Procedural Texture Evolution Using Multiobjective Optimization

Brian Ross and Han Zhu

Technical Report # CS-02-18
July 2002

Submitted for publication.

Procedural Texture Evolution Using Multiobjective Optimization

Brian J. ROSS and Han ZHU

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

Received July 31, 2002

Abstract This paper investigates the application of evolutionary multiobjective optimization to two-dimensional procedural texture evolution. Genetic programming is used to evolve procedural texture formulae. Earlier work used multiple feature tests during fitness evaluation to rate how closely a candidate texture matches visual characteristics of a target texture image. These feature test scores were combined into an overall fitness score using a weighted sum. This paper extends this research by replacing the weighted sum with a Pareto ranking scheme, which preserves the independence of feature tests during fitness evaluation. Three experiments were performed: a pure Pareto ranking scheme, and two Pareto experiments enhanced with population divergence strategies. One divergence strategy scores individuals using their nearest-neighbour distance in feature-space. Another scheme uses a normalized, ranked measurement of nearest neighbour distance. A result of this work is that acceptable textures can be evolved much more efficiently with MOP evolution than compared to the weighted sum approach done earlier. The ability to create a diverse selection of Pareto solutions is advantageous in this problem domain, since the acceptability of a final texture is ultimately a subjective, aesthetic decision by the user.

Keywords Procedural Textures, Computer Graphics, Genetic Programming.

§1 INTRODUCTION

Procedural textures are used in computer graphics to produce a variety of photo-realistic effects, such as stone, wood, clouds, and other natural and unnatural phenomena [17, 1]. The successful engineering of new procedural textures that display desired visual effects requires extensive mathematical insight and analytical modelling. This means that procedural texture design is technically difficult and unintuitive for most users. Consequently, a number of texture exploration tools based on evolutionary computation have been devised. Evolutionary computation can often excel in efficiently exploring a large search space. Most of these texture evolution systems are interactive, and rely on the user to be the judge of texture fitness and suitability to guide the direction of evolution. A few texture evolution systems used unsupervised evolution [7, 18]. These systems replace the user with a conventional fitness evaluator, in which candidate textures are scored via a number of image analyses routines, in an attempt to find a match with the features from a "target" texture image. Although rudimentary in their scope, a combination of feature tests usually gives satisfactory results. The reconciliation of independent feature tests by the fitness function, however, is difficult to do effectively. Furthermore, effective evolution in these systems also requires extensive computational effort.

This paper addresses the texture evolution problem by considering it to be an instance of a *multiobjective optimization problem* or MOP [2, 16]. MOP is characterized by a set of distinct features, in which different combinations of these features result in optimized results. We use Goldberg's Pareto fitness ranking strategy, which considers the fitness space to be a stratification of ranks [3]. The top rank represents a set of solutions that is not dominated or improved upon by any other solution in the population. The value of Pareto optimization in the texture evolution problem is that it circumvents the need to reconcile independent feature tests. Each feature test is retained as an independent dimension of a vidual characteristic of a texture, and no single feature will dominate any others during the run. Along with a pure Pareto ranking scheme, a couple of diversity promoting strategies were used. The end result is that textures are evolved with the Pareto fitness strategy that are competitive with the original Gentropy results, but are obtained significantly more efficiently.

Section 2.2 discusses the use of evolutionary computation in texture generation, and overviews the texture feature tests used in our experiments. Section 3 discusses multiobjective optimization, and the Pareto and diversification algo-

rgb(mod(turbflow(Y,X,X), sin(X)),
lum(marble(0.94, -0.78, (-0.46,0.50,-0.63))),
turb(chn(COLGRAD), cos(-0.24)))

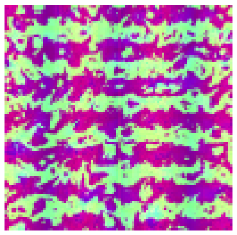


Fig. 1 Formula and texture

rithms. Details about the genetic programming experiments are given in Section 4. Some results are shown in Section 5. A discussion and directions for future research conclude the paper in Section 6.

§2 Automatic Texture Evolution

2.1 Procedural textures

Textures help create photorealism in computer graphics [17, 1]. The most common textures are procedural textures and bitmapped textures. Procedural textures are computed via algorithms and mathematical formulae. They take as input a coordinate or pixel location in 2D- or 3D-space, and compute a corresponding pixel colour, in terms of an RGB (red, green, blue) triple. Bitmap textures wrap or tile a bitmap image onto an object surface. Procedural textures have a number of advantages over bitmap textures. Procedural textures can be applied seamlessly over 3D objects, whereas bitmaps tend to produce seams, stretch marks, and tiling artifacts. Procedural textures faithfully simulate a variety of natural phenomenon, including stone, cloud, wood, and landscaping effects. Their mathematical nature makes them extremely robust, as there are an infinite variety of equations conceivable, all yielding new, unique effects. Due to their mathematical nature, it is extremely difficult to write a procedural texture formula from scratch that will produce an arbitrary desired graphical effect. With a bitmap texture, however, one merely needs to scan an image of the desired texture effect. Most applications therefore store a library of parameterized procedural textures for known effects, which the user can tailor as desired.

Figure 1 shows a texture formula and its corresponding texture.

2.2 Texture evolution

The use of evolutionary computation is well established as a means for searching the infinite space of procedural texture formulae [10]. Genetic algorithms are well-adapted to this task, as the abstract concept of chromosomal "building block" seems to correlate well with the visual characteristics found within terms of a texture formula. Texture formulae can be implemented as tree–based programs, which can then be subjected to crossover and mutation operators as used in genetic programming. Most texture evolution systems are interactive, and rely on a human being to perform fitness selection on candidate textures. This overcomes the complexity of automatic texture evaluation. A combination of user-directed selection, mutation and refinement lets these systems converge on a texture formula that satisfies some aesthetic requirements.

A few systems, such as Genshade [7] and Gentropy [18], perform automatic texture analyses. A fitness function tests how closely a candidate texture shares colours, patterns, and other features with a target texture. These feature tests perform fairly basic image analyses, since it is a practical necessity that fast and efficient tests be used in an evolutionary environment. The most accurate analyses of images would use sophisticated computer vision technology, which is too slow to be practical, and an open research problem anyway.

The Genshade system evolves Renderman shaders [7]. Chromosomes take the form of directed acyclic graphs, which are akin to the S-expression trees used in conventional genetic programming [8]. Nodes of these graphs are references to Renderman shader primitives, which are high-level texture generation functions [15]. Genshade applies lumination, colour, and wavelet analyses to candidate textures, and attempts to match these scores with counterpart analyses performed on target textures. Multiple parallel populations are used, to promote genetic diversity. Genshade can be run in automatic or interactive modes.

Gentropy uses strictly automatic texture evolution[18]. Unlike Genshade's high-level texture language, Gentropy uses a lower-level set of texture generators, such as basic mathematical operators, and noise and turbulence effects. A suite of different image feature tests is available, ranging from simple pixel-to-pixel colour matches, to higher-level wavelet shape matching. Although each feature test by itself is not a satisfactory nor adequate metric for image matching, a combination of different tests often gives impressive results. Nevertheless, sometimes the most pleasing results are those that do not necessarily have the highest fitness score. Hence the notion of an "optimal solution" is not entirely pertinent in this problem domain.

Gentropy's use of multiple feature tests is effective for automatic texture evolution. A straight-forward combination of feature tests, however, often generates unsatisfactory results. This is because one feature usually dominates the overall score for the run, resulting in a texture biased towards that particular feature test. Although this behaviour can be lessened with a strategic weighted combination of fitness scores, in general this is an unsuitable solution, because it is intuitively difficult to reconcile independent feature tests with a set of *ad hoc* predefined weights. The stochastic nature of genetic algorithms dictates that one run can differ significantly from another, which a static definition of weights may be unable to effectively address.

To overcome this problem, Gentropy resorts to the use of an Island-model genetic algorithm. A network of demes is defined, in which each deme is dedicated to one or more feature tests, possibly on different target textures. At the highest level in the deme network, the various results from other demes are combined into an overall score, using some weighted sum of feature scores. This differs from Genshade's parallel model, in which each population uses the same standard feature tests. Unfortunately, this approach is computationally expensive, as the combined population size was often over 5000.

2.3 Texture feature tests

Image feature tests evaluate how closely various visual characteristics of candidate textures match against target textures. These feature tests are adapted from those used by *query by image content* systems [13)]. The Gentropy system supports a number different feature tests, and any combination of them can be used within runs. The goal of each test is to act as a heuristic for matching the image produced by a candidate texture with the target texture, by measuring how close the candidate image matches the target's feature of interest. Except for the most trivial textures, it is unlikely that an evolved texture will exactly match the target. Hence perfect feature matches are not expected. In any case, it is not a goal of texture evolution to generate the exact target texture. Rather, the goal is to evolve a new texture having similar characteristics of the target. The remainder of this section briefly reviews the feature tests used in this research. See [18)] for more details on feature tests.

Gentropy's feature tests fall into one of three general categories: colour, shape, and smoothness (Table 1). These categories are not mutually exclusive. For example, the CDIR test indirectly evaluates shape and smoothness features as well, even though colour is the primary characteristic of interest. The colour

Colour tests	Description
CDIR	Pixel-by-pixel colour correspondence.
CHISTQ	Matches similar colours, position independent.
Shape test	Description
WAV	Matches wavelet coefficients.
Smoothness test	Description
SHIST	Matches colour smoothness, position independent.

Table 1 Feature test summary

matching tests evaluate colour characteristics of an image:

1. CDIR (colour direct): This test matches a target and candidate texture pixel-by-pixel. The distance in RGB colour space between a candidate's generated pixel colour and the corresponding target's pixel colour is computed. The overall distance is summed for all pixels in the image.
2. CHISTQ (colour histogram quadratic): The image is first quantized, by rounding colours into coarser ranges. Then a histogram of quantized colour frequencies is calculated. The histograms for two images are then compared with one another, and an overall distance between them is calculated. The term "quadratic" refers to the fact that all the histogram entries in both images are compared exhaustively with one another, to determine how close the colours distributions are between the images. Unlike CDIR, the CHISTQ test is position-independent, as it does not consider the locations of colours within images.

Shape tests evaluate pattern and edge correspondences:

1. WAV (wavelet): This performs a wavelet comparison of two images. An image is first converted to grey-scale, by assigning shades of grey to the frequencies of quantized colours in the original. Then basic Haar wavelet decompositions are performed on the rows and columns of this grey-scale image 14). The most pronounced coefficients in the image are then saved. The wavelet decompositions of two images are compared with each other, resulting in a basic shape comparison between the images.

Smoothness tests analyze inter-pixel deviations.

1. SHIST (smoothness histogram): This measures the degree to which a pixel deviates from its eight surrounding neighbour pixels, and thus measures colour discontinuity. The relative deviation is mapped into a grey-scale image, which essentially is a type of edge analyses of the texture. A frequency histogram is then computed for it, and used for comparison.

§3 Evolutionary Multiobjective Optimization

3.1 Pareto Ranking

A multiobjective optimization problem (MOP) is characterized by a set of multiple objectives or parameters, often of which are related to one another in conflicting, nonlinear ways. Evolutionary computation has been widely applied to MOP's [2, 16]. Their success in MOP resides in their natural adaptability to the MOP characterization of problems in terms of representation (chromosomes) and performance evaluation (fitness functions), and the correspondence of the multidimensional MOP search space with the schema characterization of evolutionary search [5].

A popular approach to solving MOP with genetic algorithms is Goldberg's Pareto ranking scheme [3]. The basic idea of a Pareto ranking is to preserve the independence of objectives. This is done by retaining a set of possible solutions, all of which are legitimate solutions with respect to the population at large. This contrasts with a pure genetic algorithm's attempt to ascribe one optimal solution for a MOP, which necessitates a reconciliation of different objective strengths in order to obtain a single optimal solution. Relating different objective dimensions with one another can be difficult, and the results are usually unsatisfactory for nontrivial MOP's.

The following is based on a discussion in [16], and defines the concept Pareto ranking more rigorously. We assume that the MOP is a maximization problem (higher scores are preferred).

Definition 3.1
Given a problem defined by a vector of objectives $\vec{f} = (f_1, ..., f_k)$ subject to appropriate problem constraints. Then vector $\vec{u}$ **dominates** $\vec{v}$ iff $\forall i \in (1, ..., k) : u_i \geq v_i \wedge \exists i \in (1, ..., k) : u_i > v_i$. This is denoted as $\vec{u} \succeq \vec{v}$.

```
Curr_Rank := 1
N := (population size)
m := N
While N $\neq$ 0 {     /* process entire population */
    For i := 1 to m {     /* find members in current rank */
        If $\vec{v}_i$ is nondominated {
            rank($\vec{v}_i$) := Curr_Rank
        }
    }
    For i := 1 to m {    /* remove ranked members from population */
        if rank($\vec{v}_i$) = Curr_Rank {
            Remove $\vec{v}_i$ from population
            N := N-1
        }
    }
    Curr_Rank := Curr_Rank + 1
    m := N
}
```

Fig. 2 Pareto Ranking Algorithm

The above definition says that a vector is dominated if another vector exists which is better in at least 1 objective, and at least as good in the remaining objectives.

Definition 3.2
A solution $\vec{v}$ is **Pareto optimal** if there is no other vector $\vec{u}$ in the search space that dominates $\vec{v}$.

Definition 3.3
For a given MOP, the **Pareto optimal set** $\mathcal{P}^*$ is the set of vectors $\vec{v}_i$ such that $\forall v_i : \neg\exists\vec{u} : \vec{u} \succeq \vec{v}_i$.

Definition 3.4
For a given MOP, the **Pareto front** is a subset of the Pareto optimal set.

A typical MOP will have a multitude of conceivable solutions in its Pareto optimal set. Therefore, in a successful run of a genetic algorithm, the Pareto front will be the set of solutions obtained.

To implement Pareto scoring in a genetic algorithm, chromosome fitness scores take the form of *Pareto ranks.* Figure 2 shows how a Pareto ranking can be computed for a set of vectors. To compute Pareto ranks, the set of nondominated vectors in the population are assigned rank 1. These vectors are removed, and the remaining set of nondominated vectors are assigned rank 2. This is repeated until the entire population is ranked. Genetic evolution then proceeds as usual. Note that Pareto ranks are always relative to the current population. This implies that every generation in a run will have at least a rank 1 set. This has repercussions on performance measurements, as there is no concept of "best solution" amongst all the rank 1 members.

3.2 Population diversity strategies

The Pareto ranking strategy in Figure 2 will invariably suffer from premature convergence, and hence populations that lack diversity. This occurs because the discrete Pareto ranks define a coarse search space. As soon as a significantly improved candidate chromosome is discovered, it will quickly dominate rank 1. To compensate for this, attention has been directed towards the maintenance of genetic diversity within the Pareto ranks [16)]. For example, fitness sharing amongst solutions in the Pareto front will prevent premature convergence. This can be done with respect to population density amongst niches [6)], or vector distances between members [11)]. Others have suggested more automated, parameterless techniques for Pareto population distribution, which do not require foreknowledge of objective fitness space characteristics [9)]. In any case, the use of some strategy for maintaining population diversity within the Pareto rankings is mandatory for most problems.

To counteract premature convergence, we implement two population diversity or diffusion schemes. These strategies are generic, and should give satisfactory results for many MOPs. Besides their simplicity and low overhead, an advantage of these strategies is that the user does not need to submit parameterizations of the objective fitness space or population characteristics. We do not claim that these diversity strategies are exceptionally unique within the evolutionary MOP literature. In fact, they are largely inspired by the population density and fitness sharing approachs mentioned above [6, 11, 9)].

To encourage population diversity, fitness evaluation must award diverse individuals. One indicator of diversity is the proximity of an individual's objective vector to those of its fellow members of the rank set. The heuris-

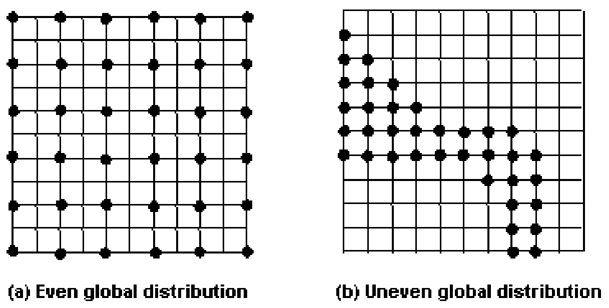


Fig. 3 Effects of nearest-neighbour heuristic on diversity

tic chosen here is the *nearest-neighbour distance* between population members within ranks. This diversity heuristic can be computed with no prior knowledge of the topology of the multiobjective fitness landscape. It captures the fact that a perfectly diverse population will have equal nearest-neighbour distance measurements (Figure 3a). It is only a local measurement of diversity, as it does not examine global distribution characteristics of the population. Hence, the global distribution can be unbalanced (Figure 3b). There is a debate in evolutionary MOP research whether crossover amongst widely diverse members in the same rank set is detrimental, and whether mating amongst distant rank set members should be restricted [16)]. Hence a localized measurement of diversity such as nearest-neighbour distance may be preferrable for some problems. It must be emphasized that texture evolution does not require overly precise diversity testing. The feature analyses used are a rudimentary estimation of texture suitability, and are always secondary to the user's subjective aesthetic judgement at the end of a run.

Both diversity strategies score individuals such that the following two constraints are maintained. In the following, each individual x_i in a population has an associated objective vector $\vec{v}_i$. We will often refer to population members by their objective vectors. Firstly, fitness scores respect the Pareto rank hierarchy:

$$\begin{array}{l} if \;\; Rank(\vec{v}_i) < Rank(\vec{v}_j) \\ then \;\; score(\vec{v}_i) > score(\vec{v}_j) \end{array}$$

Secondly, given two feature vectors $\vec{v}_i$ and $\vec{v}_j$ for individuals belonging to the same rank set R:

```
1. Find Pareto ranks.
2. Compute feature-space distance to nearest neighbour for population:
   For each rank set $R_i$ {
      For each individual $\vec{v}_j \in R_i$ {
         Compute feature-space distance between $\vec{v}_j = (v_1^j, v_2^j, ..., v_n^j)$
            and all other $\vec{v}_k \in R_i (k \neq j)$
            where $dist_{jk} = \sqrt{\sum_{i=1}^{n} (v_i^j - v_i^k)^2}$.
         $MinDist_j$ := (minimum $dist_{jk}$, the nearest-neighbour distance)
      }
   }
3. Convert nearest-neighbour distance into fitness score:
   For each rank set $R_i$ {
      $Low_i$ := (minimum fitness score for rank $R_i$)
      $High_i$ := (maximum fitness score for rank $R_i$)
      $d_{min}$ := (minimum $MinDist_j$ for members in $R_i$)
      $d_{max}$ := (maximum $MinDist_j$ for members in $R_i$)
      For each individual $v_j \in R_i$ {
         $score_j := Low_i + \frac{MinDist_j - d_{min}}{d_{max} - d_{min}} * (High_i - Low_i) * c$
      }
   }
```

Fig. 4 Diversity Rating Algorithm Div_1

$$\begin{aligned} &if\ \ nearest\ \ neighbour\ \ distance(\vec{v}_i) > nearest\ \ neighbour\ \ distance(\vec{v}_j) \\ &then\ \ score(\vec{v}_i) > score(\vec{v}_j) \end{aligned}$$

The algorithms compute a score for each population member using the above constraints. Selection will then favour more optimal (lower) Pareto ranked individuals, and more diverse individuals within the same rank.

[1] Div_1: Nearest neighbour distance diversity

Figure 4 shows pseudocode for the first diversity scoring algorithm, Div_1. In step 1, the normal Pareto ranks are assigned to the population, as done in Figure 2. Then the members in each Pareto rank set are processed . In step 2, the objective-space distance between each member and its nearest neighbour within members in its rank set is computed. Finally, in step 3, these nearest-

#	$\vec{v}$	*Nearest neighbour*	*MinDist*	*Score*
1	(0.1, 0.2, 0.9)	2	0.4243	0.92
2	(0.2, 0.1, 0.5)	3	0.3000	0.90
3	(0.4, 0.3, 0.4)	2	0.3000	0.90
4	(0.8, 0.9, 0.0)	3	0.8246	0.99

Table 2 Div_1 scoring example. Scores range is [0.90, 0.99].

neighbour distances are scaled into a score. It is assumed that the scores for all ranks will be assigned in a fractional manner, perhaps between 0.0 and 1.0, where a perfect solution is 1.0. Score calculation is done linearly with respect to the nearest neighbour distances obtained for members in that rank set, and is scaled into a range $[Low_i, High_i)$ for each rank R_i, under the above constraints. The member of R_i with the longest nearest-neighbour distance is assigned a score of $High_i * c$, and the individual with the shortest nearest-neighbour distance is mapped to Low_i. The c constant is a fraction < 1.0. It is used to prevent scores getting assigned to $High_i$, which belongs to the next rank set, or 1.0 (a perfect solution). Experiments in Section 5 use $c = 0.90$. Note that a perfect objective score (eg. 1.0) will be mapped to $1.0 * c$, which might need to be accounted for within the genetic algorithm. Also note that the linear mapping performed here works well with tournament selection. Other selection schemes may benefit with other mappings.

An example of Div_1 scoring is given in Table 2. Note how these 4 feature vectors are undomininated with respect to one another, and hence are in the same rank 1 set should they represent the entire population. *Nearest neighbour* refers to the ID of the nearest neighbour for each member, and *MinDist* is the corresponding distance to it. The score is calculated for the range [0.9, 0.99]. Using the formula from Figure 4, this range could be computed with $Low_1 = 0.90$, $High_1 = 1.0$, and $c = 0.9$.

[2] Div_2: Ranked nearest neighbour distance diversity

Although Div_1's scoring formula preserves the Pareto ranks, the nearest-neighbour distances within each rank's score range is computed from raw objective score values. As with weighted sums of multiple objective scores, this strategy can be unduly affected by changes in single objectives. This may occur because multiple objective tests are not uniform in their metrics spaces. A

#	$\vec{v}$	$\vec{d}$	$\vec{r}$	$ravg$	Score
1	(0.1, 0.2, 0.9)	(0.1, 0.1, 0.4)	(1, 1, 2)	1.33	0.922
2	(0.2, 0.1, 0.5)	(0.1, 0.1, 0.1)	(1, 1, 1)	1.0	0.90
3	(0.4, 0.3, 0.4)	(0.2, 0.1, 0.1)	(2, 1, 1)	1.33	0.922
4	(0.8, 0.9, 0.0)	(0.4, 0.6, 0.4)	(3, 2, 2)	2.33	0.99

Table 3 Div_2 scoring example

drastic change occur in one particular objective may impact the overall nearest-neighbour distance.

The Div_2 scoring scheme in Figure 5 also uses nearest neighbour distances as a diversity heuristic. Rather than using a raw combined nearest-neighbour distance, however, Div_2 normalizes these distances into relative ranks, and keeps the ranked ordering for each objective independent from one another. This ensures that objective distances will not unfairly dominate one other. In step 2, the minimum distance for each feature dimension is determined for every population member. This differs from Div_1, which computes the overall distance in feature-space. Step 3 then converts these feature distances into ranks, where each r_f corresponds to the ranking of feature v_f. The average rank value is computed in step 4. It is then converted to a fitness score in step 5, in the same manner as in Div_1.

An example of Div_2 scoring is in Table 3. The same 4 individuals from Table 2 are used. The net effect on the scores compared to those in Table 2 is that vector #3 now has an intermediate fitness level, whereas it is considered less fit in Table 2.

§4 Experiment

Table 4 summarizes the MOP strategies and feature test sets we used in our experiments. The two feature sets use at least one feature test from each of the colour, shape, and smoothness categories. Not all MOP strategies were run with all feature test sets, since it was clear during early runs that pure Pareto consistently produced poor results.

The strongly-typed lilGP 1.1 system is used [19]. LilGP is a C-based system, which implements tree-based genetic programming [8]. Table 5 lists the common parameters used in all the experiments. The GP parameters are standard in the literature; see [8] for details. A total of five rank 1 solutions were randomly extracted per run, which the user can then select from. Although the

MOP strategies:
1a. pure Pareto
1b. Div_1
1c. Div_2
Feature test sets:
2a. CHISTQ, WAV, SHIST
2b. CDIR, CHISTQ, WAV, SHIST

Table 4 Major parameter sets

actual rank 1 set is often in the 100's, the generation of five solutions is adequate to show the relative diversity of the population. The image parameters are particular to the image processing done during the feature tests described in Section 2.3.

The texture language, inspired by one in [12], is outlined in Table 6. LilGP's strong typing is useful for differentiating expressions that operate over floats and RGB colour vectors (an array of three float values). During interpretation, numeric values in floats and vectors are truncated to the range [-1.0, 1.0] before converted to RGB. The float terminals x and y are the current 2D coordinates being processed. An ephemeral constant (float or vector) is a constant that is initialized with a random value when created, and then retains that value throughout its lifetime during a run. The float nonterminal set includes standard arithmetic and trigonometric functions. Some specialty texture-oriented functions are also included. *lum* computes luminance by averaging the RGB channels. *avg* returns the mean of two arguments. Repeating tile patterns are generated with *tilerad*. Various texture effects are generated by *noise*, *turb*, *turbflow*, and *cloud*. The *if* function permits conditional processing, and *forv*, *chn*, and *ichn* perform iterative processing on vectors. Vectors terminals include ephemeral constants, as well as *colgrad*, which generates a vector using the current x, y, and distance to origin. The nonterminal *rgb* constructs a vector from 3 float values. The remaining vector nonterminals generate a variety of noise and other texture effects. Further details about the texture primitives are found in [18].

§5 Results

GP Parameter	*Value*
Evolution paradigm	generational
Max generations	100
Runs/experiment	6
Rank 1 solutions/run	5
Population size	1000
Initialization	ramped half&half
Initial tree depth	2 to 6
Max tree nodes	100
Max tree depth	50
Crossover rate	0.90
Mutation rate	0.10
Tournament size	5
Image Parameter	*Value*
Resolution	50x50
# quantized colours	1000
# quantized greys	50
# wavelet coefficients	50

Table 5 Common experiment parameters

Figures 6 and 7 illustrate solutions that show typical behaviours of the pure Pareto, Div_1 and Div_2 experiments. These results are from the feature set 2b in Figure 4. Two separate run results are shown for each experiment, and 5 random solutions are shown for each run. These runs are selected from a total of 6 for each experiment, and are chosen as examples of better quality solutions. All of these experiments used the same random number seed, and hence the same initial population. The individual feature test scores are included with each texture. These scores are between 0 (worst) to 100 (best). The top feature test scores are underlined.

In the Pareto runs, the first thing to note is that the solutions show a high degree of convergence. In other Pareto runs examined, it was common to find all the solutions to be identical. Also note how a few of the solutions in run 2 have a problem with colour; the CDIR scores are correspondingly low. Compared to the Pareto runs, the Div_1 and Div_2 runs have a better overall colour match

Float terminals:	x, y, *ephemeral constants*
Float nonterminals:	lum, avg, +, -, diff, *, /, max, min, not, sin, cos, mod, log, pow, tilerad, noise, turb, turbflow, cloud, if, chn, ichn
Vector terminals:	colgrad, *ephemeral constants*
Vector nonterminals:	rgb, marble, warprel, warpabs, kaleid, tile, forv

Table 6 Texture language

with the target image. Furthermore, the Div_2 runs have more diverse solutions than Div_1, thus showing how the ranked nearest neighbour diversity heuristic is advantageous. Nevertheless, Div_1 did produce more higher-scoring feature tests. Which solutions from these two strategies are better is a subjective decision.

Figures 8 and 9 show good results selected from different runs using Div_2 diversity, and using the two different feature test sets from Figure 4. In the first figure, both feature test sets produce visually similar results. It was found that set 2b, which uses the additional CDIR test, tended to produce textures that matched the positions of colours in the target, since the CDIR test scores positional colour matches. A good example of this tendency is the first texture in the upper row 2b. That texture is attempting to create a colour gradient that roughly matches the target colour distribution. This is similarly apparent in the first texture of the lower row for test 2b. Here, there is a good attempt at putting colours into the quadrants where they are found in the target.

The stripe target texture in Figure 9 are fairly well simulated in the results, and there are not many differences in the quality of the Div_1 and Div_2 runs. Perhaps the most challenging texture studied is the second one in this figure. This target has a wide variety of colours of different luminosities within complex convoluted shapes. All the results shown have fairly good colour and luminosity (brightness) matches. The shape was difficult to reproduce, however. Many results used a familiar radial pattern, which is seen in 8 of the 10 results shown. This difficulty is perhaps due to too imprecise a wavelet analysis, or a lack of necessary texture primitives. We also noted that the 2b results were much less varied than the corresponding 2a ones. This is due to 2b's attempt at positional colour matching. An example solution showing this is the second last one in the 2b row.

Table 10 shows some results using multiple targets – one target image for shape, and the other for colour. These experiments use Div_2 with the CHISTQ feature test for colour, and the WAV feature test for shape. The results in the figure show a number of hand-selected solutions from multiple runs.

§6 Conclusion

This research establishes that evolutionary MOP with diversity is ideally suited to texture evolution. Pareto with diversity is an excellent way to evolve a variety of textures from which the user can inspect and select, which is natural for texture evolution, given the ultimately subjective nature of the problem. Treating multiple feature tests as independent objectives removes the difficulty of reconciling independent feature scores. The results obtained here with the Div_2 diversity strategy are competitive with the non-MOP ones in [18]. One difference, however, is that these MOP results were obtained more efficiently, as a population size of 1000 was used here, instead of 5600 used in [18].

The pure Pareto strategy without diversity heuristics was unacceptable since, predictably, premature convergence always arose. The nearest-neighbour distance strategies used by Div_1 and Div_2 prevented premature convergence. Although Div_1's population was diverse, the quality of solutions was often unsatisfactory. This undoubtedly arose because of domination by individual features when computing the raw nearest-neighbour distance in objective space. Div_2's ranked diversity scoring consistently produced the best results.

One unexpected discovery is that increasing the number of feature tests can detract from the quality of evolved results. This contradicts the intuitively appealing idea that a large bank of feature tests would result in more accurate analyses: since each feature test measures one specific characteristic of an image, lots of tests will therefore cover a host of various aspects. After comparing the results from feature sets 2a and 2b, we found that the 2b results, which use the additional CDIR test, were often weaker. This was puzzling at first, especially considering the tenet of Pareto ranking – that features scores be kept independent. The reason this happens, however, is because an increase in the number of objectives creates a more difficult optimization problem. With additional objectives, populations are stretched thinner across the corresponding higher-dimensional search space. With respect to texture evolution, fewer feature tests means that the population has a greater opportunity to evolve solutions with better performance in all feature dimensions, which naturally results in better overall matches to target textures.

There are a number of obvious ways in which we could evolve better quality results using our system. First, more sophisticated feature tests could be used. A simple extension would be to increase the number of coefficients used in the wavelet analysis. More advanced extensions would employ more sophisticated image analyses as feature tests. Of course, such techniques will impact computation time, and hence evolution efficiency. Second, the texture language can be expanded. Fractals and other texture generation primitives can be added to enrich the texture generation formulae [10]. Multiple expressions taking the form of texture channels could be used, which would result in more complex images. An advantage of using MOP evolution is that it removed the need for multiple subpopulations, thus reducing computational effort. Nevertheless, it would be interesting to apply distributed MOP evolution to texture generation, especially if more complex feature tests were to be used.

Our treatment of texture evolution as an MOP can useful in other unsupervised systems such as Genshade [7]. In addition, population diffusion heuristics might be beneficial in interactive systems, to ensure that the user is not presented with a population of identical textures.

The nearest-neighbour diversity heuristic used in Div_1 and Div_2 are similar in spirit to other diversity promotion strategies. We do not require the user to set any problem-specific parameterization of the MOP search space, unlike [6, 11], which require the user to enter *niche radius* values in order to define the objective size of niches. This requires some foreknowledge of the MOP fitness space, and can adversely affect evolution if the values are inadequate. [9] present some alternative strategies for promoting diversity in evolutionary MOP, many of which are applicable to texture evolution. However, given the ultimately subjective nature of texture selection, it is unlikely that more sophisticated diversity strategies will add significant improvement to texture evolution as done in this paper.

Acknowledgement: Thanks to Andrea Wiens and her work on the Gentropy system, upon which this research is founded; and to Beatrice Ombuki for her helpful feedback. This research is supported by NSERC Operating Grant 138467-1998 and an NSERC USRA award.

References

1) D.S. Ebert, F.K. Musgrave, D. Peachey, K. Perlin, and S. Worley. *Texturing and Modeling: a Procedural Approach*. Academic Press, 2 edition, 1998.

2) C.M. Fonseca and P.J. Fleming. An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation*, 3(1):1–16, 1995.

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

4) P.S. Heckbert. A Seed Fill Algorithm. In A. Glassner, editor, *Graphics Gems*, pages 275–277. Academic Press, 1990.

5) J.H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1992.

6) J. Horn, N. Nafpliotis, and D.E. Goldberg. A Niched Pareto Genetic Algorithm for Multiobjective Optimization. In *Proceedings ICEC'94*, pages 82–87, 1994.

7) A.E.M. Ibrahim. *GenShade: an Evolutionary Approach to Automatic and Interactive Procedural Texture Generation*. PhD thesis, Texas A&M University, December 1998.

8) J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

9) R.C. Purshouse and P.J. Fleming. Elitism, Sharing, and Ranking Choices in Evolutionary Multi-Criterion Optimisation. Technical Report 815, Dept. of Automatic Control and Systems Engineering, University of Sheffield, January 2002.

10) S. Rooke. Eons of Genetically Evolved Algorithmic Images. In P.J. Bentley and D.W. Corne, editors, *Creative Evolutionary Systems*, pages 330–365. Morgan Kaufmann, 2002.

11) J. Rowe, K. Vinsen, and N. Marvin. Parallel GAs for Multiobjective Functions. In *Proceedings of the 2nd Nordic Workshop on Genetic Algorithms and their Applications (2NWGA)*, pages 61–70, University of Vaasa, Finland, 1996.

12) K. Sims. Interactive evolution of equations for procedural models. *The Visual Computer*, 9:466–476, 1993.

13) J.R. Smith. *Integrated spatial and feature image systems: retrieval, analysis and compression*. PhD thesis, Center for Telecommunications Research, Graduate School of Arts and Sciences, Columbia University, 1997.

14) E. Stollnitz, T. Derose, and D. Salesin. *Wavelets for Computer Graphics: Theory and Application*. Morgan Kaufmann, 1996.

15) S. Upstill. *The Renderman Companion: A Programmer's Guide to Realistic Computer Graphics*. Addison-Wesley, 1989.

16) D.A. van Veldhuizen and G.B. Lamont. Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art. *Evolutionary Computation*, 8(2):125–147, 2000.

17) A. Watt and M. Watt. *Advanced Animation and Rendering Techniques: Theory and Practice*. ACM Press, 1992.

18) A.L. Wiens and B.J. Ross. Gentropy: Evolutionary 2D Texture Generation. *Computers and Graphics Journal*, 26(1):75–88, February 2002.

19) D. Zongker and B. Punch. *lil-gp 1.0 User's Manual.* Dept. of Computer Science, Michigan State University, 1995.

```
1. Find Pareto ranks.
2. Compute feature distance vectors to nearest neighbours:
   For each rank set $R_i$ {
       For each individual $\vec{v}_j \in R_i$ {
           Compute $\vec{d}_j := (d^j_1, ..., d^j_n)$
               where each $d^j_f$ is the minimum $|v^j_f - v^k_f|$
                   for all $\vec{v}_k \in R_i (k \neq j), (f = 1, ..., n)$
       }
   }
3. Assign ranks for all feature distances to nearest neighbour:
               $\vec{r}_i := (r^i_1, r^i_2, ..., r^i_n)$
   where ranked ordering increases as distances $d^j_l$ increase.
4. Compute average rank for each individual:
               $ravg_i := (\sum_{f=1}^{n} r^i_f)/n$
3. Convert average nearest-neighbour ranks into fitness score:
   For each rank set $R_i$ {
       $Low_i$ := (minimum fitness score for rank $R_i$)
       $High_i$ := (maximum fitness score for rank $R_i$)
       $r_{min}$ := (minimum $r\_avg_j$ for members in $R_i$)
       $r_{max}$ := (maximum $r\_avg_j$ for members in $R_i$)
       For each individual $v_j \in R_i$ {
           $score_j := Low_i + \frac{ravg_j - r_{min}}{r_{max} - r_{min}} * (High_i - Low_i) * c$
       }
   }
```

Fig. 5 Diversity Rating Algorithm Div_2

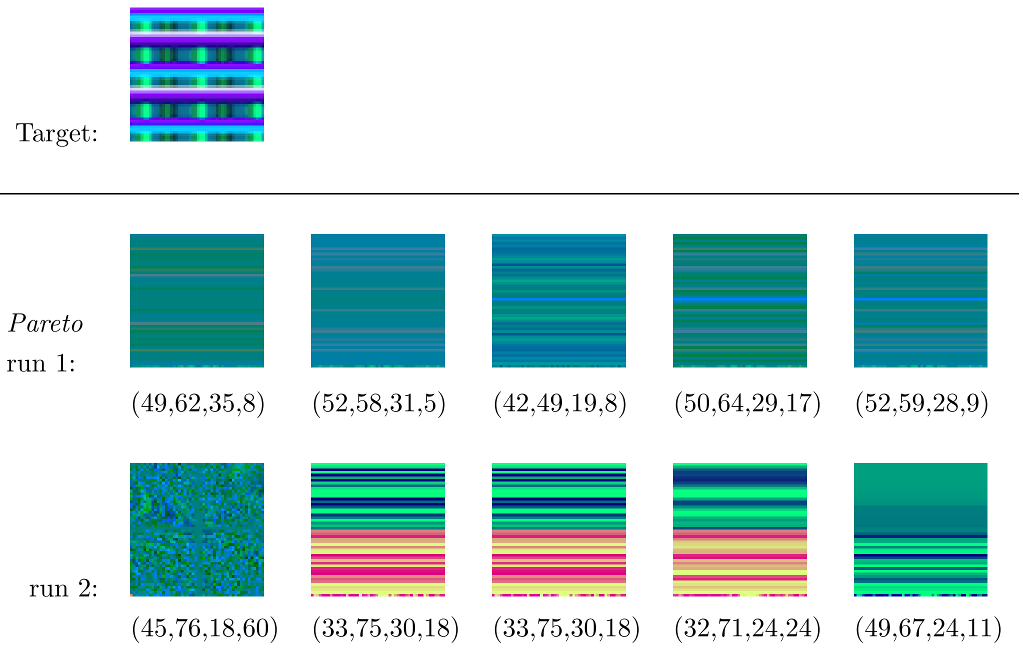


Fig. 6 Randomly selected rank 1 solutions from single runs for Pareto, with feature set 2b (CDIR, CHISTQ, WAV, SHIST).

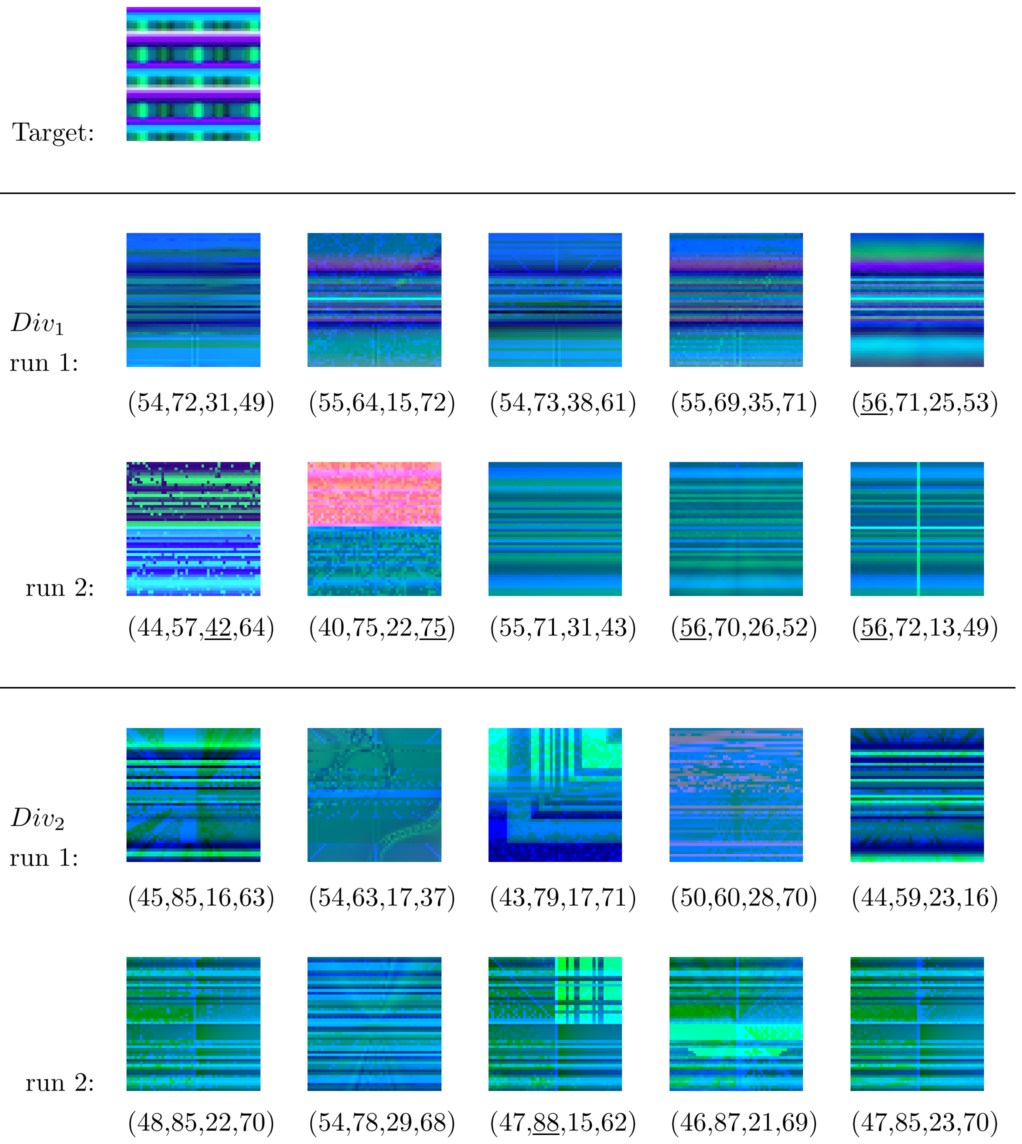


Fig. 7 Randomly selected rank 1 solutions for Div_1 and Div_2. Feature set 2b (CDIR, CHISTQ, WAV, SHIST).

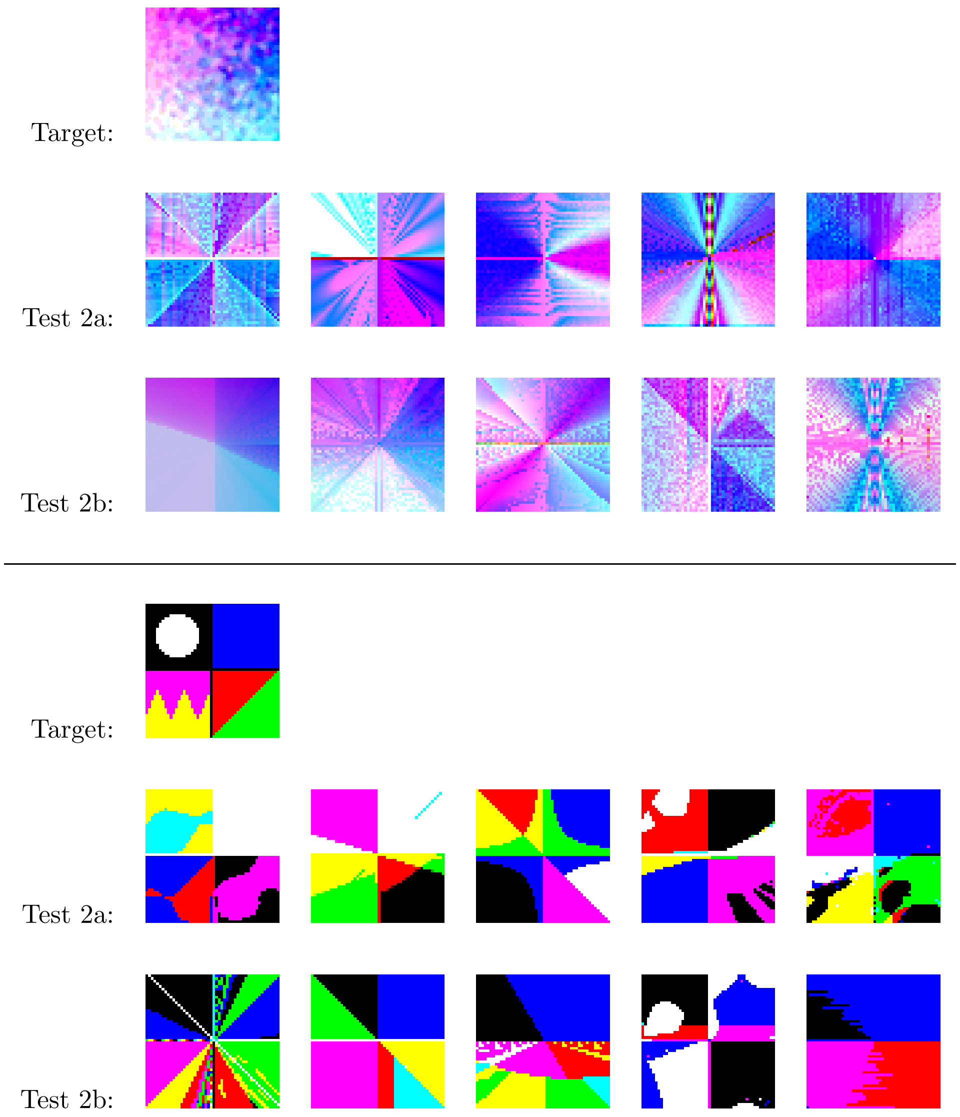


Fig. 8 Selection of results from Div_2

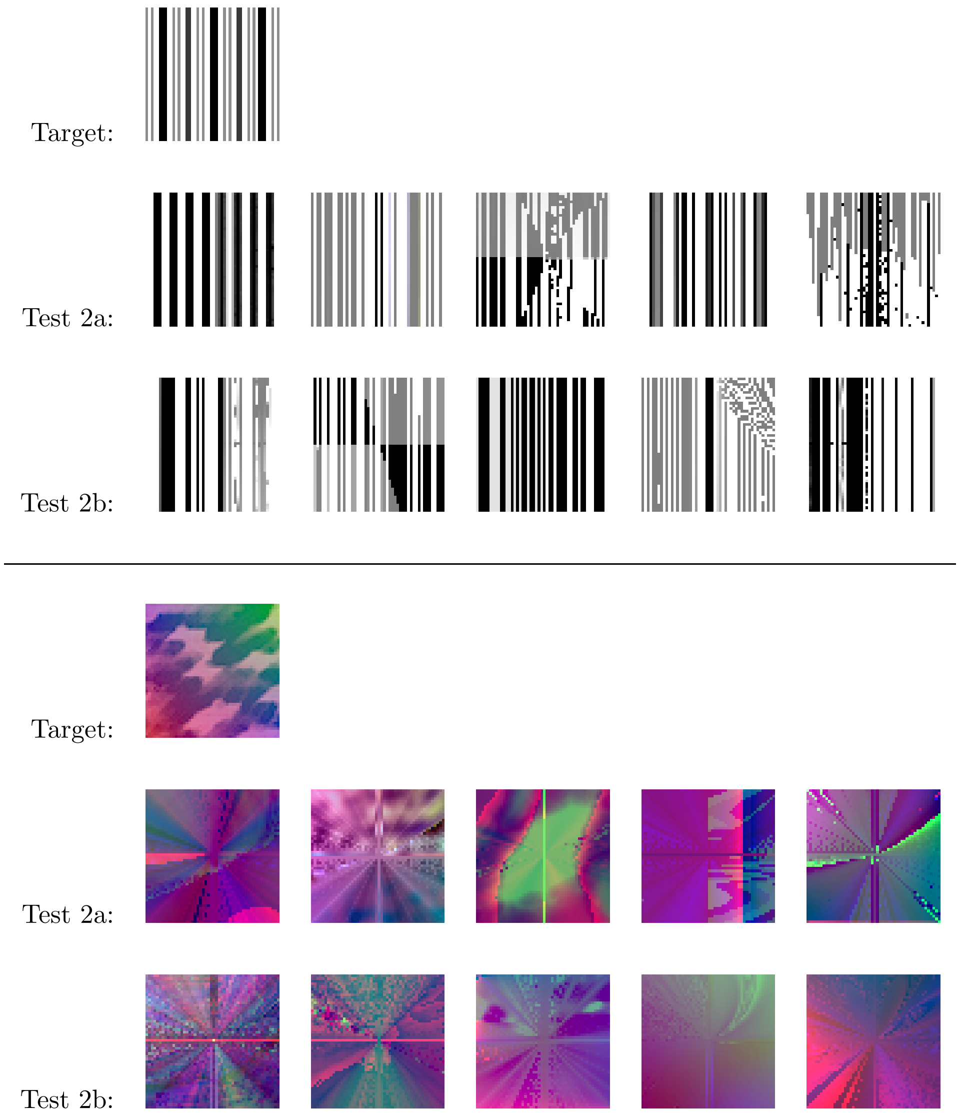
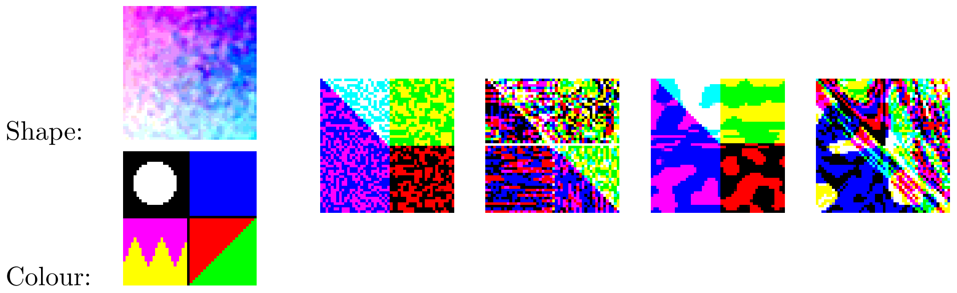


Fig. 9 More results from Div_2



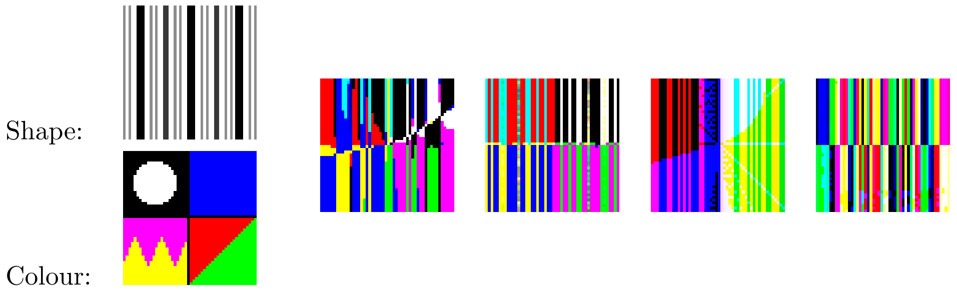


Fig. 10 Multi-target results using Div_2