Animating Typescript using Aesthetically Evolved Images

Ashley Mills, ashley@ashleymills.com

The University of Kent, School of Computing

Abstract. The genotypic functions from apriori aesthetically evolved images are mutated progressively and their phenotypes sequenced temporally to produce animated versions. The animated versions are mapped onto typeface and combined spatially to produce animated typescript. The output is then discussed with reference to computer aided design and machine learning.

Keywords: aesthetic evolution animated, animated typeface, typescript

1 Introduction

Systems for evolving aesthetic images, whilst still unfamiliar in popular culture, are academically well known and date back at least 20 years [14,15,6,13,8]. Ordinarily the output of an image-centric aesthetic evolution system is seen as the end product in itself; aesthetically pleasing images are the goal. The genotypes of these images however encode the labour of a prolonged and iteratively applied intelligent cognitive filtering process. This serves to apriori mark out these artifacts as cognitively interesting and artistically compelling.

It is prudent therefore to make use of these creatively valuable artifacts in producing further creative output. With this motivation, in this work, the output of an Aesthetic Evolution (AE) system based on [10] is used to produce animated typefaces. This work focuses on using AE output to animate *existing* typefaces, which sets it apart from work such as [7] which focuses on the construction of a system to evolve novel typefaces.

The rest of this document is outlined as follows, in Section-2 we briefly describe the AE system which generates the image inputs for typeface animation, in Section-3 we explain how the images, or rather their genoyptic functions, can be modulated to produce animation, in Section-4 we describe how these animations can be mapped to individual glyphs, and in Section-5 we explain how the last can be combined into typescript. Finally, in Section-6 we discuss the artistic applications and potential machine learning applications that this work inspires.

2 Aesthetic Evolution Basis

The aesthetic evolution system used here employs a classical Cartesian coordinate model [14] as follows. Images are represented genotypically as a triplet of

function trees; one tree for each color in the RGB colorspace. The function trees are initially grown stochastically from a pool of primitive function nodes such as sin, cos, etc with arguments that are themselves either recursively function nodes, or terminal primitives such as numerical constants or external variables. In the set of terminal primitives, the variables x and y exist to represent the Cartesian coordinates of the image and at least one must occur in each generated function tree to produce a valid image.

To produce an image, the genotypic function triplet is evaluated for each pixel, i.e for each x and y coordinate of the output image, where x and y are normalized, irrespective of image size, to fall within the range $[0, 1]$. Thus, the leftmost pixel of an image assumes the value $x = 0$ and the rightmost the value $x = 1$. The return values from the evaluated triplet provide the RGB values for each pixel of the image respectively. A trivial example of such a function triplet is shown in Figure 1, along with its image. Note that the triplet members are the three branches that enter the function "new_hsv".

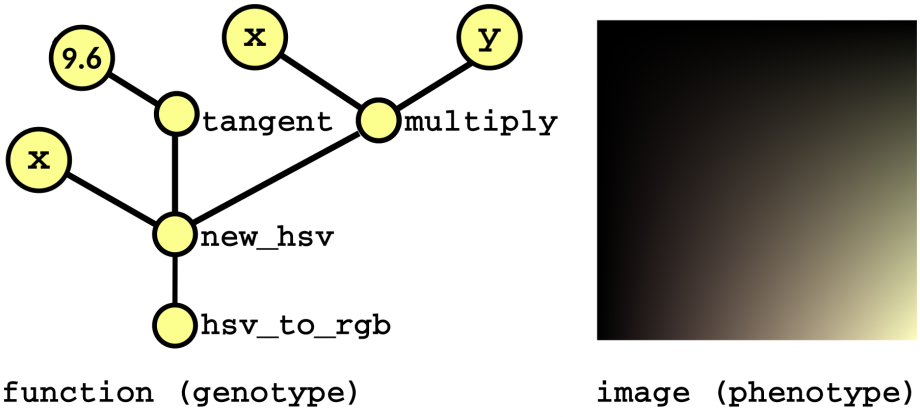

Fig. 1. A simple functional triplet is shown on the left along with its phenotypic expression, the image, on the right. Note in this case, the triplet passes through an HSV colorspace before entering the RGB colorspace.

Successive generations of images are obtained in the well known manner of visual cortex guided manual selection, followed by randomly seeded mutation and crossover. After many such generations, the images produced converge on aesthetically pleasing forms. These images are used as the input to the animated typeface system described subsequently. For more information about the aesthetic evolution system used here, please see [10].

An interesting and useful feature of this AE system is that it outputs phenotype files whose genotypes are embedded within them: each image is output as a JPEG and the genotype is stored in a JPEG metadata segment (APP0 marker segment) of that image. By combining genotype and phenotype into a

single file whose format conforms with an existing standard, the output images can be viewed and loaded by a plethora of existing software, which makes for convenient manipulation and organization. It also makes it a straightforward matter to obtain the genotype for an image when it is desired to manipulate the underlying form. To avoid namespace confusion, the format uses the extension "exp.jpg" meaning expression embedded JPEG instead of the usual ".jpg".

3 Animating Aesthetically Evolved Images

To animate a previously aesthetically evolved image, the underlying function needs be modified through time. This process is relatively straightforward and proceeds as follows. First, an image with appropriate artistic merit is selected and it's nodes are enumerated and examined. For example, the image in Figure 1 has four function nodes "hsv_to_rgb", "new_hsv", "tangent", and "multiply" as well as four terminal nodes "x", "9.6", "x", and "y". After examining this information, the artist might decide to modulate the leftmost x terminal by replacing it with the constant values $x = \{0, 0.1, 0.2, ..., 1.0\}$ to generate 10 new functions. Each of the 10 resultant functions can then be evaluated to produce 10 animation frames.

As an example consider Figure 3 where an interesting function is animated and 6 frames from the animation sequence are shown, taken as equidistant samples across the animated range.

In this particular case the function in question has 191 nodes, 120 of which are functions and 71 of which are terminals. To produce the animation, the node at index 22, which initially has the value x, has been replaced by $x \cdot k$ where $k \in \{0.00, 0.01, 0.02, \ldots 10.00\}$. Each of the latter substitutions generates a new function which can be evaluated to produce a single animation frame.

In the AE system used here, the node "tweaks" are performed by passing the base image (and it's embedded genotype) to a custom tool called "evotool". This supports many kinds of manipulations for both the genotype and phenotype. As an example, the following invocation creates a new AE artifact called "out.exp.jpg" by changing the terminal node at index 22 in the file "in.exp.jpg" to "$sin(x)$". The node index 22 was obtained by previous invocation of evotool with the "node listt" command sequence which lists terminal nodes.

```
evotool node tweakt default 22 "sin(x)" out.exp.jpg in.exp.jpg
```

Thus the process of animating a function becomes a simple case of scripting the use of evotool to create a sequence of progressively tweaked functions and their phenotypes. The outputted image sequence can then be stitched together using the opensource tool FFmpeg [1], in a manner such as:

```
ffmpeg -i out_%d.exp.jpg -c:v libx264 -crf 17 -c:a copy out.mkv
```

To encode the generated image sequence at the default of 25 frames per second as an x264 [9] encoded movie within a Matroska [12] container.

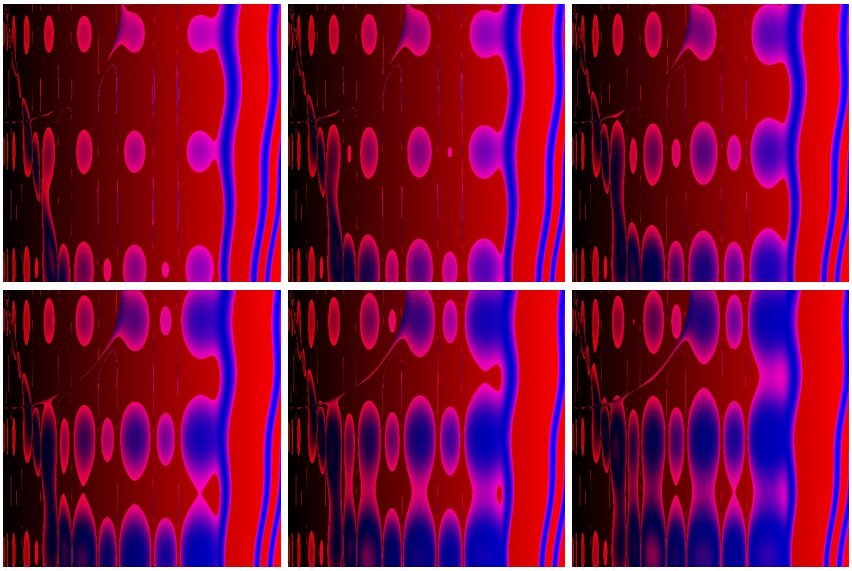

Fig. 2. Six animation frames extracted from a sequence where the terminal at index 22 of the underlying function has assumed the values $x \cdot k$ where $k \in \{0.00, 0.01, 0.02 \ldots 10.00\}$

Some experimentation is required to find interesting modulations, as not all nodes contribute constructively to the phenotype[1]. Of course, it is feasible to modulate more than one node simultaneously, and the sequence of tweak inputs can be arbitrarily controlled. For example, the artist may wish to provide a sequence of inputs consistent in magnitude differentials with that of a audio segment to produce animation that follows the sound. Several examples of animated functions, created in this manner, including an audio-modulated examples, are available to view online [2].

4 Mapping of Animated Functions to Typefaces

To produce an animated typeface, each glyph in the typeface must be mapped to an animated function. The first step in this process is to generate mapping images for each glyph. Here, for each glyph, the mapping image is simply that glyph rendered using anti-aliasing in black against a white background at the desired point size. Figure 3 shows such a mapping image along with a mapped function.

In this case the mapped function has 258 nodes in total of which 100 are function nodes and 158 are terminal nodes. To map the function to the glyph

[1] It is interesting that this "junk DNA" accumulates in AE images, as it apparently does in natural beings.

Fig. 3. An aesthetically evolved image mapped onto the roman alphabet glyph "e" in typeface "Organo" [11] against a black background.

the function is evaluated for each pixel in the map that is not completely white, i.e those pixels which are either part of the image proper, or part of the anti-aliasing. For the anti-aliased pixels, the resultant function output is blended with the background color in the output image in proportion to the level of grayscale in the map, whereas the ordinary pixels take the function output directly. This approach produces a straightforward silhouette of the evaluated function which accords with the shape of the glyph selected, as is shown in Figure 3. This is again achieved using "evotool" and the sub-command "map", which takes only three arguments: the output file, the mapping silhouette as a jpg, and the input file to map:

```
evotool map default out.exp.jpg map.jpg in.exp.jpg
```

Thus, each frame from an animated function is mapped onto a glyph. Once this has been done, the resultant image sequence is composited into a movie as described in the previous section. Examples of animated glyphs can also be viewed online [2].

5 Compositing Animated Typefaces into Animated Typescript

By following the process outlined in the previous section for each glyph in a given typeface, a corresponding typeface that is animated is obtained. By combining animated glyphs together into strings, animated typescript can be created, which provides a novel way to animate text.

Whilst there are many ways to composite video together, the method used here is simple. FFmpeg is used to stitch together the chosen glyphs onto a 1080p white background into a single movie in their correct textual positions. An example command is shown below:

```
ffmpeg -i E.mkv -i V.mkv -i O.mkv -i A.mkv -i R.mkv -i T.mkv \
   -filter_complex "
   color=size=1920x1080:c=white [a];
   [0:v] scale=320x320 [E];
   [1:v] scale=320x320 [V];
   [2:v] scale=320x320 [O];
   [3:v] scale=320x320 [A];
   [4:v] scale=320x320 [R];
   [5:v] scale=320x320 [T];
   [a][E] overlay=shortest=1:y=200         [b];
   [b][V] overlay=shortest=1:x=320:y=200   [c];
   [c][O] overlay=shortest=1:x=640:y=200   [d];
   [d][A] overlay=shortest=1:x=960:y=200   [e];
   [e][R] overlay=shortest=1:x=1280:y=200  [f];
   [f][T] overlay=shortest=1:x=1600:y=200  [g]"
```

Whilst this may seem somewhat involved, the invocation is quite straightforward: the input letters are passed to a complex filter that scales and labels each letter, whereupon a series of composite overlays position the scaled input letters accordingly. It is a trivial matter to automate the process of generating such a command sequence for any number of letters, scaling, and relative positioning. Figure 4 shows a single frame from an animation of the text "EVOART" in the font Cantarell-Bold rendered using this process.

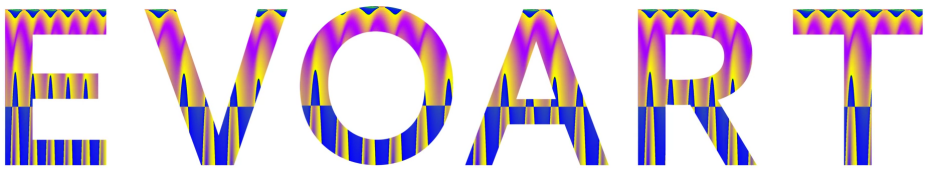

Fig. 4. A single frame from animated text "EVOART" in Cantarell-Bold font, where every animation is the same for each letter.

In the animation frame shown in Figure 4, all of the glyphs are mapped to the same animation. Whilst this effect may be desired by the artist or designer, it is perhaps more interesting to vary the animation applied to each glyph used in the sequence so that the combined typescript animation has a little more life to it. Figure 6 shows an animation frame taken from constructed alphabet where where each glyph has been animated in a different manner to the others. In this case the evaluation range was modulated. Other approaches are to tweak different nodes, or tweak more than one node, or change other evaluation constraints.

Figure 6 shows an animation frame using this alphabet where the word "EVOART" has been rendered. Fully animated versions of Figures-4 and 6 can be viewed online [2].

Fig. 5. A single frame from an animated alphabet in Cantarell-Bold font, where every animation is slightly different for each letter.

Fig. 6. A single frame from animated text "EVOART" in Cantarell-Bold font, where every animation is slightly different for each letter.

6 Discussion and Conclusion

The ability to animate typefaces with aesthetically pleasing functions is useful for artistic and promotional purposes. It gives text a high quality and lifelike feel that would be painstakingly hard to achieve using manual animation techniques. After the initial cognitive filtering that yields the input functions, the process amends itself to a high degree of automation. Given the limitless supply of diversity available within a given AE system, this entails limitless diversity and customization for animated typefaces and script.

Beyond the immediate artistic merit of animating aesthetically evolved images, at a deeper level there is something potentially very powerful here. An animated function consists of a base function and a sequence of parameterized mutations. The sequence of parameter values is a time series, and thus we can think of an animated function phenotype as being the projection of a time series into a high-dimensional space.

In machine learning (ML), Cover [4] showed that pattern classification problems are more likely to be linearly separable when projected non-linearly into a high dimensional space. This idea is exploited by a wide range of ML techniques from the seemingly artificial support vector machine [3] through to the biology emulating Liquid State Machine neocortex model [5].

Typically, when animating a function as described in this paper, the parameterization sequence is chosen to be a simple geometric sequence with no external meaning, but this need not be so, as was illustrated in the musically driven example shown online [2]. We can extend this idea therefore to *any* time-series

data, including those which are bound to meaningful classification and prediction problems.

Ergo, subject to sufficient funding, in future work we plan to explore the potential of AE functions to the application of time-series classification and prediction tasks. The reason this is particularly interesting is that the AE functions have been preselected by an existing form of intelligence, and thus it is interesting to ask to what extent this intelligence is embedded or correlated with the selected functions and their applied computational power. Exploration in this direction might start by evaluating the Lyapunov exponent [16] for some time series for functions taken from increasingly greater generations, i.e images that are increasingly "aesthetic", to see if there is a correlation between that which is aesthetically interesting and that which is computationally powerful.

The very fact that the created animations have structure, that is, that our brains can distinguish earlier frames from later frames *at all*, indicates clearly that the input information is not destroyed upon projection, strongly implying underlying computational power. This makes the prospect of research in this direction extremely exciting, and we look forward to observing the results.

References

1. Ffmpeg, http://www.ffmpeg.org
2. Multimedia examples of the artifacts described in this paper, http://www.evoart.club/evomusart2016
3. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning 20(3), 273–297 (1995), http://dx.doi.org/10.1007/BF00994018
4. Cover, T.: Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. Electronic Computers, IEEE Transactions on EC-14(3), 326–334 (June 1965)
5. Maass, W., Natschläger, T., Markram, H.: Real-time computing without stable states: A new framework for neural computation based on perturbations. Neural Computation 14(11), 2531–2560 (2002)
6. Machado, P., Cardoso, A.: Nevar - the assessment of an evolutionary art tool. In: Proceedings of the AISB'00 Symposium on Creative and Cultural Aspects and Applications of AI and Cognitive Science, Birmingham, UK (2000)
7. Martins, T., Correia, J., Costa, E., Machado, P.: Evotype: Evolutionary type design. In: Johnson, C., Carballal, A., Correia, J. (eds.) Evolutionary and Biologically Inspired Music, Sound, Art and Design, Lecture Notes in Computer Science, vol. 9027, pp. 136–147. Springer International Publishing (2015), http://dx.doi.org/10.1007/978-3-319-16498-4_13
8. McCormack, J.: Aesthetic evolution of l-systems revisited. In: EvoWorkshops. pp. 477–488 (2004)
9. Merritt, L., Vanam, R.: x264: A high performance h. 264/avc encoder (2006), http://neuron2.net/library/avc/overview_x264_v8_5.pdf
10. Mills, A.: Evolving aesthetic images. MSc Mini Project Thesis (2005), https://www.ashleymills.com/ae/EvolutionaryArt.pdf
11. Nikolov, N.: Organo font landing page, http://logomagazin.com/organo-font/
12. Noé, A.: Matroska file format (2009), http://www.matroska.org/files/matroska.pdf

13. Rooke, S.: Eons of genetically evolved algorithmic images. Morgan Kaufmann Publishers Inc. (2002)
14. Sims, K.: Artificial evolution for computer graphics. Computer Graphics 25(4), 319–328 (July 1991)
15. Todd, S., Latham, W.: Evolutionary Art And Computers. Academic Press (1992)
16. Wolf, A., Swift, J.B., Swinney, H.L., Vastano, J.A.: Determining lyapunov exponents from a time series. Physica D: Nonlinear Phenomena 16(3), 285–317 (1985)