there are trillions of mathematical descriptions available, the nature of the creative process shifts from *constructing* a desired image to *searching* for a desired image. The searching method that has received the most attention is known as the art-by-choice paradigm. There are two obstacles to overcome when using this method. First, one must develop tools for sifting through the images. Second, one must have some understanding of the relationship between similar mathematical descriptions and their resulting images. In this paper we focus primarily on the second obstacle. By working under controlled conditions, we attempt to learn about the visual diversity that can result from searching images that are closely related mathematically. This leads us to speculate about the future of semi-automatically generating images.

Art-by-choice methods rely on enhanced computer processing capabilities to search for visual imagery based on user-guided aesthetic principles. As these methods become more well-known and widespread [7] [8] [4], it becomes critical to learn more about the limitations and difficulties of the central concept underlying this process – exploring regions of non-linear spaces whose "points" are really mathematical descriptions of visual images. In investigating this unique form of computer generated abstract art, with its close ties to algorithmic art and image processing, we shall attempt to obtain some evidence about the size and complexity of the typical image space that must be searched, and to consider how our first generation tools for searching within such spaces are able to cope with ambient image coherence and image variation. More precisely, we shall restrict our search to roughly 4×10^{19} images which form a neighborhood of images all of which are "genetically related" to one fixed base image. We wish to gauge the extent of the variation within the neighborhood

and to obtain information about how difficult it is to locate visually interesting (*i.e.*, aesthetically valuable) variations. We shall describe in the sequel an experiment conducted over the course of a four month period on an art-by-choice system of our own design [5] which we hope sheds some light on these matters.

2. Background

We define an *image space* to be a mathematical object consisting of "points" with the property that each point can be associated with a (two-dimensional) image that can be viewed on a computer output device such as a color monitor. Thus "points" can correspond either to algorithms for generating such images or "points" can correspond to (mathematical) data for input into software systems for generating such images. We distinguish between an image space that is (artistically) *linear*, meaning that nearby points generate closely related images, and one that is (artistically) *non-linear*, meaning nearby points may generate wildly different images. Clearly, nonlinear spaces are visually richer, but at first glance it would appear that controlling or guiding searches within such spaces is problematic.

Appealing to a biologically inspired metaphor, many practitioners of the art-by-choice search paradigm view the points of an image space as *genomes*, whence the search for images within image space is analogous to following an evolutionary trajectory in image space. This idea seems to have originated with Dawkins [2], but came to prominence in the computer art world through the efforts of Latham [9] and Sims [8]. In addition to our own attempts that we shall describe in detail below, we should mention that Rooke (as described in Voss [11]) and Ibrahim [6] have also exploited this approach in their work.

There is an important difference in the way Latham and Sims designed points/genomes for their image spaces. For Latham, a genome is a vector — an array of drawing parameters — in a twenty-four dimensional real vector space. Though Latham's resulting image space is highly non-linear in the sense described above, the Latham-Todd solution to the search dilemma in this space is "steering." A record of previous movement within the space helps predict distance and direction for future exploration. For Sims, a genome is a mathematical expression — an expression tree — which can be *interpreted* to yield the visual image. Regardless of how the genome is defined, the key contribution, due to Dawkins, is to make evolution a user-directed process by selecting a small population of genomes, mutating, mating, or otherwise evolving the genomes within the population according to an evolutionary algorithm, and then displaying the new population of images so that the image development cycle can continue.

Since Sims' expressions are rooted trees, Sims' image space is the infinite-dimensional space of rooted trees. The fact that Sims' image space is more non-linear than Latham's is neatly pointed out by Margaret Boden, who describes what is taking place in the context of agents rather than genomes [1]:

> [Sims] results are always "viable," in the sense that the newly transformed code will generate some visible image or other. But the process is utterly undisciplined. Although it could be used to help graphic designers come up with images they would never have

thought of themselves, it cannot be used to explore or refine an image space in a systematic way. However, that is possible if the mutating agents are allowed to alter only the superficial parameters of the code. Significantly, these less powerful agents are preferred by [Latham].

Our objective is to make precise notion of "nearby" points in a Sims' style system by fixing a base genome and considering all possible mutations that *preserve the rooted tree structure.* In the next section, we shall introduce our Sims' style system, which is a hybrid model that strives to make exploration of an image space of expressions a more disciplined process.

3. Our Mathematical Model for Expressions

In this section we give a formal model for expressions as rooted trees, as well as our algorithm for converting expressions into images. We use the words *expression*, *rooted tree*, *genome*, and *point* (of image space) interchangeably. Roughly, the idea is to define a small number of mathematical entities that will constitute a set of visual primitives which can be combined in a seemingly endless number of ways to form the space of images. By themselves, the primitives appear rather dull. There are several color ramps including linear, curvilinear, and radial ones. There are also some simple two-dimensional patterns, one or two of which look "fractal." The subtle feature of our method is that at every stage during the process of building up an image from primitives we can gently perturb the description by invoking "drift coefficients." These numerical values induce the color shifts, distortions, magnifications, and disruptions that provide additional detail and variation. From the mathematical description that we assemble, the *expression*, we can perform a computation for each pixel, the *evaluation*, to generate the visual image. A bottleneck arises from the large number of evaluations that have to be carried out to generate a full size image.

To prepare for the formal description, we let $\mathcal{V} = \{u, v, w, c, e\}$ be a set of *variables* which may appear in the leaf nodes of our rooted trees. When an image is generated from an expression by evaluating the expression, the coordinate pair (u, v) will correspond to a point in the unit square. The leaf variables c and w will be used to indicate when the leaf nodes will only be accessed for coefficients (see below), hence they will always assume the value zero when the expression is evaluated. The variable e is reserved for "indirection" which will be explained later. At internal nodes of our rooted tree, we allow functions in either one or two variables accordingly as the node has either one or two children. The set of admissible functions comprise our set of primitive functions. For convenience and efficiency all functional inputs and outputs are required to lie in the unit interval I. Thus, we must construct a set of (normalized) function primitives $\mathcal{F}$, where each primitive is defined on either the unit interval I or on the unit square $I \times I$. Finally, we add one more feature which vastly enlarges the number of points in our image space while trying to maintain a coherency between images obtained from nearby image points within the space. The idea is to attach drift coefficients to each node. The coefficients are a drift *multiplier*, a, and a drift *constant*, b, which are used to define an affine transformation local to each node. We are ready for a formal definition of an expression.

DEFINITION 3.1. An *expression* is defined recursively to be a *node* $(a\ b\ G)$, where a and b are nonnegative real constants and either $G \in \mathcal{V}$ or $G \in \mathcal{F}$. If $G \in \mathcal{F}$, then each argument of G must

again be a node.

EXAMPLE 3.1. Assume $\mathcal{F}$ contains the *normalized* sine and cosine functions, `nsin`: $I \longrightarrow I$ defined by $\texttt{nsin}(x) = 0.5+0.5*\sin(2\pi x)$ and `ncos`: $I \longrightarrow I$ defined by $\texttt{ncos}(x) = 0.5+0.5*\cos(2\pi x)$. Assume further that $\mathcal{F}$ contains the square root function `nsqt` and the function $\texttt{nmin}(x, y) = \min(x, y)$. Then, omitting superfluous parentheses, according to our definition a valid expression is:

```
0.05 0.38 nmin(0.54 0.09 nsin(0.42 0.52 u), 0.03 0.69
ncos(0.22 0.90 nsqt(0.04 0.63 v))).
```

DEFINITION 3.2. An expression $(a\ b\ G)$ is *evaluated* at a point $(x, y) \in I \times I$ using the recursive evaluation rule $\mathcal{E}$: $I \times I \longrightarrow I$ given by:

$$\mathcal{E}(a\ b\ G) = a * \mathcal{E}(G) + b \pmod 1,$$

where, if $G \in \mathcal{V}$, then

$$\mathcal{E}(G) = \begin{cases} 0 & \text{if } G = c, \text{ or } G = w \\ x & \text{if } G = u \\ y & \text{if } G = v \end{cases}$$

and, if $G \in \mathcal{F}$ is a function in d variables whose arguments are the nodes $n_1, \ldots, n_d$, then

$$\mathcal{E}(G) = G(\mathcal{E}(n_1), \ldots, \mathcal{E}(n_d)).$$

Of course, here we are only allowing d to be one or two.

While the formal model explains how the imaging algorithm is implemented, its details may obscure the relatively straight forward computations that yield images. An example may help to clarify the algorithm.

EXAMPLE 3.2. To visualize or "render" the expression

```
0.05 0.38 nmin(0.54 0.09 nsin(0.92 0.42 u), 0.04 0.63 v)
```

as a 100×100 pixel image, we convert the pixel address (m, n) to the point with coordinates $(x, y) \in I \times I$ where $x = m/100$ and $y = n/100$ and then evaluate the expression at (x, y). Therefore at the pixel with address, say $(85, 30)$, evaluation would proceed in stages producing intermediate values E_1, E_2, E_3 and final value E as follows:

$$\begin{aligned}
E_1 &= \mathcal{E}(0.04\ 0.63\ \texttt{v}) = 0.04 \cdot 0.30 + 0.63 = 0.012 \pmod 1, \\
E_2 &= \mathcal{E}(0.92\ 0.42\ \texttt{u}) = 0.92 \cdot 0.85 + 0.42 = 0.202 \pmod 1, \\
E_3 &= \mathcal{E}(0.54\ 0.09\ \texttt{nsin}(E_2)) = 0.54 \cdot 0.977 + 0.09 = 0.617 \pmod 1, \\
E &= \mathcal{E}(0.05\ 0.38\ \texttt{nmin}(E_3, E_1)) = 0.05 \cdot 0.617 + 0.38 = 0.411 \pmod 1.
\end{aligned}$$

Using a look-up table, the final value E is converted to a color which is then assigned to the pixel. Though images appear here in gray scale giving some sense of image contrast rather than image color, in fact, the same one hundred and eighty color palette — a palette closely resembling the ordinary color spectrum — was used for all images generated.

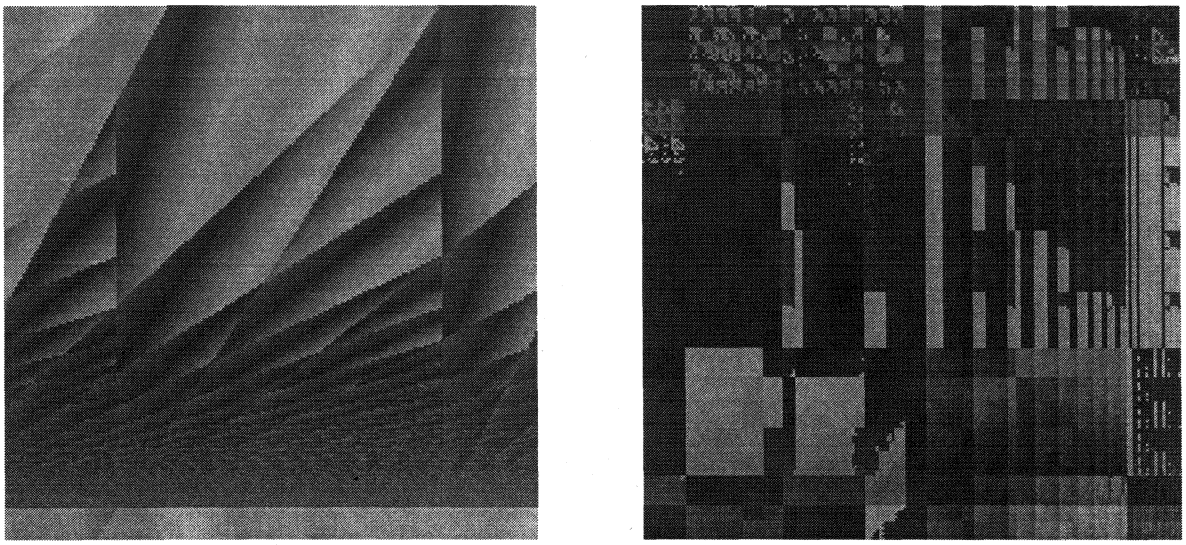

Figure 1: Background image (left) and base image (right).

The previous example shows how the evaluation map $\mathcal{E}$ is used to manage the pointwise computation of an expression after inputs are assigned to those variables which are present at the leaf nodes of the expression. Notice how the presence of drift coefficients affects the node evaluation algorithm: after passing through the function primitive, but *before* being used as an input to the parent node, the intermediate result is perturbed by the linear equation determined using the node's drift coefficients and then only the *fractional* part of that result is passed up to the parent node.

Expressions in which the variable e occurs at a leaf cause evaluations to be redirected to a user-specified "background" expression, say H, *when one is present.* If there is no background image present, then e is treated like the leaf variables c and w. Formally, at the point $(x, y) \in I \times I$,

$$\mathcal{E}(e) = \begin{cases} \mathcal{E}(H) & \text{if "background" expression is present} \\ 0 & \text{otherwise.} \end{cases}$$

We think of occurrences of the variable e in an expression G as providing "sockets" for feeding the background image defined by H into the base image G. Figure 1 shows the base image and background image we shall use throughout this paper, while Figure 2 shows the result of feeding the background image into the base image through the available e sockets, yielding an image which we call *Emergent Blue.*

As an infix expression, the base image that we selected, which was originally "evolved" in 1992, is given by,

```
0.85 0.20 nvee(0.58 0.06 nmod(0.85 0.20 nvee(0.58 0.06
nmod(0.79 0.66 nabs(0.08 0.60 nsin(0.94 0.08 c))  0.91 0.98
nand(1.00 0.46 nsqt(0.34 0.17 v)  0.32 0.63 u))  0.90 0.09 npwr
(0.80 0.10 nmod(0.55 0.53 e  0.19 0.92 nnot(0.59 0.87 u))
0.14 0.33 nsin(0.80 0.50 e)))  0.08 0.22 nabs(1.00 0.54 e))
0.90 0.09 nmin(0.80 0.10 nmod(0.62 0.96 v  0.61 0.54 v)  0.02 0.19 c))
```

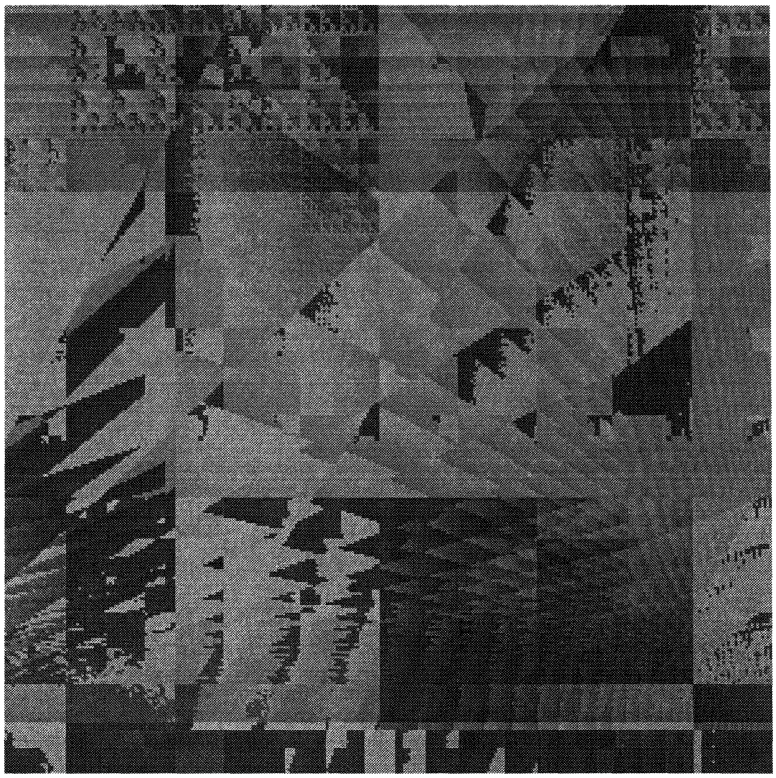

Figure 2: *Emergent Blue*, obtained by "compositing" the background and base images of Figure 1.

4. Nearby Points of Image Space

To explore our image space, we must start with the image we have selected. Its mathematical description will only be altered in ways such that subsequent images will have similar mathematical descriptions. This is done by replacing existing primitives by other available primitives and by adjusting the way intermediate images are combined by regulating the drift coefficients. There is a biological analogy. If we think of the expression we wish to start with as a genome then it becomes the progenitor of a species and all subsequent images must be genetically similar because genetics imposes limits on the kind of variation that is possible within the genetic makeup of the progenitor's descendants.

The rooted tree for the base image we are using is shown in Figure 3. In Figure 3 the normalized primitives of $\mathcal{F}$ which were given in infix form in the previous section are now referenced using their *symbolic* equivalents, and the drift coefficients have been temporarily suppressed. The tables below establish the correspondence between infix and symbolic codes, and also provide abbreviated descriptions of the primitives. The nine primitive functions of one variable appear in Table 1 and and the nine primitive functions of two variables appear in Table 2. As we noted previously, these (normalized) primitive functions are designed and crafted for their two-dimensional characteristics when imaged on the unit square.

The base image for *Emergent Blue* has twenty-five nodes apportioned as follows: ten nodes are leaves containing variables, six nodes contain one variable primitive functions, and nine nodes

Infix	Code	Description
nsin	S	$f(u) = 0.5 + 0.5 * \sin(2 * 3.1415927 * u)$
ncos	C	$f(u) = 0.5 + 0.5 * \cos(2 * 3.1415927 * u)$
nexp	E	$f(u) = \exp(u - 1.0)$
nlog	L	$f(u) = \log(u + 1.0)/0.69314718$
nabs	A	$f(u) = \lvert 2 * u - 1 \rvert$
nsqt	T	$f(u) = \sqrt{u}$
nsqr	R	$f(u) = 4.0 * (u - 0.5) * (u - 0.5)$
ncub	U	$f(u) = 4.0 * (u - 0.5) * (u - 0.5) * (u - 0.5) + 0.5$
nnot	N	$f(u) = 1.0 - u$

Table 1: Primitive functions of one variable.

Infix	Code	Description
nadd	+	$f(u, v) = 0.5 * (u + v)$
nmul	*	$f(u, v) = u * v$
nmod	%	$f(u, v)$ is the remainder when u is divided by v
nmin	_	$f(u, v) = \min(u, v)$
nmax	-	$f(u, v) = \max(u, v)$
npwr	^	$f(u, v) = u^v$
nand	&	$f(u, v)$ is the bitwise logical and of u and v
nvee	|	$f(u, v)$ is the bitwise logical or of u and v
ncir	0	$f(u, v) = 2.0 * ((u - 0.5) * (u - 0.5) + (v - 0.5) * (v - 0.5))$

Table 2: Primitive functions of two variables.

contain two variable primitive functions. Since the set of primitive functions contains nine one variable primitives and nine two variable primitives, and for two-dimensional images there are four distinct variables (recall both c and w will always be evaluated as zero and there is a background image present for use with e), the number of "nearby" points we wish to explore is nominally

$$9^9 \times 6^9 \times 10^4 \approx 3.9 \times 10^{19}.$$

Further, since each node has two drift coefficients, a multiplier a, with $0 \leq a \leq 2$, and a constant b, with $0 \leq b \leq 1$, both of which we resolve to within 0.01, each "point" via drift has 25^{300} neighbors within a *drift* neighborhood! Of course, there are not really that many points in the neighborhood of our base image. There is a certain amount of redundancy in the genome itself as well as dormant areas within the genome (the biological analogy of introns and extrons comes to mind), and as artificial life experiments using this setup have revealed, many of the mutations are superficial [4]. Moreover, since (1) the base image was originally evolved with each multiplier drift coefficient a in the unit interval, (2) the software we are using to explore image space severely restricts the amount of drift that can accumulate at internal nodes, and (3) drift will have no effect at latent sites of the genome or negligible effect at many active sites (for example, this can occur when the multiplier a is close to zero and therefore the constant b dominates), it would be more realistic to assume that the number of distinct "nearby" points in the neighborhood of our base image is closer to 10^{20} rather than $3.9 \times 10^{19} \times 25^{300} \approx 9.4 \times 10^{438}$. One should recognize, however, that our image space is infinite dimensional and we are discussing the nearby points of only one fixed base point.

5. Searching Nearby Points

The system we described in [5] for following evolutionary trajectories within image space was used here in a more restrictive fashion in order to make sure that we examine only those points that are nearby the base image. The overall objective of the system is to specify algorithms for modifying expressions. This establishes the genetic "rules" which determine how expressions can evolve.

Because our goal for this project was to explore only a neighborhood of one point and not all of image space, we were only able to use a few of the algorithms that were available to evolve genomes — those algorithms that preserve the rooted tree structure (as shown in Figure 3) of the base image. This restricted suite of algorithms "operate" on existing expressions in the neighborhood of the base point. The names we gave to the restricted suite of algorithms together with their descriptions are listed below.

- *DIFFER*, which substituted for a primitive function at *one* internal node, a node that was randomly chosen in the expression;
- *VARY*, which replaced variables in approximately half the *leaves* in the expression;
- *PERTURB*, which adjusted the drift coefficients at each node by an amount that was weighted inversely with respect to the node's depth, so that coefficients near the root drifted minutely, while coefficients close to leaves drifted up to a maximum value controlled by a global parameter;

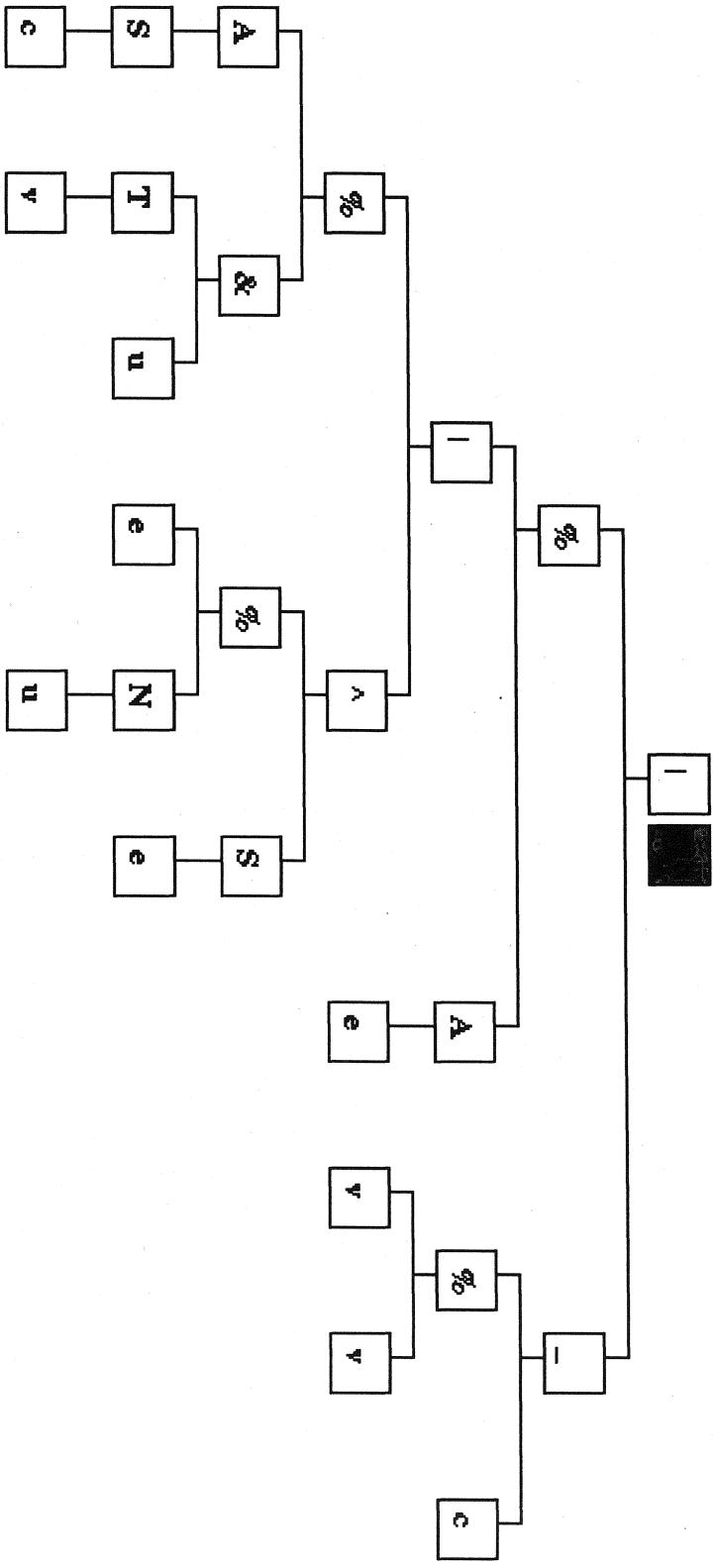

Figure 3: The rooted tree for the base image shown in Figure 1. (See text for details.)

- *SLIDE*, which shifted the drift coefficient b at the *root* of the tree thereby resulting in a color shift of the image;
- *FEATHER*, which re-randomized all the drift coefficients at the leaves. (It turned out that this was the least effective algorithm.);
- *CROSS*, which performed a *uniform* crossover between two existing points in the neighborhood of the base point to yield a third point in the neighborhood of the base point which had approximately one-half of its nodes identical to the first point and the other half of its nodes identical to the second point.

In simpler terms, we have tried to describe a user interface that displays the visual renderings of a small group of expressions but whose user controls can only affect the mathematical descriptions of those expressions. Thus, exploration of nearby images in the neighborhood of an image is indirect, because it is accomplished by altering the contents of the expressions as given in Figure 3, not by directly altering the images themselves. Next, we give a more explicit account of how we actually carried out the exploration.

Since every time an expression modification algorithm is invoked all the drift coefficients in all the nodes in the expression it produces are subject to drift of up to a maximum amount of 0.05, the working strategy was to rely on the *DIFFER* algorithm to effect a sequence of one-node-at-a-time mutations to an expression already available in the population and then to periodically apply the *VARY* algorithm to the newly evolving expression in order to, in the language of genetic algorithms, escape local minima. At intervals, back crossing with ancestors or other expressions already in the population using the *CROSS* algorithm was successfully employed.

6. Examples from the Catalog of Nearby Points

In sessions spent exploring image space in the neighborhood of our base point, we always began each session starting with our base point. Typically, in two hours, one or two aesthetically worthwhile images could be mined. We would estimate that in an hour no more than a thousand images could be examined under the search paradigm we used. Various researchers are currently investigating ways to seed searches, set search goals, or process larger populations of nearby points. This is an active field of research with many ideas waiting to be tried.

The examples from the catalog of images we obtined (Figures 4–11) represent a sample of twenty-four culled from the eighty that were archived over the four month exploration period. These eighty images were labeled using a chronological ordering as M1 through M80. In more closely examining the primitive functions appearing in the eighty nearby images we obtained, we were able to observe that there was a typical variation from the base genome in only five to seven internal nodes, but that the number of changes in leaf nodes *i.e., variables* was erratic. This is easily explained by the fact that the *DIFFER* algorithm substitutes for one internal node, while the *VARY* algorithm replaces several of the leaves. The largest drift an internal multiplier a coefficient reached was 1.11, but because of wholesale replacement of leaves, multipliers as high as 1.85 could be found in them.

Figure 4: Images M4, M5, and M9.

Figure 5: Images M10, M13, and M18.

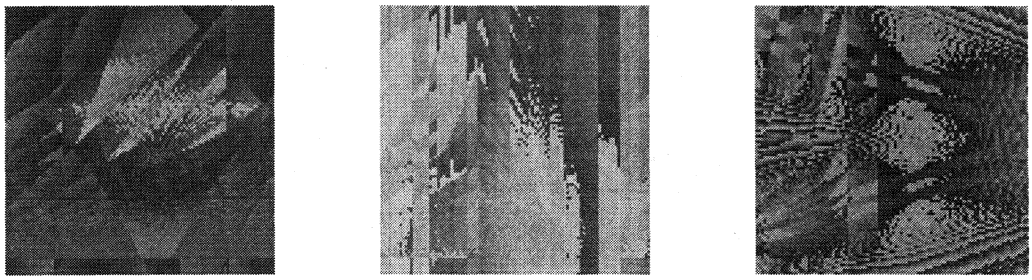

Figure 6: Images M22, M25, and M26.

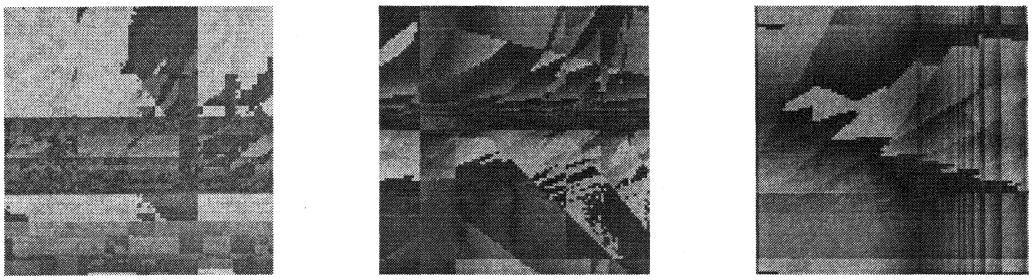

Figure 7: Images M30, M35, and M37.

Figure 8: Images M43, M45, and M48.

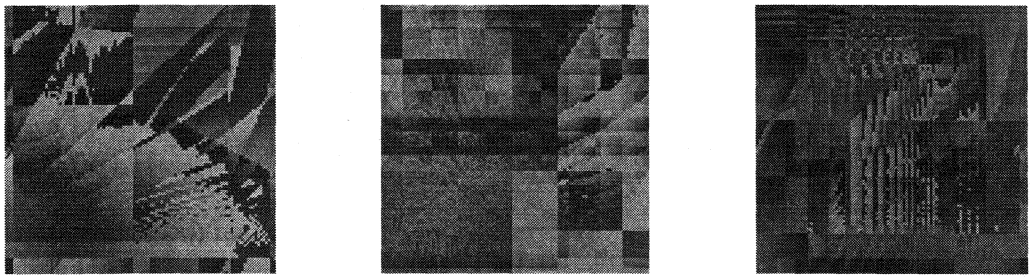

Figure 9: Images M52, M53, and M57.

Figure 10: Images M61, M68, and M72.

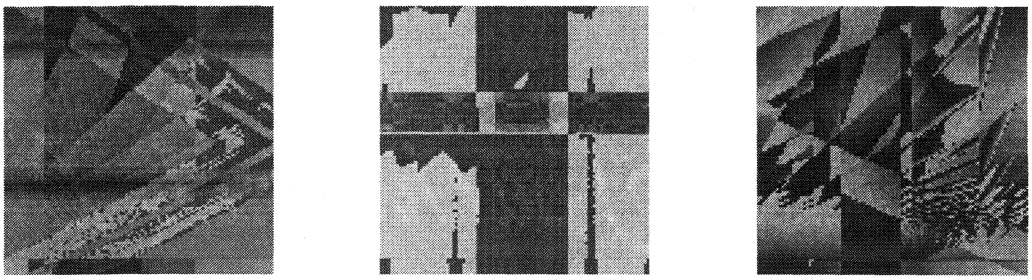

Figure 11: Images M74, M77, and M79.

Even in contrasting gray-scale mode, the diversity of images found from nearby points speaks for itself. It strikingly conveys the nature of the non-linearity of the image space that we have alluded to. What conclusion do we draw from our experiment? Given the amount of time spent searching the neighborhood and the diversity it produced, and remembering that this was a neighborhood of but one fixed point in an infinite dimensional space, we believe that future work must be directed towards increasing the throughput of images that can be examined.

One approach is to consider using software agents, perhaps in a co-evolutionary setting, to help sift through the abundance of images in an image space. The elementary observation is that a software agent could sample an expression's pixel values *numerically* without ever having to generate a visual image of the expression. Possibly the base image could be used to spawn "fingerprint" agents which could preserve some of the aesthetic properties of the base image. Thus, future descendants of the base image would be inspected by the fingerprint agents to make sure that the desired visual continuity has been retained.

We are currently investigating the feasibility of dedicating large numbers of extremely feeble agents to this finger printing task by embodying the agents as image processing filters. We will evaluate the filters by attaching them to the images at fixed locations and measuring how well the images area able to repel them. As an image evolves, it is subjected to more filters. Co-evolution could vary these filters and allow them to spread across the image. We think of these filters as parasites so that a successfully evolved expression, an aesthetically valued expression in the neighborhood of the base image, is one that will be able to repel many different kinds of parasites.

References

[1] Margaret A. Boden, Agents and creativity, *Communications of the ACM*, **37** No. 7 (July, 1994), 117–121.

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

[3] Gary Greenfield, An algorithmic palette tool, UR Technical Report TR-94-02, 1994.

[4] Gary Greenfield, Graphical evolution experiments in artificial life, UR Technical Report, TR-93-01, 1993.

[5] 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.

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

[7] Lev Manovich, The engineering of vision and the aesthetics of computer art, *Computer Graphics*, **28** No. 4 (November, 1994) 259–263.

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

[9] Steven Todd & William Latham, *Evolutionary Art and Computers*, Academic Press, San Diego, CA 1992.

[10] Steven Todd & William Latham, Mutator, a subjective interface for evolution of computer sculpture, IBM UKSC Report 248, 1991.

[11] David Voss, Sex is best, *WIRED*, December, 1995, 156–157.