# Investigating the use of aesthetic measures over time with unsupervised evolutionary art

Ben Schlegel

March 21, 2021

**Abstract**

We present a study of several aesthetic measures over time in unsupervised evolutionary art. We evolve shaders without any human evaluation, using aesthetic scores as our fitness function. We take samples of the shader at several timestamps, calculate the score, and sum all the scores up to do fitness evaluation

# Introduction

Evolutionary art is a field investigating the use of evolutionary computing techniques to produce aesthetically pleasing images. Traditionally this involves the assistance of human judges, determining which phenotypes live on. This leads to a fitness bottleneck, and also leads to subjectivity. Using aesthetic measures gives us numbers to look at and run statistics on, and also does not rely on humans to function. This means the production and evolution of our genotypes can be speed up massively. We have implemented three aesthetic measures to evolve our images with. The shaders will be placed on a blank canvas, and the images produced will be evaluated using a single aesthetic measure at several time steps. The sum of the scores at each time step will then be used to give a fitness score to the shader. These fitness scores will be used to determine which shaders survive and are used to produce the next generation.

# Aesthetic Measures

A set of X images are produced using each shader for different time steps. These are then fed through an aesthetic measurement to get a score, and the scores are then averaged to get a score for the shader. A score for producing change over time is also added, and is discussed further on.

## Shannon Entropy

Shannon entropy is an aesthetic measure that attempts to use information theory to get an aesthetic value of an image. We use a method similar to Heijer and Eiben to calculate the score for these images. First we convert the image to grayscale using the average method, and then sort the image regions into a histogram of 128 values. We then calculate the score by

$$M = -\sum_{i=0}^{127} p(x_i) * log(p(x_i))$$

where $p(x_i)$ is the probability that a random region will have that value. The image will score high on this measure if the probability is distributed uniformly.

## Bell Curve

This aesthetic measure is based on the observation that many fine art paintings have color gradients that conform to a normal distribution. The measure was first proposed by Ross Ralph and Zong. To start the calculation, we calculate the gradient of the red values of each region using

$$|\Delta r_{i,j}|^2 = (r_{i,j} - r_{i+1,j+1})^2 + (r_{i+1,j} - r_{i,j+1})^2$$

Calculating the blue and green region gradients is similar. Using the gradients for the RGB values of each region, we now calculate the overall gradient $S_{i,j}$ for each region

$$S_{i,j} = \sqrt{|\Delta r_{i,j}|^2 + |\Delta g_{i,j}|^2 + |\Delta b_{i,j}|^2}$$

After this we calculate the response $R_{i,j}$ for each region

$$R_{i,j} = \frac{S_{i,j}}{S_0}$$

where $S_0$ is the detection threshold, set to 2 as in Ross Ralph and Zong. This score is calculated by the difference between the normal distribution and actual distribution, so we need to calculate a mean $\mu$

$$\mu = \frac{\sum_{i,j}(R_{i,j})^2}{\sum_{i,j}(R_{i,j})}$$

and standard deviation $\sigma^2$

$$\sigma^2 = \frac{\sum_{i,j} R_{i,j}(R_{i,j} - \mu)^2}{\sum_{i,j} R_{i,j}}$$

Using $\mu$ and $\sigma$, The values for $R_{i,j}$ are stored in a histogram where bin width is $\sigma/100$. Using the histogram we can calculate the actual probability $p_i$ and the expected probability $q_i$. This is the score

$$M = 1000 \sum p_i * log(\frac{p_i}{q_i})$$

## Reflectional Symmetry

We use a reflectional symmetry aesthetic measure similar to the one define din Heijer and Eiben. First the image is divided into 4 quadrants, $A_1$, $A_2$, $A_3$, $A_4$. Left, right top and bottom are defined as $A_{left} = A_1 \bigcup A_3$, $A_{right} = A_2 \bigcup A_4$, $A_{top} = A_1 \bigcup A_2$, $A_{bottom} = A_3 \bigcup A_4$
Horizontal symmetry is defined as

$$S_h = s(A_{left}, A_{right})$$

vertical symmetry is

$$S_v = s(A_{top}, A_{bottom})$$

and diagonal symmetry is

$$S_d = \frac{s(A_1, A_4) + s(A_2, A_3)}{2}$$

where the similarity between two areas is defined as

$$s(A_i, A_j) = \frac{\sum_{x=0}^{w} \sum_{y=0}^{h} (sim(A_i(x, y), \bar{A}_j(x, y)))}{w * h}$$

where $x$ and $y$ are the coordinates of the region, and $w$ and $h$ are the width and height of the area. $\bar{A}$ is the mirrored area of $A$, with respect to the symmetry we are trying to measure. $sim$ is defined as

$$sim(A_i(x, y), A_j(x, y)) = \begin{cases} 1 & \text{if } |A_i(x, y) - \bar{A}_j(x, y)| < \alpha \\ 0 & \text{otherwise} \end{cases}$$

We use $\alpha = .05$ as our threshold. Therefore the score for strict symmetry is

$$M_{strict} = \frac{S_h + S_v + S_d}{3}$$

There is a danger, however, in just using symmetry as a fitness function, as monochrome and monotonous images are very easy to evolve, and are also very symmetrical. Heijer and Eiben point this out, and suggest a 'liveliness' factor. We do the same here, and use our Shannon Entropy score to add some more variety to the images produced. Therefore our score is

$$M = M_{strict} * M_{shannon}^3$$

# Evolutionary Algorithm

## Representation of shaders

The evolution of shaders is done by our own software. In the program, shaders are represented through 3 expression trees, with one for red, blue, and green colors. In these trees, there are operator nodes, which have two children, and leaf nodes which do not have children. All operators have 2 children to avoid making invalid trees during mutation and crossover stages. Operators are split into 3 types, which determine how their children are used in calculation and writing. Type 1 operators use a "left right" orientation, for example, the plus operator: Uv.x + 3. Type 2 operators are functions with 2 inputs, for example the noise operator, noise(2,4). Type 3 operators are functions with only one input, and to preserve each operator having 2 children, for example the PlusCos operator: 3 + cos(14).

| Operator | Type | Example | Description |
|---|---|---|---|
| Plus | 1 | Uv.x + 32 | Standard Addition |
| Minus | 1 | TIME - 3 | Standard Subtraction |
| Multiply | 1 | 15 * Uv.y | Standard Multiplication |
| Divide | 1 | 23 /32 | Standard Division |
| PlusCos | 3 | Uv.y Cos(34) | Left child added to cosine of right child |
| MultCos | 3 | 20 * Cos(Uv.x) | Left child multiplied by cosine of right child |
| PlusSin | 3 | 24 + sin(32) | Left child added to sin of right child |
| Multiply | 3 | 15 * sin(21) | Left child multiplied by sine of right child |

Leaf nodes are further split into constants, and variables.

| Variable | Description |
|---|---|
| Uv.x | x coordinate of current pixel, normalized [0,1] |
| Uv.y | y coordinate of current pixel, normalized [0,1] |
| TIME | Time since shader started |

These expression trees are traversed inorder to write the trees into a Godot shader. The shader's trees are solved to produce a matrix of colors at different time steps. The average difference between every timestep and its following timestep is then computed, and then the average of those averages is added to the fitness score, with shdaders that vary more being favored over more static shaders.

## Evolving shaders

To start, the algorithm is initialized by shaders with random numbers of nodes and randomized values. Then each shader has its fitness score assigned, by the average difference over time and the chosen aesthetic measure. After the shaders have had their fitness value calculated X number of survivors are selected. The survivors are then paired up, and crossed over by swapping 2 nodes (and by extension its children) on each red, blue, and green tree. Then Y copies are made of each survivor, and then mutated. Then the procedure is repeated until desired stopping point.