Teaching a machine beauty

Intelligent interactive evolution of abstract animations

MSc thesis
Written by
Wouter Deenik

Supervisors:
dr. M. Poel
dr. ir. D. Reidsma

August 2017
Human Media Interaction

UNIVERSITY OF TWENTE.

Abstract

Generative art in the form of animations is used more and more in the current screen-filled world. If these animations would be generated with as much freedom as possible, some problems arise. More freedom means more 'ugly' animations are generated, and an ugly animation can get unpleasant to look at, in contrast to an ugly image. Also, to steer the algorithm towards better animations, human feedback is needed, but giving feedback on animations takes time. A system is proposed and developed which can generate diverse abstract animations, while minimizing the number of unpleasant animations in its output to (partly) solve the stated problems. This system uses a modified version of an algorithm developed by Karl Sims [1] to render the animations, a genetic algorithm to improve those animations and a neural network acting as a filter to reject unpleasant animations from the system's output. Feedback is gathered using a custom developed feedback interface.

The system is evaluated and the performance of the filtering component determined in terms of precision, recall and F score in several different tests. When a system without a neural network to reject unpleasant animations, and one with such a network are compared, the system with neural network shows significantly less unpleasant animations. The precision of the neural network in the different tests tends to be around 0.75, which means around one out of four animations that pass the filter is given the lowest rating by users. Without a filter, the precision of the system is on average around 0.66: one in three of the shown animations would get the lowest rating. This shows an increase in precision when a neural network is used to filter the animations. Some good animations are rejected too, but since the system has a near infinite 'pool' of animations to draw from, this is not considered as a big problem. Several different neural networks are tested, but only the learning rate seems to matter in terms of performance. However, the learning speed seems to depend much on the dataset used for training the neural network. 50-100 animations seem to be the minimum number to train an untrained network to a performance close to the 0.75 precision observed in other tests. Several suggestions are made on how to improve the system.

Table of contents

1 Introduction[1]

Generative art, the type of art that is a result of some computer program being left to run by itself, with minimal or zero interference from a human being, has been around since not that long after the first programmable computers emerged [2]. Although the name 'generative art' suggests that the art is generated completely by a computer, this is often far from what happens. The algorithm is still designed by a human and the inner workings, limits and constraints of this algorithm determine what the resulting art will look like. The art itself is generated by the algorithm, but the 'blueprint' of that art is made by the 'artist'[2] behind the algorithm, and thus the artist has a certain indirect influence on the art. It depends on the algorithm how big this influence is, but it is always there. This influence is often a necessity though, since an algorithm with complete freedom in what it displays will have a very low probability of producing pleasing art. This can be compared to the famous infinite monkey theorem, where an infinite number of monkeys behind typewriters for an infinite amount of time will eventually write the complete works of William Shakespeare by chance. Similarly, an algorithm without any constraints will surely be able to produce art that is considered beautiful by people, but the probability will be so low that it will practically never happen.

Some quite impressive attempts have already been made to make a generative art system with as much freedom as possible. One of the first ones, which also was one of the first generative art systems around since computers were a thing, was AARON [3]. Harold Cohen worked on this system for years, and it could in a way come up with its own paintings. In the beginning the involvement of Cohen was quite substantial; since he initially could not find a good way for the system to give colour to the paintings, he colourized them himself by painting them himself. Later, the system could also choose the colours, but it never became fully autonomous; Cohen always selected the art that was best. He phrases this problem well in 'Color, Simply' [4]:

[1] Some parts of this section are based on a similar text from 'User feedback for a genetic algorithm generating animated art'. This paper can be found in Appendix B.

[2] There is an interesting discussing about who can be called the artist in cases like this; the programmer of the algorithm, the algorithm itself or the computer running the algorithm. More about this can be read in an article by Boden [6]. For clarity, in this document the programmer is seen as the artist.

“I was trying to write a program that could function as an artist. Not an artist’s tool, a kind of proto-photoshop, that I could use to make art – after twenty years as a professional artist I already knew how to make art – but as an autonomous entity capable of generating original artworks; as autonomously, at least, as I could figure out how to make it. I’ve been trying ever since. Autonomy isn’t an absolute, of course, but given that AARON – that’s the name of the program I started then – makes most of its images at night, while I’m asleep, the program is obviously more autonomous than it was. But its autonomy doesn’t extend to exercising judgment about what it’s doing, and exercising judgment myself the next morning isn’t easy. Which of the hundred or so images should I print and which should I discard? They’re all good enough to print.”

Note that Cohen says here that all produced art is good enough to print. He worked for decades on AARON and over the years substantially changed the rules and constraints of the algorithm. The high quality of the generated art can be largely accounted to Cohen’s fine-tuning of these rules. The style of art that the algorithm generates is also very dependent on these rules, and this is visible in the art the system generated; over the years it changed visual styles due to the changing rules, but for each period the style of the generated artworks is similar, because the rules contained in the algorithm are similar. You can see this in Figure 1 and Figure 2.

There are several other algorithms that can generate art independently, but they all need someone to indicate what looks good and what does not, although the amount of human involvement differs per algorithm. Karl Sims developed one where the involvement is reduced to just selecting artworks [1]; there are almost no rules or constraints in the algorithm itself, but the human involvement is still needed. This makes sense, as a machine has no concept of beauty. Some examples of the results of Sims’ algorithm can be found in Figure 3.

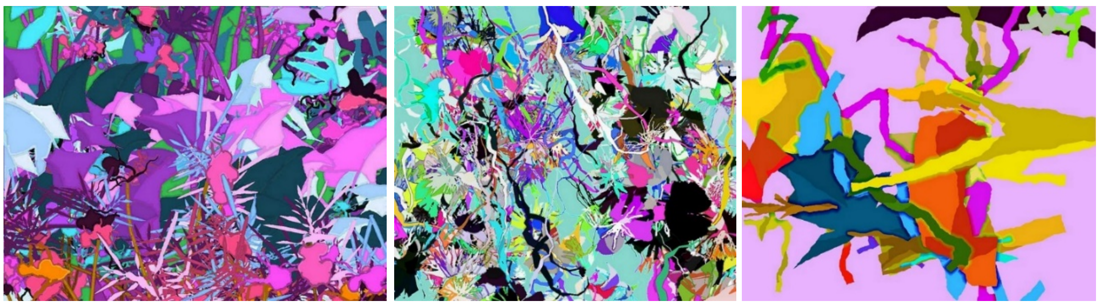


Figure 1: Three different paintings generated by AARON. Years of creation from left to right: 2004, 2008, 2010[3]

Figure 2: Two paintings that are from the same period as the painting in the middle of Figure 1[3]

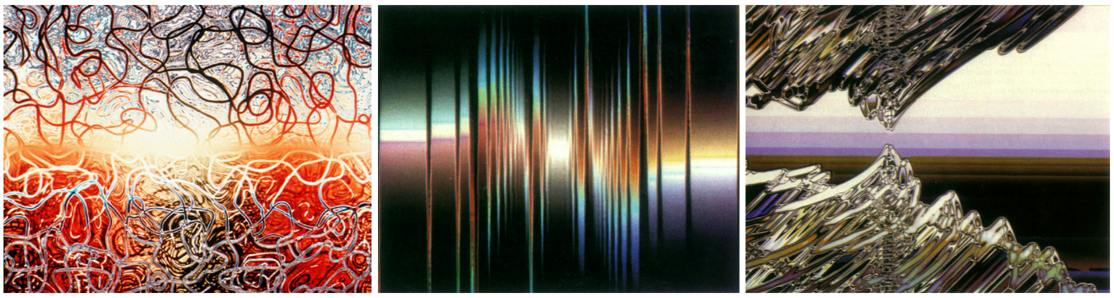


Figure 3: Three examples of the output of Sims' algorithm [1]

Generative algorithms like this are found more and more in digital art installations. The abundance of screens and other digital media nowadays makes it easy and relatively cheap to set up an installation or even a website that shows generative art. These kinds of installations also start to show more and more animated art, as it is often quite easy to animate generative art. The computing power of modern computers and especially GPUs makes it possible

[3] Source: AARON's home (http://www.aaronshome.com/aaron/aaron/gallery/FS-main-galleryS4.html)

to generate the same artwork more than 20 times a second while some of its parameters are changed over time, resulting in animations.

An algorithm that can generate animated art with a lot of diversity could make semi-permanent art installations that are on display for months or even years much more interesting, as the content they show can vastly differ over time. An algorithm like that can also be very useful for artists to explore art and get inspired by completely different art than they are used to see; a computer is not bound to certain ideas and can come up with visuals that are totally different than what human artists would create.

However, as mentioned at the beginning of this section, a larger diversity in the generated art also means a higher probability of producing unwanted art. This is also the case for animations, and the impact of unwanted animations will probably be even bigger than with static images. Where with images the worst ones will at most look boring or ugly, animations can get very unpleasant to look at when they contain a lot of flickering, for example. The involvement of humans to tell the algorithm what looks good and what does not is thus even more important for animations. However, with animations it will probably also take more time to give this feedback. Where with images you can show more than 100 at once on a screen to be able to tell the algorithm which ones look good or bad, it would be much more difficult to judge several animations this way. The amount of different movements will probably be quite overwhelming and it would therefore be difficult to give each animation a fair judgement. The strain of displaying all these animations will also be higher for the system, possibly limiting the framerate of animations, which can change how they look. Furthermore, since animations contain a time element, it is difficult to scan through animations and give feedback quickly. Each animation needs to be observed for at least a couple of seconds to see how it behaves, before a judgement can be made. This will probably slow down the process significantly compared to static images.

These are the main design problems faced when building a system that can independently generate diverse animations. The global aim of this project is to design a system that can independently generate a vast diversity of abstract animations, while minimizing the problems stated in this section. The system should also improve the generated animations to increase the probability of generating pleasant animations. To do this, this system needs a couple of components:

1. An algorithm that can generate and display a vast diversity of abstract animations.
2. A component that retrieves user feedback on each animation (since the system cannot make judgements about its own output, as mentioned at the beginning of this section).
3. An algorithm that will improve the generated animations using this feedback.
4. A component that can recognize the most unpleasant animations and can filter those out of the output of the system before it is shown to the user.

The focus of this project will be on this last component, since this will play the biggest role in solving the problems stated in the previous paragraph. This can all be summarized in the first (design) question of this project:

> **RQ1:** "How to design a system that generates a vast diversity of abstract animations, while minimizing the amount of unpleasant animations in its output?"

Since the focus of this project is on the filtering part of the system, the quality of the designed system will be tested on the amount of unpleasant animations in its output. This leads to the second research question of this project:

> **RQ2:** "What is the performance of the filtering component in the designed system in terms of precision, recall and F-score?"

To answer these questions, first past research towards user feedback methods and art preference is discussed. Next, the design process of the system is described per component. Finally, the user tests and their results are discussed and conclusions are made.

2 Context

In this section, some studies towards factors that influence people's preference in art and user feedback methods for preference are discussed. Next to this, the state of the art of generative algorithms for animations is discussed. These subjects are already studied in two earlier studies: 'Finding machine intuition criteria for generating aesthetically pleasing animated art' and 'User feedback for a genetic algorithm generating animated art', which can be found in respectively Appendix A and Appendix B. The relevant pieces of these studies (mainly the review of other studies towards the subjects) will be used here and the results and conclusions of the studies will be summarized here.

2.1 Preference in art

2.1.1 Static art

Several studies found that the complexity of generated images correlated with subjects' preferences of the images [5] [6] [7]. These studies used random polygons that differed in the amount of sides (more sides were interpreted as a higher complexity of the polygon), or visuals made by a random walk (where the length of the walk was interpreted as the complexity of the visual). Participants were asked which visuals they preferred. The results suggested that people have a preferred amount of complexity; preference increased with complexity up to a certain point, after which preference decreased again. Later studies used non-representational art instead of generated images. These studies used a small panel of participants to judge the complexity of each artwork. Results showed that preference increased with complexity, but did not find a very clear decrease of preference at the highest complexity ratings. It was suggested that this might be because the artworks with the highest complexity were simply not complex enough, which made it not possible to notice this effect [8] [9]. The latter study also found that the complexity of artworks positively correlates with the amount of times subjects would look at the artworks. This was to be expected since there are more visual stimuli in the artwork, so it would take more time to process everything visually.

Although studies using generated random polygons seemed to find a clear relation between complexity and preference, a study by Martindale found a different relation (a monotonic function instead of an inverted-U function) when the test circumstances were slightly changed [10]. They argued that the shape of the polygons did probably not just measure complexity but also other variables. Further experiments indicated that meaningfulness had a bigger influence on preference than complexity. Meaningfulness is interpreted here as how strong of a mental connection a viewer of the art can make to concepts he knows. For example: a cloud that looks like a cat is probably more meaningful to a viewer than a cloud with a much more random shape.

However, there is also ambiguity in art, which can be described as how many things a viewer might see in art. This makes that meaningfulness and ambiguity are quite related in art. Most artworks with a high ambiguity will have a low meaningfulness and vice versa. Speaking in cloud's terms again: the cat-shaped cloud has a higher meaningfulness but at the same time is less ambiguous to a viewer since the similarity to a cat predominates other interpretations of the shape. The random-shaped cloud probably has a higher ambiguity since it is easier to see multiple similarities to known shapes in it, but these similarities are less strong and thus the cloud has a lower meaningfulness. Like with complexity, research was done on a relation between ambiguity and preference in art. Jakesch and Leder performed a study that showed that subject's preference and interest towards artworks was highest for artworks with a medium amount of ambiguity [11]. Both artworks with higher and lower ambiguity were rated significantly lower on both interestingness and preference. This shows that there is probably a preferred level of ambiguity in art.

Several studies found a link between preference and colour. Ambiguity and liking ratings were found to be higher for coloured artworks compared to grayscale artworks [11]. Furthermore, the more prototypical colours were, the higher the preference was found for those colours [12] [10]. This might indicate that artworks using prototypical colours are preferred over artworks using less prototypical colours.

The type of art also seems to matter. Vartanian and Goel found that representational art was preferred over abstract art [13].Another interesting find in their study was that normal artworks were preferred over filtered (blurred) artworks. This might be linked to earlier found correlations between preference, complexity and meaningfulness; a blurred artwork will lose complexity and meaning.

2.1.2 Animations

A study by Bartram and Nakatani on how motion is perceived showed that motion features like fast speed, an angular movement shape (vs. curvy), obtuse angles and NOT smooth motion were associated with negative terms such as angry, painful, threatening, disgust, rejecting, urgent, fear, and annoying [14]. Features like slow speed and a curvy motion shape were associated with calm terms such as reassuring, calming, unimportant, relax, boring, and relieved. This indicates that fast motion, motion that changes direction abruptly and jerky motion might not be preferred by people.

The study 'Finding machine intuition criteria for generating aesthetically pleasing animated art', which can be found in Appendix A, tried to find out what factors of abstract animation influenced people's preference for these animations, and in what way. Several animations were created that differed in speed, the predictability of their movements and whether they were blurred or not. People could watch and rate the animations online. Overall, the preference for slower moving and more predictable animations was slightly but significantly higher. People also tended to prefer non-blurred animations but this seemed to differ based on the complexity of the animations.

2.2 User feedback methods

2.2.1 Feedback scales

Research on preference feedback often focuses on recommender interfaces for things like movies, music or books on the internet [15]. These are interfaces on websites where people can rate these products to give other people an idea of what is good and what is bad. There are different ways in which this feedback is gathered, mainly differing on the scale. Facebook started with its famous unary scale, where people can only give 'thumbs ups'. Other common scales are binary (positive/negative), 5 or 10 point (often stars/half stars) or a 100-point scale. Another method that is less common works differently than the previous mentioned scales, because it presents the user with two options. The user must then choose the best option of those two. Previous studies already tested which of these scales is best in several scenarios and according to several criteria.

These studies can be used to find the criteria that determine if a feedback method is good or not and are discussed in the following two sections.

2.2.2 Reliability

It is possible to (accidentally) introduce a bias in feedback because of certain design choices for the feedback interface. Friedman and Amoo [16] mentioned different causes for a possible bias. Connotations can bias the results because the negative connotations might sound more negative to the participant than the positive connotations sound positive. This also applies to numeric values; there is a difference between having a scale range from -5 to 5 and from 0 - 10. People are more likely to use the lower end of the scale in the last case since the lower end is perceived more negative in the first case. Forcing a choice can bias the results if the amount of undecided people or people without an opinion is significant. Unbalanced scales, meaning there are more positive than negative points on the scale (or the opposite) can have a big influence on results. The order in which the scale is presented (does it start at the positive or negative side) also influences the reported answers, but it is difficult to say which one of the two has the smallest bias. This may also depend on the subject of the question. Garland [17] found that the absence of a mid-point on a scale can distort results, although it was not possible to tell which scale was most accurate.

The amount of points on the scale can also have an influence on the reliability of the scale. Too few points means a loss of information, and too many points increases variance while accuracy may not increase significantly. Friedman and Friedman [18] studied this problem and recommended the range between five and eleven points as most optimal. Friedman and Amoo [16] stated that this also depends on what the scale will be used for. If there is no reason to think participants will have a complicated opinion about something, even a three-point scale might suffice.

2.2.3 Usability

Besides affecting the reliability of the scale, the amount of points also has an influence on users' behaviour and satisfaction. Sparling and Sen [19] studied how different scales (unary, binary, five-point and 100-point) influenced the time it took users to rate an item, their cognitive load, and overall satisfaction with the scale. They found that overall the rating time of users increased with the amount of points on the scale. Cognitive load was estimated by measuring the reaction time of users on a secondary task: clicking a button as soon as it becomes red (on random intervals). The cognitive load was found to be lower for a unary scale compared to other scales, but differences between the other scale were not significant. Users preferred the binary and five-point scales, and liked the five-point scale best.

2.2.4 Animations

The study 'User feedback for a genetic algorithm generating animated art', which can be found in Appendix B, investigated if a binary or 5-point scale was preferred by people if used to indicate preference for abstract animations. People could rate several abstract animations generated by an algorithm which used a genetic algorithm to evolve the animations based on the user's feedback. After rating 30 animations with each scale, participants were asked which scale they preferred and why. The results were inconclusive, but the explanations given seemed to indicate that usability in terms of mental effort and the amount of freedom people had to express their opinion were key factors in their preference.

2.3 State of the art

In this section, some existing algorithms that generate animations are discussed.

2.3.1 Electric Sheep

A well-known algorithm is Electric Sheep, a project that uses distributed computing to render animated art that is generated using a genetic algorithm [20]. Each frame in an Electric Sheep animation is a fractal flame, a member of the Iterated Function System (IFS) class of fractal algorithms [21]. The algorithm allows a big diversity of visual animations, although they still have a distinct visual style. An example of two fractal flames can be seen in Figure 4. Fractal flames are animated into Electric Sheep animations by varying some parameters of the fractal flame over time.

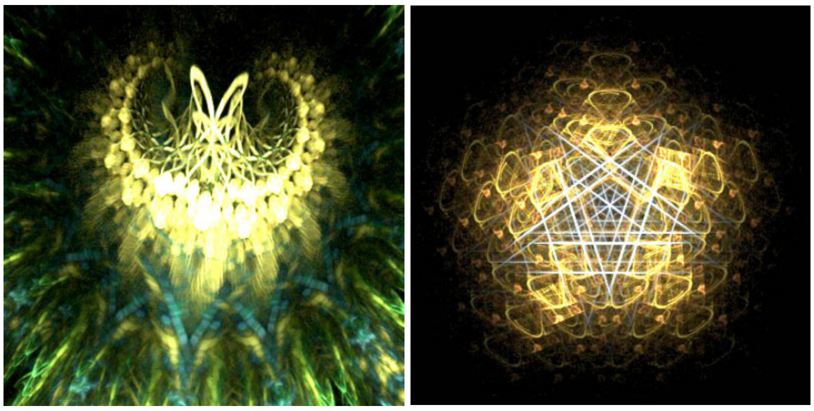


Figure 4: Two fractal flames [21]

Each animation can be rated by users all around the world using a binary scale; each user can vote an animation 'up' or 'down'. A genetic algorithm mates and mutates the animations in the current 'population' to create a new population, so the animations evolve over time based on user feedback. Low rated animations have a lower chance of mating in the genetic algorithm, so low rated animations might 'die out' after a while, while high rated animations have lots of offspring. Besides these genetic operators, the algorithm can add new randomly generated animations and users can add animations they designed themselves to the population.

Iterated Function Systems take quite some computing time to render, and an animation would need at least 20 fractal flames per second to be animated smoothly. That is why Electric Sheep uses distributed computing to render the animations into short videos, that are then distributed over the network. However, rendering fractal flames fast enough to render animated ones in real-time is possible nowadays using the GPU [22].

2.3.2 Milkdrop

Another existing algorithm that generates animations is used as music visualization in the music player Winamp[4]. It is called Milkdrop[5] and uses audio wave forms, shapes, spatial transformations and fragment shader programs to render its animations. Unlike Electric Sheep, Milkdrop animations are made by people programming so-called 'presets'; it does not have a genetic

[4] http://www.winamp.com/

[5] http://www.geisswerks.com/milkdrop/

algorithm to automate this process. Milkdrop presets can be rendered in real-time. An example of the output of a Milkdrop preset can be seen in Figure 5.

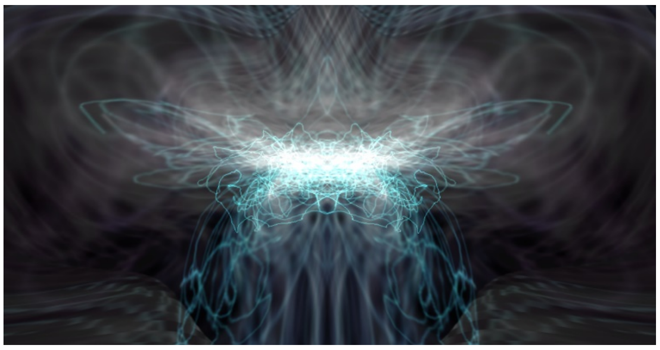


Figure 5: A Milkdrop preset (Inkblot by Geiss)

2.3.3 Self-developed

During the study 'User feedback for a genetic algorithm generating animated art', which can be found in Appendix B, an algorithm was developed that was loosely based on Milkdrop, but it used a genetic algorithm to generate the animations. It used several layers of shapes and several image transformation functions to form animations that could range from simple to quite complex. An example can be seen in Figure 6.

Just like with Milkdrop, the animations could be rendered in real-time. Users could give feedback for each animation using the keyboard in a binary or 5-point scale.

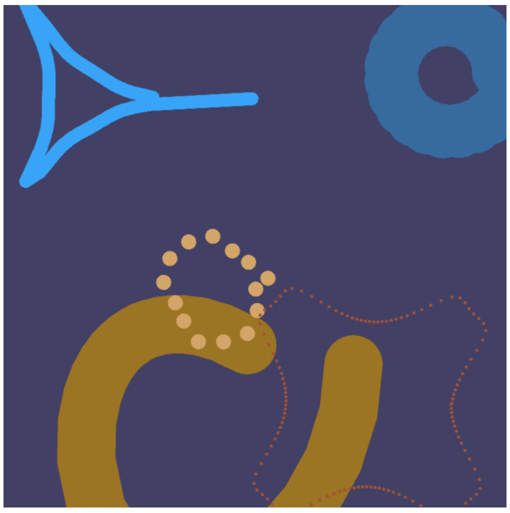


Figure 6: A frame of an animation generated by the self-developed algorithm

2.3.4 Karl Sims

Karl Sims developed a genetic algorithm that generates different images (not animations) by changing its own code based on human feedback [23]. The algorithm builds recursive lisp expressions using different functions. These expressions are executed for each pixel, resulting in an image. The lisp expressions can generate complex images with a very big diversity, since the value of each pixel is determined separately, so the algorithm is not bound to certain shapes or colours. Some examples of the output of Sims' algorithm can be seen in Figure 3.

The algorithm, as Sims developed it, does not generate animations, but he suggests several methods that are easy to apply to animate the output; the simplest one being the use of time (the number of seconds elapsed, for example) as an input variable in some of the lisp functions of the expression.

The images are evolved using a genetic algorithm, like the Electric Sheep algorithm. It seems like Sims could choose from a grid of generated images which ones would be used to generate the next 'generation', but he does not explain this process in detail.

2.3.5 Differences

The main differences between the algorithms can be described in terms of the animations they (could) generate, how the animations are designed, and how much computing power is needed.

Animations

The type of animations the algorithms generate are very different: Electric Sheep animations are very diverse, although the animations most of the times look like some form of geometry that glows on a black background. They are very abstract. Milkdrop can generate very diverse animations that can range from very abstract to semi-representative animations that are made to look like something, or use distinct shapes such as circles and squares. The self-developed algorithm mainly uses shapes and thus is never completely abstract. Its user test showed that people thought there was a lack of diversity in the animations. Karl Sims' algorithm could generate very diverse animations, although mostly abstract. Unlike Electric Sheep, these animations do not seem to have a common visual style, which makes this a promising algorithm for the to-be-developed system, since RQ1 states that a vast diversity of animations is needed.

Design

Electric Sheep are mainly generated by a genetic algorithm which adjusts a parameter set, but can also be designed by people. Milkdrop animations are exclusively designed by people, since the animations are partly a written program. The animations in the self-developed algorithm are designed by a genetic algorithm which adjusts parameters in a parameter set. The animations from Sims' algorithm are also designed by a genetic algorithm, but in this case animations are altered by adjusting the rendering program itself, rather than a parameter set which is used by that program to render the animation.

Computing power

Both Milkdrop and the self-developed algorithm do not require a lot of computing power, and their animations can be rendered in real-time on most modern computers. The traditional rendering method of Electric Sheep however, needs a lot of computing power and cannot render most Electric Sheep in real time. However, the method developed by Lawlor [22] is a promising method that could make it possible to render Electric Sheep in real time. Sims' algorithm can be computationally intensive, but this could be improved if it is possible to run the algorithm on the GPU. This should be the case, since Sims' algorithm performs the same calculation on each pixel of the output image; which is exactly what GPU's are designed to do.

3 Design & Implementation

This section describes the design and implementation of the system. It describes the main problems and goals for each component, the design choices made to reach these goals or solve these problems and a description of how this was implemented. This project was implemented in C# and GLSL.

3.1 Main goal & requirements

The main goal of the system is to independently generate a vast diversity of abstract animations, while minimizing the number of unpleasant animations it shows. It should also improve the generated animations. This is translated into the global requirements stated below. Some requirements have several sub requirements to clarify the global requirement.

REQ1. **The system should generate visibly moving animations.**

1.1 The output of the system should be animations; sequences of images where each image changes a bit, which translates to an illusion of motion when the images are shown in rapid succession.

1.2 The animations should be visibly moving; the changes over time should be big enough for the user to clearly see that the output is not a static image.

REQ2. **The generated animations should be diverse.**

2.1 The algorithm that generates the animations should not be bound to certain shapes or colours.

2.2 There should be no clear common visual aspect between the generated animations. An example of such an aspect is the common visual style between Electric Sheep animations: they all seem to consist of some sort of glowing geometry.

REQ3. **The system should gather feedback on the animations from users.**

3.1 The feedback method should be easy to use.

3.2 The feedback method should allow the user to express their opinion sufficiently.

3.3 The feedback method should give the system enough information about each animation to be able to improve the animations and reject unpleasant ones.

REQ4. **The system should improve the generated animations.**
The system should suit the generated animations to the preferences of the user; generate more animations towards the user's likes, and less animations with elements the user dislikes. This should translate to better rated animations over time.

REQ5. **The system should keep on exploring new animations.**
Since we are not looking for one global optimum (the best animation of all possible animations), but rather want to constantly show different animations, the algorithm should not converge to one 'best' animation, but rather keep on looking for other good animations.

REQ6. **The system should filter out unpleasant animations.**
The system should minimize the amount of animations that receive the lowest possible feedback by learning what kind of animations receive a low feedback, and filtering those animations from its output. This should result in less low rated animations, and speed up the evolution process, since the filtered animations do not have to be rated by the user.

3.2 System architecture

The architecture of the designed system can be found in Figure 7. It consists of four main components. The task of each component and which requirements it should fulfil is shortly described in this section.

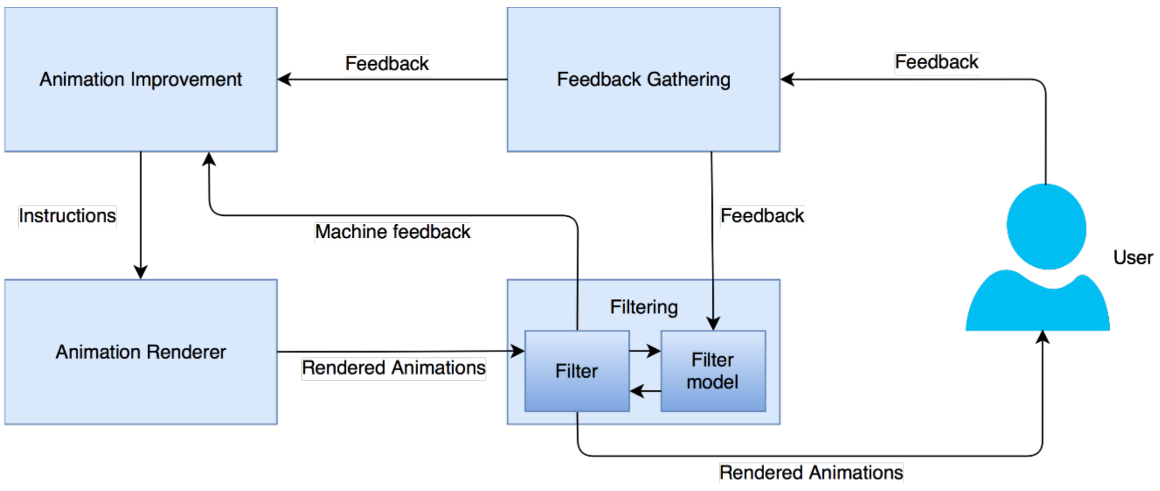


Figure 7: The system's architecture

3.2.1 Animation renderer

This component renders the animations using an algorithm such as the ones described in section 2.3. It takes 'instructions' from the animation improvement component, which tell how to render each animation. These instructions can be parameter sets, or almost complete programs which are executed to render the animations. What form these instructions take is dependent on which algorithm is used for the animation rendering.

Renderer requirements

- REQ1: The system should generate visibly moving animations.
- REQ2: The generated animations should be diverse.
- (REQ4: The system should improve the generated animations.)

The last requirement is not directly related to this component, but the rendering algorithm should support being 'steered' by instructions from the animation improvement component. It is therefore important to keep this requirement in mind when choosing a rendering algorithm.

3.2.2 Animation improvement

The animation improvement component takes feedback from the user or the filter to improve the animations. In short, it makes more variations of animations rated as being good, while discontinuing 'bad' animations. It does this by modifying the instructions it gives to the rendering component.

Improvement requirements

- REQ4: The system should improve the generated animations.
- REQ5: The system should keep on exploring new animations.
- (REQ2: The generated animations should be diverse.)

The last requirement is relevant to this component, because, as can be read in section 3.4, some techniques for improvement are prone to generate lots of similar animations, which would decrease diversity of the generated animations.

3.2.3 Feedback gathering

This component gathers the feedback from the users and forwards it to the animation improvement component and the filter model.

Feedback requirements:

- REQ3: The system should gather feedback on the animations from users.

3.2.4 Filtering

The filtering component is the focus of this project. It builds a filtering model by learning from the user feedback it receives on the shown animations. This model classifies each rendered animation as a 'bad' or 'good' animation, where 'bad' means that it is expected to receive the worst possible rating when shown to the user. If an animation is classified as 'good', the filter does nothing and the animation is shown to the user. If the animation is classified as 'bad', the filter gives the worst possible feedback to the improvement component, and the animation is not shown to the user. In short, it rates the animation on behalf of the user, so the user does not have to see the bad animation.

Filter requirements:

- REQ6: The system should filter out unpleasant animations.
- (REQ1: The system should generate visibly moving animations.)

The filter also rejects all animations that do not contain enough motion to be visibly moving. That is why REQ1 is also related to this component.

3.3 Animation renderer

The choice for an algorithm to render the animations is discussed in this section. The chosen algorithm and its implementation in the system are then explained in more detail.

3.3.1 Choosing an algorithm

To make a choice on what algorithm to use for the rendering of the animations, the algorithms discussed in 'State of the art' (Section 2.3 on page 13) are compared. More details about each algorithm can be found there.

In section 2.3.5, the differences between four existing algorithms are described. To make a fair comparison, the differences described there in terms of the generated animations, the design process of these animations and the computing power are translated to the requirements set at the beginning of this chapter. This means the algorithms are compared here in terms of the diversity of the generated animations, the ease at which the animations can be modified by the system, and the computing power needed to render the animations. The computing power needed is not part of the requirements, but since the differences can be quite big, this factor is also considered in the decision. An overview of how each algorithm scores on these factors can be found in Table 1.

Table 1: Overview of differences in algorithms

Algorithm	**Diversity**	**Modification by system**	**Computing power needed**
Electric Sheep	+/-	+	-
Milkdrop	+	-	++
Self-developed	-	++	++
Karl Sims	++	+	?

As can be seen, all algorithms except for Sims' algorithm have at least one disadvantage over the other algorithms. However, the computing power needed for Sims' algorithm is unknown. Since it can probably run well on modern GPU's, and it is superior in the diversity of the generated animations, this algorithm was chosen to be used in the system.

3.3.2 The algorithm explained

The visual algorithm will be explained in short here. More details can be found in 'Artificial Evolution for Computer Graphics' by Sims [1]. This is also where the information and images in this section come from. Note that only the visual part of the algorithm will be discussed here; the evolution part will be discussed in section 3.4. Sims' algorithm works with a function set where each function works on a per-pixel basis. An expression is built by starting with one of these functions, and filling the arguments of this function with constants, vectors, variables such as X and Y coordinates, or another recursively generated expression. This results in a large recursive expression, which if executed on each pixel, can generate complex images. The function set consists of basic functions like *+*, *-*, *mod*, *sin*, *log*, but also more advanced noise generation functions or functions that use neighbouring pixels to determine gradients, for example. Sims gives some examples of relative simple expressions and the associated images to explain the concept:

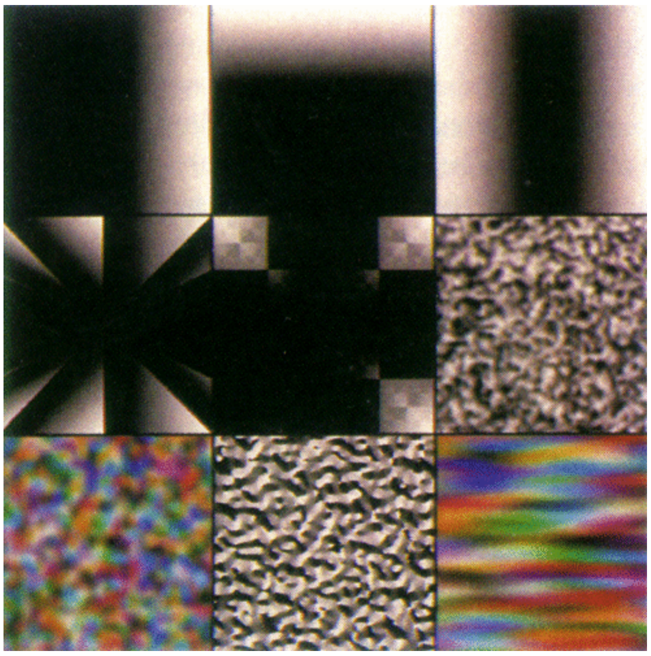


Figure 8: Some examples of results of simple expressions in Sims' algorithm

The LISP expressions associated with the above images (from left to right, top to bottom) can be found below. Note that the X and Y coordinates range from -1 to 1.

a. X
b. Y

c. (abs X)
d. (mod X (abs Y))
e. (and X Y)
f. (bw-noise .2 2)
g. (color-noise .1 2)
h. (grad-direction (bw-noise .15 2) .0 .0)
i. (warped-color-noise (* X .2) Y .1 2)

The fractal-like shapes in Figure 8e are a result of using a bitwise operator on floating-point numbers (X and Y coordinates in this case). The difference between normal noise generation functions and their warped versions is that the normal versions always use the X and Y coordinates as input parameters for the noise generation, where in the warped versions these input parameters can be given. This makes it possible to warp the generated noise, as can be seen in Figure 8i, which is the same noise as in Figure 8g, but with a scaled X coordinate as input parameter.

The above described expressions are relatively simple, which makes it possible to see the relation between the expression and the resulting image. However, more complex images are a result of complex expressions, and the relation between the expression and the result are hard to see. An example of this can be seen in Figure 9. This image was rendered using the following expression:

```
(sin (+ (- (grad-direction (blur (if (hsv-to-rgb
(warped-
color-noise #(0.57 0.73 0.92) (/ 1.85 (warped-color-
noise x y 0.02 3.08)) 0.11 2.4)) #(0.54 0.73 0.59)
#(1.06
0.82 0.06)) 3.1) 1.46 5.9) (hsv-to-rgb (warped-color-
noise y (/ 4.5 (warped-color-noise y (/ x y) 2.4 2.4))
0.02 2.4))) x))
```

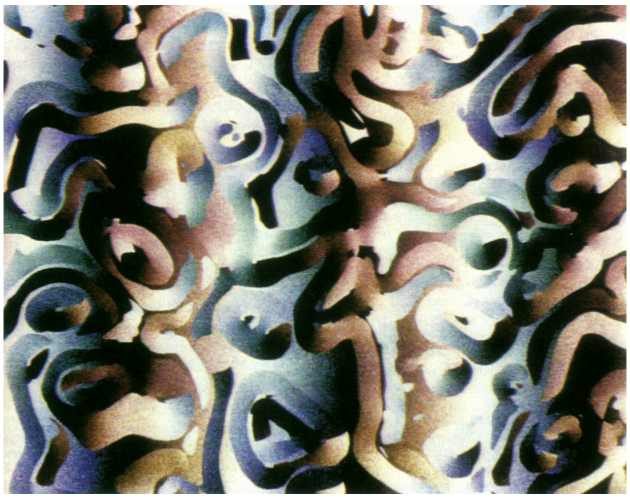


Figure 9: A more complex result of Sims' algorithm

Sims suggested different ways to animate the images:

- Adding a variable 'TIME' that can be used in the expressions just like X and Y coordinates are used.
- Using existing animated images as input in the expression.
- 'Dissolving' two expressions. This requires the expressions to have a similar structure, so the identical parts can stay the same while the differences are interpolated.
- Altering the mapping of the X and Y coordinates to create panning and zooming effects.
- Experimenting by hand, for example by interpolating parameters in the expression.

3.3.3 Implementation

The algorithm lends itself well for implementation in fragment shaders on the GPU. Fragment shaders are essentially little programs that are executed for each pixel of the output texture, and return the colour that that pixel should have. A system was developed that can translate the 'instructions' received from the improvement component into shader programs written in GLSL. These instructions are in the form of trees, where every node is either a function from the function set with its arguments as children, or a variable.

Function set

First, the function set was determined. This function set can be found in Appendix C, including the types of arguments each function takes, and basically consists of three types of functions:

1. **Float functions:** functions that output floating point values.
2. **Vector functions:** functions that output three-component vectors.
3. **Image functions:** Functions that act on an input image: they use the values of neighbouring pixels to determine their output (which consists of a three-component vector).

The functions are a mix of functions that are already present in GLSL and functions that Sims used. Since GLSL supports vectors and floating-point variables, the parts of the expressions that only contain the first two types of functions can basically be copied into one shader program without any problems. However, this approach does not work for image functions. These functions need to know the colour values of their neighbouring pixels, which means the image on which this function is executed needs to be rendered first. Therefore, the subtree that renders the input image of such a function is executed in a separate shader program. Once that image is rendered, it is passed to the shader program which executes the image function. The output of that shader program is then passed to the shader program containing the 'parent' function of the image function.

Iterated Function Systems

The only exception for the above described method is the function that calculates an Iterated Function System (IFS). This is a function that does not need the information of neighbouring pixels, but still runs in a separate shader program, because the rendering of such an image works differently than the rendering of the other functions. Since this function is quite different than other basic functions, and the results of an IFS can differ greatly depending on the implementation, a short explanation is given here.

Since Sims does not explain what kind of IFS his algorithm uses, and the Electric Sheep algorithm (which also uses IFSs) was also considered a pretty good option for the visual algorithm, fractal flames were used as IFS.

Generally, IFSs are systems with a set of linear transformative functions that, when alternatively and iteratively executed, form complex shapes. The fractal flame algorithm also contains non-linear functions and several other modifications that together make the resulting images very complex and give them a three-dimensional feeling. More details about this algorithm can be read in Draves' and Reckase's article on the subject [21].

Fractal flames are normally rendered using a method called the *chaos game*. This is a computationally heavy method that does not lend itself well for execution on the GPU, as we would like to do here. However, Lawlor came up with a different method of rendering fractal flames that is well suited for execution on a GPU. Details on the implementation can be found in Lawlor's article on the subject [22].

A simple version of this method is implemented here, which only uses functions that can be well implemented in a fragment shader program. These functions (of which examples can be found in the appendix of 'The Fractal Flame Algorithm' by Draves and Reckase [21], they call them 'variations') are:

- Linear
- Sinusoidal
- Spherical
- Swirl
- Horsehoe
- Polar
- Heart
- Hyperbolic
- Julia
- Bent
- Fisheye
- Exponential
- Power

It was decided that a maximum of five different functions can be used in one IFS function, because that already gives a lot of possibilities for different fractal flames.

Displaying the animations

The animations do not necessarily contain periodic elements, which means that quite some animations will only move during the first couple of seconds; until the time values used in their trees go outside certain ranges. For example, if the red channel is connected directly to the time parameter, the red value of all pixels will increase during the first second (since colour is coded in a range from zero to one in GLSL), and then stay at the maximum, since after that the time value is always greater than one. Preferable, the animations should animate longer than a couple of seconds, so it was decided that the first ten seconds of the animations are skipped, also in the analysis done on them for the filtering component. Animations like the one described above will then contain no movement during analysis, and can be rejected by the filtering component.

Two examples of simple animations with small function trees, generated by this algorithm, can be seen in Figure 10, and their function trees in Appendix F.

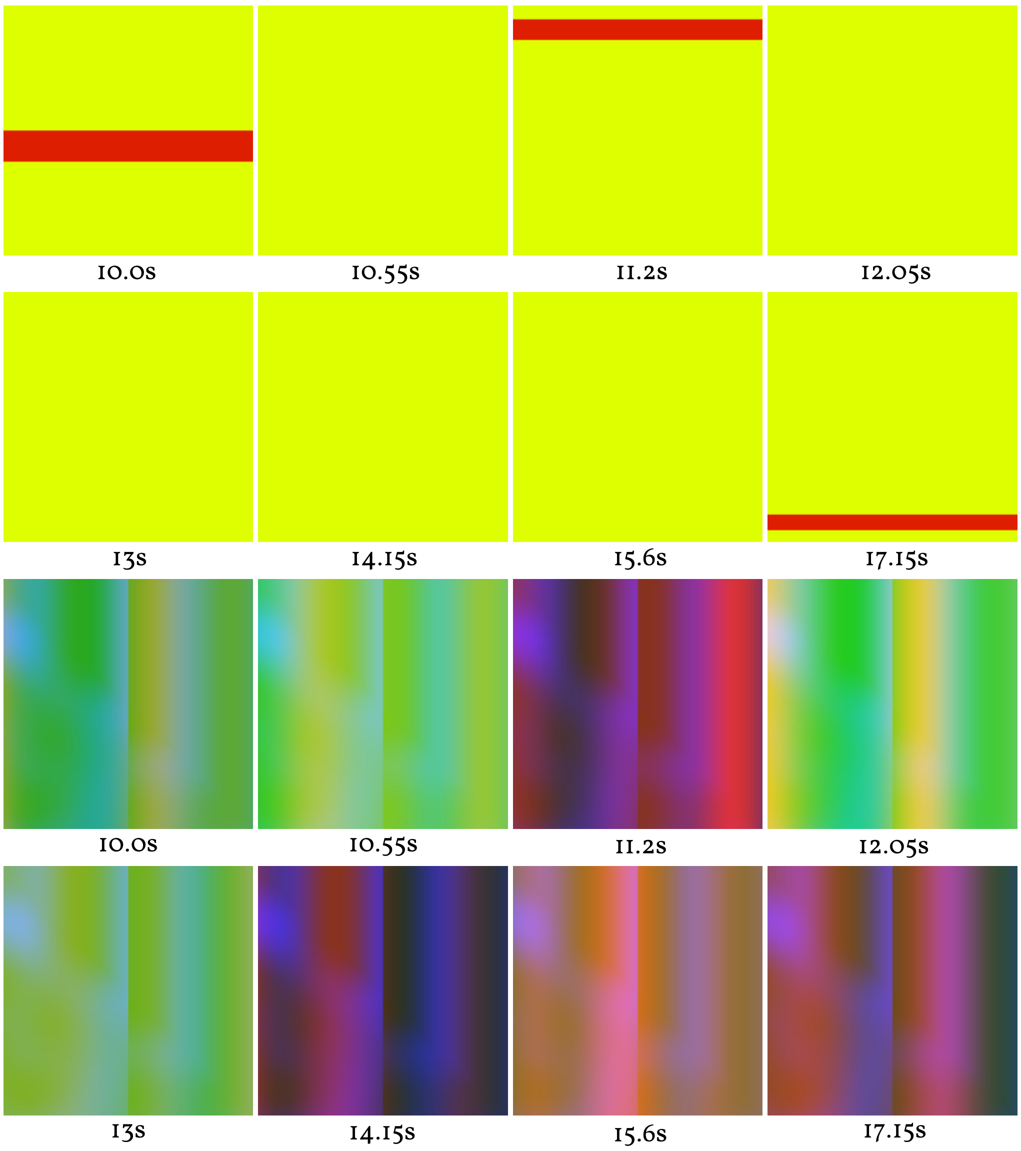


Figure 10: Two example animations with time codes

3.3.4 Reflection

A reflection is done on the requirements of this component. The relevant requirements for this component are:

- REQ1: The system should generate visibly moving animations.
- REQ2: The generated animations should be diverse.
- (REQ4: The system should improve the generated animations.)

REQ1

This requirement has two sub requirements. The first says the system needs animations. This was accomplished by modifying Sim's algorithm to output animations instead of static images, as it was originally designed to do.

The second sub requirement states that the animations should be visibly moving. It is possible that by chance the system generates animations without any 'TIME' nodes in its function tree, or the 'TIME' nodes are only found in parts of the tree that are not expressed in the final animation. This will still result in a 'static image'; an animation that does not contain any motion. Also, it is possible that an animation does contain motion, but it is so subtle that a human viewer will not see this as motion. The filtering component is designed to filter these cases from the output of the system. More details on how this is implemented and further reflection on this requirement can be found in section 3.6.

REQ2

This requirement states that the animations should not be bound to certain shapes or colours, and there should not be a clear common visual aspect between them.

Since the functions of the rendering algorithm are executed per pixel, and can work freely on all three colour channels, the algorithm is not bound to shapes or colours. The broad function set should give the algorithm enough freedom to render animations with very diverse visual styles.

If this actually results in diverse animations without any common visual aspect, is hard to measure. However, observing the animations generated for the different tests, as discussed in chapter 4, should give an idea.

REQ4

This requirement is related to this component, since the algorithm should support being ‘instructed’ by the improvement component. Since Sims himself used a genetic algorithm to improve the images, this should be the case.

3.4 Animation Improvement

The design and implementation of the animation improvement component are explained in this section. Since a lot of the design is inspired by the work of Sims and Rooke, there are quite some references to their work in this section. If referred to Sims' work, the information comes from his article 'Artificial Evolution for Computer Graphics' [1], unless otherwise specified. The same goes for Rooke's work; this comes from his chapter 'Eons of genetically evolved algorithmic images' in the book 'Creative evolutionary systems' [24], unless otherwise specified.

To improve the generated images, Sims used a genetic algorithm, which slowly evolved the images in the direction he indicated by selecting certain images. Steven Rooke later used an extended version of the algorithm, also with a genetic algorithm to improve the image. Both reported that the genetic algorithm worked well, although there are almost endless ways of using a genetic algorithm. Rooke experimented with several combinations of settings and methods for the genetic algorithm to increase its performance and reported his findings. Since a genetic algorithm seemed to work well for both Sims and Rooke, and Rooke's observations with different settings for his genetic algorithm are very useful, this project also used a genetic algorithm to improve the animations.

In this section the settings and implementation of this genetic algorithm is explained. It is assumed that the reader is familiar with the concept of genetic algorithms and common technical terms used when talking about these algorithms. If that is not the case, a short explanation of what a genetic algorithm does and the terms used to describe a genetic algorithm can be found in section 2.1 of 'User feedback for a genetic algorithm generating animated art', which can be found in Appendix B.

3.4.1 Structure and methods

As we are evolving a program here, we follow the workings of Koza on genetic programming [25], instead of an algorithm inspired by the work of Holland on genetic algorithms [26]. This means our genetic representation is a tree, where every node contains either a function with its arguments as children, or a variable or constant without any children. Trees are mutated by randomly regenerating a random branch. Crossover is done by swapping two randomly chosen subtrees between two trees. Examples of these operations on trees can be found in Appendix G. The process that Koza calls ‘encapsulation’ was also implemented but was not used in the end, to keep the genetic algorithm simple.

The fitness function is in practice not present, as the fitness of an animation is a direct result of the feedback this component receives for that animation. More information about how human feedback translates to a fitness value can be found in section 3.5, ‘Feedback gathering’.

3.4.2 Construction of generations

It is useful to first explain how the used genetic algorithm constructs generations, since most of the settings explained in the next section will influence this process.

At the start of a run of the system, and whenever a population is replaced, a new population is generated by making random function trees. These trees are constructed using a ‘grammar’ for functions, which is summarized in the function set in Appendix C. The grammar consists of all functions and the types of their arguments and output. For each argument of a function, a function is found that outputs a variable of the required type. In turn, that function’s arguments (if present) are filled the same way. To be sure that this recursive process ends, a maximum tree depth of five levels is set for these randomly generated trees. If an argument is needed of the ‘float’ type, there is a chance that a variable is used instead of a function. This can be a mathematical constant such as π or e, or a random number. Another possibility is that a parameter such as time, the x-coordinate, or the y-coordinate is chosen. The chance that a variable is chosen over a function gets bigger the closer the current level is to the maximum level.

There is a 75% chance that a tree is generated using the above described method. Otherwise, the tree comes from the crossover of successful animations from past populations. More about this is explained in section 3.4.3, subsection ‘Long-term memory’.

An animation is only added to the new population if it passes the filter in the filtering component.

After running for a while, the system will reach a points where it gathered feedback on all animations in a population. At that point, a new generation of animations is constructed for that population. First, the fitness values of all animations in the population are normalized, so their sum is exactly one. The new generation is then filled with animations by repeating the following process until the new generation is filled:

1. Choose between the following methods, where each method has a pre-set probability of being chosen: reproduction, crossover or mutation.
2. If crossover is chosen, two random animations are chosen from the old generation by using their normalized fitness values as probabilities of being chosen. Crossover is performed on these animations to form two new animations, which are placed in the new generation.
3. If one of the other two methods is chosen, one random animation is chosen with the normalized fitness values as probabilities of being chosen. The animation is reproduced or mutated, and placed in the new generation.

To improve the diversity of the animations in each population, animations that look the same within a generation are removed, so every animation is unique. Note that animations often look the same, despite the low chance of the genetic algorithm generating two or more duplicate trees. This happens because there are often parts of the tree that have no influence on the rendered animation. An example of this can be found in Figure 11: the ‘max’ function will always give the constant number as a result, as this is always bigger than the result of the cosine. This results in two duplicate animations, despite both having been rendered by different trees.

Therefore, the animations are compared using the actual rendered animations: each animation is partly rendered to extract some features which say something about the motion and colours in the animation. These features are further explained in section 3.6.2. Animations that generate the exact same features are assumed to look the same, which is used to find and remove duplicates within a generation. A check was done with a small batch of around 100 animations to see if animations with the same features looked the same as well. Animations with the same features were not distinguishable from each other.

After this process has been completed, the new generation will be presented to the user until all animations are rated and this process repeats to form a new generation again.

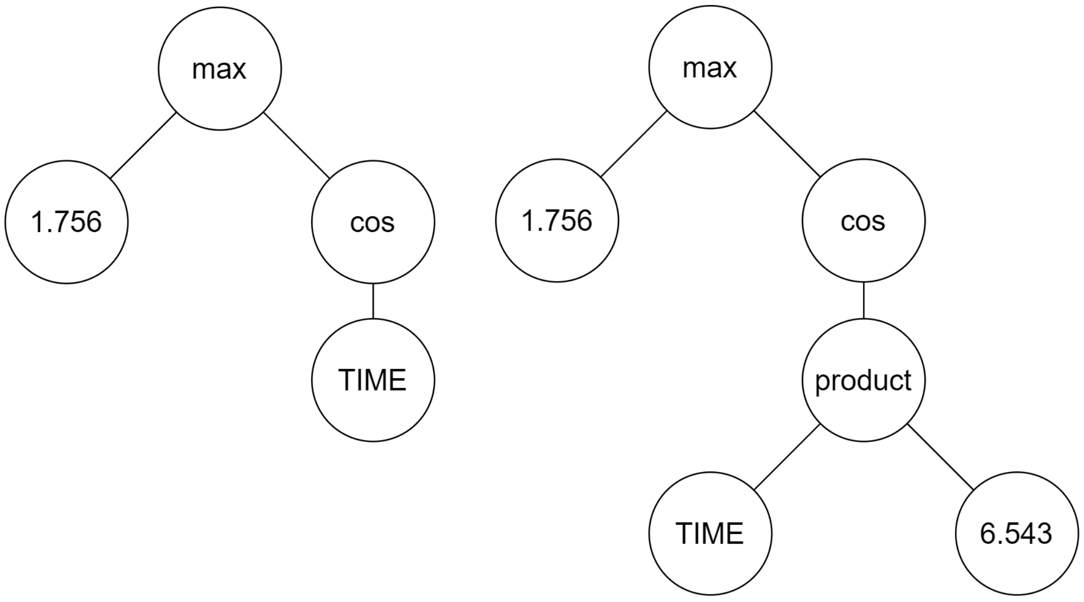


Figure 11: Two different trees that will generate the same animation

3.4.3 Settings

How a genetic algorithm behaves is very much dependent on variables like population size, probabilities of reproduction, crossover and mutation, and other possible modifications. It is therefore important to choose the right values to let the genetic algorithm behave the way you want. For example: a high probability of crossover will have a high chance of producing good animations, but reduces the diversity. For mutation, it is the other way around. A high probability of reproduction ensures that good animations will not be lost easily, but this also reduces diversity. A big population size increases diversity and probably the quality of the found animations, because it takes longer until the algorithm converges on a certain type of animation, and therefore has more time to explore. However, a big population size significantly impacts the speed at which the genetic algorithm goes through its generations and therefore the speed at which the animations will improve. The design choices on these variables and other implemented concepts that differ from a conventional genetic algorithm are explained below.

Population

Sims used population sizes of 20-40 images, Rooke used 100-200. This is small compared to other genetic algorithms, but most genetic algorithms have a fitness function that determines fitness automatically. This is often done by doing some calculations, running a simulation, or comparing the results of a program (in case of genetic programming) with the desired results. As Sims and Rooke had to give the feedback themselves, which is much slower than an automated process, they chose smaller population sizes. However, rating even 20-40 animations for each generation of the population is still a tedious process, which should be optimized as far as possible.

Sims and Rooke could look at multiple images at once to give feedback quite fast. This approach was tried with animations during the study 'User feedback for a genetic algorithm generating animated art', which can be found in Appendix B. Showing only two animations at once already seemed overwhelming, and it was thought that this would influence the feedback of the animations too much. This could result in animations receiving different ratings than if they were rated separately. Since rating animations would therefore take even more time than rating images, a population size of 20 animations was first experimented with. One of the most notable things seen was that, most of the times, in the first generation only one or two animations got a good rating, resulting in a second generation that was filled with variations on mainly those two animations. This reduced the diversity to such extend that an alternative had to be found.

A solution that works well in keeping the diversity high, is using multiple small populations. Each time a new animation needs to be shown, a random animation without a rating from a random population is chosen. This way, the user does not see similar animations all the time. Each population will still converge to a couple of good animations, but since there are multiple populations, the diversity is kept high. Using ten populations with each a population size of ten seemed to give a good balance between the speed of the genetic algorithm and the diversity of the shown animations.

The small population sizes, a new problem arises: because there are only ten animations in each population, on average only one animation gets reproduced. The chance that this is one of the better perceived animations is still not that big, and thus it is possible that good animations are ‘lost’, leaving the new generation with only worse animations. This happens all the time in genetic algorithms, and is normally not a big problem, because there are plenty of other good animations that are not lost. However, with populations this small, this is not always the case. It makes the probability of losing high quality animations quite big. Increasing the probability of reproduction would partly solve this problem, but there is still a chance that reproduction does not happen, or that only the worse animations are reproduced. Therefore, to make sure that quality is not lost, a mechanism was implemented that always reproduces the animations with the highest score. This comes at the cost of diversity, but in this case, it was decided that it was worth it for the increased quality of the animations.

Convergence

Since the algorithm should continuously show new animations instead of looking for a global optimum (REQ5), it needs to decide when a population converged enough. This means it should recognize when the population gets ‘stuck’ on a certain type of animation, so it can replace that population, and new animations will be shown.

Because duplicates are already removed from each new generation, convergence can roughly be measured by the size of a population. Therefore, when a population gets to a size of three or smaller, it gets replaced.

Long-term memory

The small size of the populations make that the populations converge quite quickly and thus the animations do not have a lot of time to evolve. This would break the genetic algorithm. A solution was found by implementing an idea that Rooke used in his algorithm, something he called a ‘Genetic Library’. Rooke would often ‘feed’ his first generations with successful images from past runs of the algorithm, stored in this library. This would give the algorithm a head-start and would increase the quality of the evolving images. A similar system was implemented in this system, although the process was automated. This allowed the continuation of the evolution process of successful animations from past populations. Since the process is fully automated here, it is referred to as a ‘long-term memory’ of the genetic algorithm.

The memory works by saving the best animations of a population before the population is replaced. Every time a new function tree is generated for a new population, there is a 25% chance that that tree is generated by performing crossover on two trees from this 'long-term memory' of past successful animations.

This way, there is a short-term evolution going on within each population until it is replaced, and a long-term evolution of the most successful animations that spans multiple populations and could go on for a long time.

Probabilities

The probabilities of reproduction, crossover and mutation are normally the main tools to adjust the behaviour of a genetic algorithm. However, the focus of this study is on the automatic recognition of unpleasant animations and the other factors in the genetic algorithm (explained above) seemed to have a bigger impact on the speed of improvement and the diversity of the animations. Therefore, for the final user test, these probabilities were chosen based on Rooke's findings, and were not extensively tested and compared with other possibilities. The probabilities for the final user test were as follows:

- Reproduction: 0.1
- Crossover: 0.45
- Mutation: 0.45

Koza argued that mutation was not necessary since crossover could give similar results [25]. However, Rooke suggested that certainly when a population contains relatively many new animations, mutation seemed to work better for him, while crossover worked better if there were more sophisticated animations present. Since each new population contains both new animations and more sophisticated animations from the long-term memory, the probabilities for crossover and mutation were given equal values. A small probability of reproduction was kept, despite the fact that the best animations always get reproduced. This ensures that slightly worse scoring animations can still reproduce.

3.4.4 Reflection

A reflection is done on the requirements of this component. The relevant requirements for this component are:

- REQ4: The system should improve the generated animations.
- REQ5: The system should keep on exploring new animations.
- (REQ2: The generated animations should be diverse.)

REQ4

A genetic algorithm is implemented to improve the animations. Genetic algorithms should, by their nature, improve the phenotypes (animations in this case) they are working on over time if their settings are chosen right. However, the implemented algorithm works differently than most conventional genetic algorithms, mainly because of REQ5. It is unknown what the impact of this difference is. If the implemented algorithm improves the animations and if so, how fast, can be seen in the results of the tests, discussed in chapter 4.

REQ5

The implemented genetic algorithm is designed to never stop exploring: a population is replaced after a certain amount of convergence is met. The way the convergence is determined is very rough and might not be the best way to do this. However, the used method still makes sure the algorithm does not keep on converging on one animation.

REQ2

A method was implemented to remove duplicate animations from each generation, preventing generations to be mainly filled with the same animations. This should improve the diversity. However, it is impossible to maximize diversity; a genetic algorithm needs variations of the same animations in order to improve the animations.

3.5 Feedback gathering

The feedback gathering component gathers the feedback for the displayed animation and stores it, so the animation improvement and filtering components can use this for their purposes. The design choices of the feedback interface are discussed here.

3.5.1 Scale

The rating scale is important for the feedback you get, as discussed in section 2.2.1. Using the studies in that section, the decision was made to use a five-point scale to rate the animations. It seems to have a good balance between ease of use and resolution. A five-point scale is also preferred by people in a study by Sparling and Sen [19].

However, the filtering component is designed to recognize the worst animations; those that people genuinely do not want to see. The genetic algorithm should not use these animations to fill new generations; otherwise there would still be a chance that this would result in one or more bad animations in the next generation, which is unwanted. On a scale form one to five, the 'one' rating could be used to recognize the worst animations. However, a score of one would imply that the animation still would have some amount of value in the genetic algorithm, while it is essentially thrown out. This could result in people getting a wrong idea of what a score of one would mean.

This problem can be solved in two ways. The first is making the worst possible score zero, as this would make clear the animation would have no value in the algorithm. The other solution is to introduce a separate rating that makes clear the animation will get thrown out: using a picture of a recycling bin, for example. This second solution is not used, because it is expected that people would be hesitant to knowingly throw an animation out, while this is necessary for the filter to work properly. The other solution can be applied by shifting the whole rating scale, so it ranges from zero to four, or by adding a separate zero rating, so the whole scale ranges from zero to five. The difference is mainly in the resolution that is left for the genetic algorithm to work with, since the 'zero' rating is not used in the genetic process itself. The choice was made to not decrease the resolution of the part of the scale that can be used for the genetic process. Therefore, an extra zero rating was added to the five-point scale, to create a scale reaching from zero to five.

3.5.2 Interface

During the development of the system, the feedback was given through the 0-5 number keys on a standard computer keyboard. However, for the user test another interface was needed, something more sturdy and more attractive to use.

A box was laser cut which holds six buttons, each with a number engraved below it. The box also contains a short explanation of what the system does. The box is connected to the pc through USB and sends the given rating over a virtual serial port once a button is pressed. The box is filled with little bags of sand and rubber feet are glued to the bottom to prevent it slipping away when people push a button. A picture of the box can be found in Figure 12.

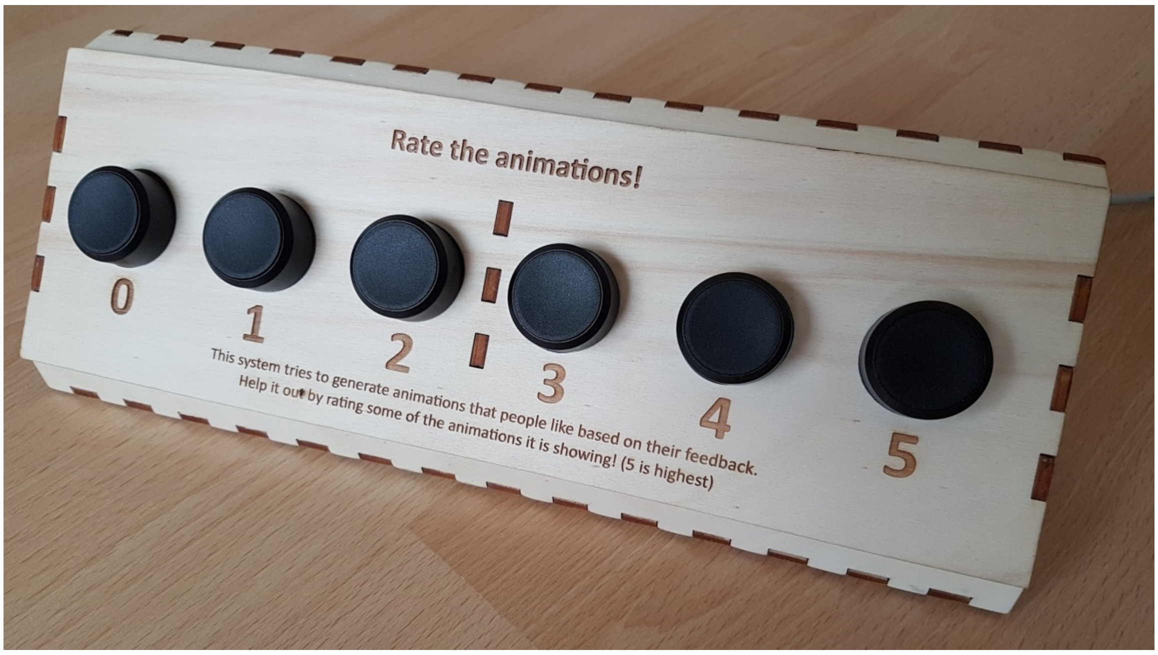

Rate the animations!
0
1
2
3
4
5
This system tries to generate animations that people like based on their feedback.
Help it out by rating some of the animations it is showing! (5 is highest)

Figure 12: The rating box

3.5.3 Reflection

A reflection is done on the requirements of this component. The relevant requirements for this component are:

- REQ3: The system should gather feedback on the animations from users.

The sub-requirements of this requirement asked for a feedback interface that is easy to use, allows the user to express their opinion sufficiently and gives the system enough information to improve and reject animations.

The design choices made are backed up by past studies into different feedback interfaces, and are expected to fulfil this requirement in terms of the needs of the user. An interface with big, sturdy physical buttons was made to be inviting and increase the ease of use. An extra ‘zero’ rating was added to the rating scale to allow the filtering component to distinguish between the worst animations and all others, without decreasing the rating resolution used by the genetic algorithm. This should give the system enough information to perform its task and fulfil this and the other requirements. However, if the system turns out to not fulfil all its requirements after testing, the influence the feedback interface could have on this will not be ignored.

3.6 Filtering component

This section describes the design and implementation of the filtering component.

3.6.1 Method

The filter consists of two steps: one rejects animations that contain too little motion, the other rejects animations that are predicted to get a 'zero' rating.

Since we have REQ1: "The system should generate visibly moving animations.", the filter is used to filter animations with too little motion. This is done using thresholds for different features (explained in section 3.6.2) that tell something about the motion in an animation. The filter is used for this since it already extracts these motion features for the second step, thus it can also easily filter animations with a lack of motion.

If an animation passes this first step (and thus should contain enough motion to be visibly moving), its aesthetic value to the user is predicted. In section 2.1, some studies towards human preference in art are discussed. Although some relations are found, there are no hard numbers that can be used for an algorithm to make a clear distinction between pleasant and unpleasant art. Furthermore, the features that are linked to preference are hard to measure: complexity and ambiguity. However, for animations, things like speed and predictability were linked to emotions and preference, which are easier to measure.

Still, the lack of a clear line between pleasant and unpleasant makes it hard to make this distinction; a simple threshold on some measurable values would not work. This problem therefore asks for another method to detect this difference: a method that can use human feedback to learn to recognize the distinction between unpleasant and pleasant animations. Therefore, a machine learning algorithm will be used.

This algorithm can make predictions about the rating each animation will receive. If an animation is predicted to get a low rating, the filter can give that low rating to the animation instead of the user. This way, the user does not have to see the animation; the animations get essentially rejected from the system's output.

3.6.2 Feature extraction

There are several machine learning methods that can use images as input data directly, but these are often mainly focussed on pattern recognition. Recognizing certain features that can predict human preference is a whole other problem though, it is unknown if a machine learning method could solve this problem this way. Furthermore, the machine learning algorithm should also be able to analyse motion in the animations, which means it should also look at differences between frames, for example. This would make these machine learning methods very complex to implement. This will get very complicated while it is still unknown if the algorithms will be able to accurately predict people's preference in animations.

Since past studies (discussed in section 2.1) already found some features that are linked to preference, it would be unnecessary to let the algorithm figure those relationships out by itself. It would be better to develop a method to extract relevant features from the animations, and provide the algorithm with only these features. This will probably increase the accuracy of the filter, while decreasing the complexity of the machine learning algorithm.

Choice of features

Past studies showed that complexity, ambiguity, colour use, speed, and predictability of motion can probably be used as indicators for human preference. Speed of motion and colour use are features that can be extracted well from the animations. Complexity may be estimated by using the colour data; an animation that consists of mainly one colour is less complex than an animation that uses the full colour spectrum. Ambiguity and predictability are much harder to extract from an animation.

However, some early tests indicated that the lowest rated animations can probably be already recognized by their colour use and speed of motion. Most of them were given a low rating because of flickering, fast movements, too little motion, or a lack of diversity in the used colours (which can also be seen as too little complexity). Therefore, it was decided to focus on motion and colour use for the extracted features.

Motion

To be able to detect motion, several frames of the animation are rendered. The differences between these rendered frames can then be calculated on a per-pixel basis and can be used to analyse the motion in the animation. Comparing two frames results in an image where each pixel's value is the absolute difference between the two frames for that pixel. This difference is calculated using the grayscale versions of the frames, since the colour is analysed separately and it decreases the time needed to analyse an animation. An example of some frames and their differences can be found in Figure 13.

It matters greatly which frames are rendered; if they are close together, the difference will say something about the short-term motion, while if there are several seconds in between the frames, the difference will say something about long-term motion. Both options are probably required to get accurate predictions. In the short-term difference, flickering and fast motion can be detected, while a lack of motion will be especially well visible in the long-term difference. The times that are used to render the frames for analysis can be found in the first column of Table 2. To avoid the case where animations with periodic movements with the right frequency are invisible for analysis, the time difference between the long-term frames increases slowly. The difference between each render time is based on prime numbers. The long-term difference is determined by comparing the first with the second frame, the second with the third, etc. The short-term difference is determined by comparing each frame with a frame 100ms later, as can be seen in Table 2.

Table 2: The times used for rendering frames for analysis

First frame (sec)	Long-term difference (sec)	Short-term difference (sec)
10.0	10.55	10.1
10.55	11.2	10.65
11.2	12.05	11.3
12.05	13	12.15
13	14.15	13.1
14.15	15.6	14.25
15.6	17.15	15.7
17.15	-	17.25

The difference between two frames could be summarized in one number. For instance, the average value of all pixels in the difference image. However, it might be useful to also have the minimum and maximum difference within the difference images. This will make it possible to distinguish between an animation where there is a lot of motion in one corner, and an animation where there is moderate motion on all pixels, for example.

The same goes for the minimum and maximum differences that are found between all analysed frames. For example: there is a big difference between an animation where the colour changes from black to full red gradually between 10 and 17.15 seconds, and an animation where the colour changes instantly from black to full red at the 15 second mark. However, the average difference over the frames will be the same for both animations. If the maximum and minimum difference between the frames is also given, the difference between the two animations can be seen: for the gradually changing animation the minimum is bigger than 0 and the maximum is small, while for the instant changing animation the minimum is 0 and the maximum is big.

Therefore, the minimum, average, and maximum differences are calculated both between and within the difference images. The first tells something about how the motion is spread over time, the second how it is spread over the 2D space the animation is rendered in. This results in nine features for motion. Since this process is done for both long-term and short-term motion, in total 18 features are extracted that tell something about the motion in an animation. This process is illustrated in Figure 13.

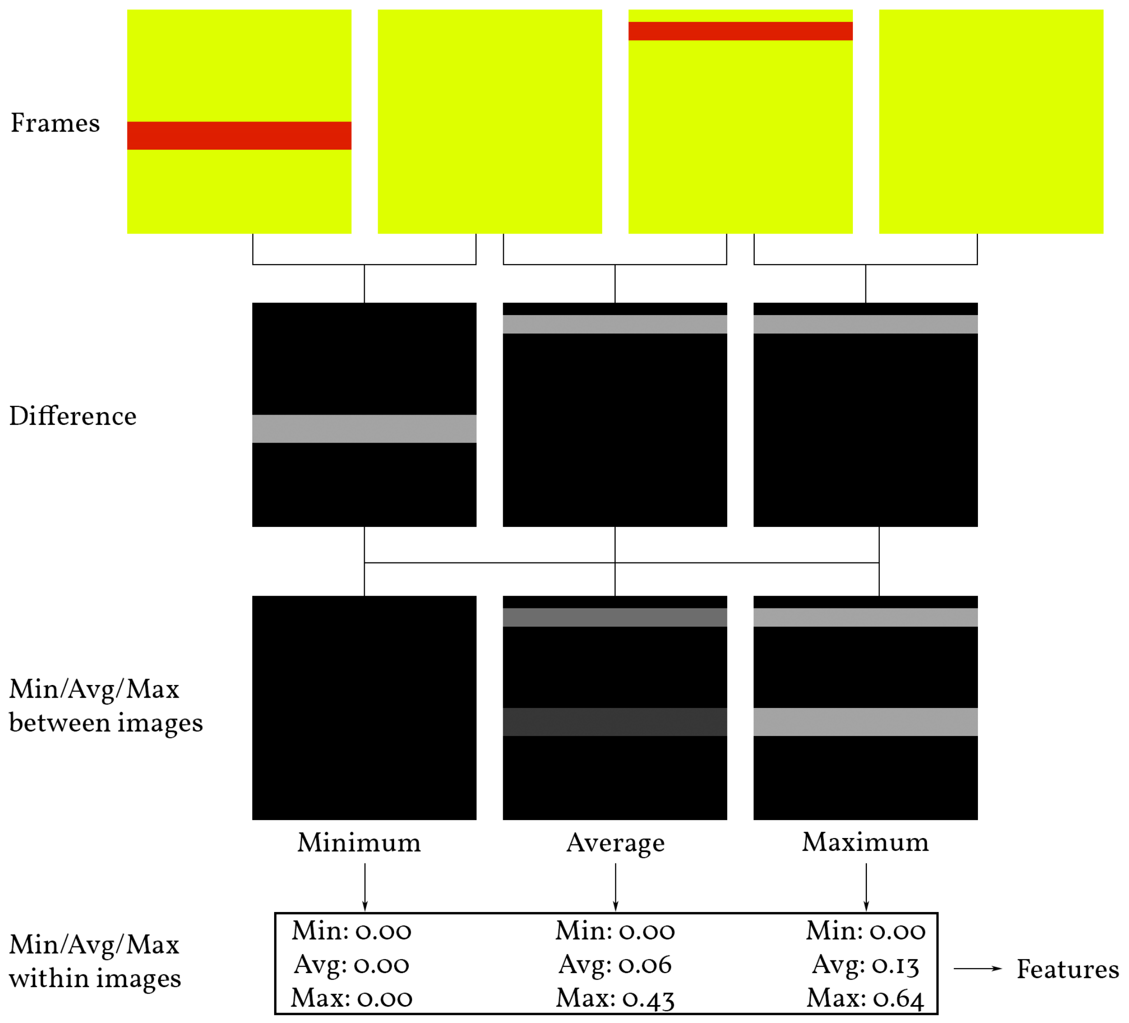


Figure 13: Example of motion feature extraction

Colour & complexity

To analyse the colours, the frames that are already rendered to calculate the long-term motion are used. The colours are analysed by their hue, saturation and brightness values, since this gets close to how humans perceive colour [27].

For each analysed frame, the number of unique colours and unique hue, saturation and brightness values are counted. Also, the mean and standard deviation of the hue, saturation and brightness values are retrieved per frame. For these ten numbers, the average over all frames is calculated. The number of unique colours and values of hues, saturations and values, and their means give an idea about colour use. The standard deviations are an estimation of (colour) complexity; if the standard deviations are big, the range of hue, saturation and brightness is big and thus the complexity of the animation in terms of colour is probably high.

Feature set

These analyses result in a feature set of 29 features: nine for long-term motion, nine for short-term motion, ten for colour, and one for the average render time of the frames. This last value can be used to reject animations that would take too long to render, as there is always a chance that a certain function tree has a combination of functions that would take several seconds per frame to render, for example.

All values in the feature set are normalized to fall in the range between zero and one.

3.6.3 Machine learning algorithm

Since the human feedback is subjective, can be inconsistent, and might change over time, a flexible machine learning method is needed. Also, the exact relation between the extracted features and human preference is unknown. Therefore, it seems like a good task for a neural network: neural networks can learn difficult problems that are hard to put into rules or other conventional arithmetic methods, can be quite robust against inconsistent input, and can learn online; it can learn from the same samples it is predicting on. Therefore, a neural network was implemented to learn which animations to filter from the system's output. Since the most important part of the filtering component is to filter the most unpleasant animations, the implemented neural network is used to classify animations between animations that receive a 'zero' rating and all other animations.

Neural networks

Since not all readers might know what neural networks are, this will be explained shortly. Neural networks are inspired by the way brains work. The network consists of several 'neurons', each with several inputs and an output. Each neuron translates its input values to an output value, often by calculating a weighted sum of the inputs, where each input has its own weight. A neuron can 'learn' by adjusting these weights based on how its output differs from the desired output. This 'error' of the output of the network is propagated through the network, so every neuron can adjust its weights. This learning algorithm is called 'backpropagation learning'. There are also other algorithms to 'teach' neural networks, but that goes beyond the scope of this project. The speed of the learning process can be adjusted by modifying the learning rate of the network. The learning process can also implement 'momentum', which means part of the weight changes done using a learning 'step' carry on to the next step, which can stabilize the learning process when there are big differences between succeeding samples.

A neural network can consist of several 'layers' of neurons, where (often) the inputs of the neurons in a layer are the outputs of all neurons in the previous layer. A network always has an output layer with the same number of neurons as outputs needed from the network. The other layers are called 'hidden layers'. A simple example of a neural network can be found in Figure 14. For overview, only the outputs of the neurons in the hidden layer, and the weights and output of the neuron in the output layer are shown here.

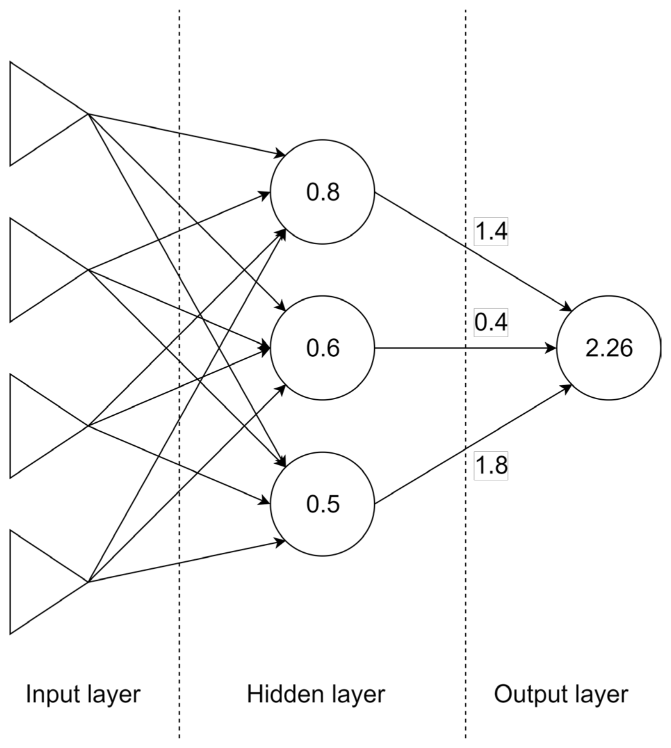


Figure 14: Example of a neural network

The implemented network

The implemented neural network uses the features from the feature set directly as inputs, and has one output, which is a number between zero and one. When this output value is bigger than 0.5, the animation is classified as an animation with a rating higher than zero. Otherwise, it is classified as an animation with a rating of zero.

Since it is proven that neural networks with one hidden layer can learn any non-linear function as long as it has sufficient neurons in the hidden layer [28], the network has only one hidden layer. In this layer, all neurons are connected to all inputs, and every connection has its own weight. Next to this, each neuron has an offset value, which can be seen as an extra input (with its own weight) that always has a value of one. The output of each neuron is a sigmoid function (Equation 1) of the weighted sum of all inputs (and the offset), and thus always has a value between zero and one. The neuron in the output layer works the same and has the outputs of the neurons in the hidden layer as inputs. The number of neurons in the hidden layer and the alpha value of the sigmoid function can be varied. Different values for these parameters are tested in chapter 4.

$$\frac{1}{1 + exp\left[-\alpha\left(\sum_{j=1}^{d} w_j x_j + w_0\right)\right]}$$

Equation 1: Sigmoid function of neurons. The x variables are the input values, w the weights (where w0 is the offset), and d the amount of inputs. α determines the steepness of the sigmoid function.

The true class of an animation is determined by the human feedback that is gathered by the system. This information is used to train the network, using a backpropagation algorithm with momentum. The learning rate of the network and the momentum can be varied. An untrained neural network starts with random weights between -0.01 and 0.01.

This basic setup leaves four parameters that can be varied to create different neural networks:

- Number of neurons in the hidden layer
- Sigmoid function alpha
- Learning rate
- Momentum

To determine what the differences in performance are between networks with different values for these parameters, a test was performed, which is described in section 4.3.

3.6.4 Reflection

A reflection is done on the requirements of this component. The relevant requirements for this component are:

- REQ6: The system should filter out unpleasant animations.
- (REQ1: The system should generate visibly moving animations.)

REQ6

This requirement dictates the main task of this component, and the focus of this project. The true reflection on this requirement is therefore done by performing multiple tests in the next chapter, and using the results of these tests to answer the second research question “What is the performance of the filtering component in the designed system in terms of precision, recall and F-score?” in the Conclusion on page 76.

REQ1

The filtering component is used to filter animations that are not (visibly) moving from its output. Certain rules are used to filter these animations based on the motion features extracted from each animation. These rules filter all animations which contain no motion in the analysed sections of the animation. If these rules also work well for filtering animations that only contain very subtle (and thus almost invisible) motion, is determined by looking at the observed reactions of users in Test 1, discussed in section 4.2. The chosen features are not perfect: if motion is only present outside the rendered frames, this motion is not visible. Also, since the motion is extracted from the grayscale images, changes in just the hue value of a pixel are not seen as motion.

4 Testing

To answer the second research question “What is the performance of the filtering component in the designed system in terms of precision, recall and F-score?” adequately, three different tests are performed:

1. A user test involving multiple users who use a system with a pre-trained neural network, and a system without a neural network.
2. A test comparing the performance of different neural networks.
3. A test which investigates how many animations are needed to train a neural network adequately.

These tests are performed to see how the designed system performs overall, and to find out how the behaviour and performance of the system changes when:

1. There are one or more users involved.
2. When the neural networks used in the filtering component differ.
3. When it is trained on a small number of animations.

4.1 Overview

First, an overview is given of the performed tests, and how they relate to each other. Each test works with one or more sets of animations, annotated with their rating, hereafter called ‘datasets’. These are also described.

Tests

Three tests are performed, in chronological order:

Test 1 A user test with multiple users and a pre-trained neural network. The first goal of this test is to test the whole system in the way it could be used in a public setting: with multiple users that are free to interact with the system as they would like. The second goal is to compare the influence of the neural network on the system’s behaviour. Two systems are compared: one with a neural network and one without. Furthermore, the ratings users give to animations on the system without a neural network are used to create a dataset which is used for the other two tests.

Test 2 A test comparing different neural networks.

The goal of this test is to find out the influence of the layout and learning parameters of the neural network on its learning behaviour and its performance. This is tested on datasets generated with the ratings of both one and multiple users.

Test 3 A test with one untrained neural network

The results of the previous test are used to choose the neural network used for this test. The goal of this test is to find out the minimum number of animations needed to train a network well enough to be of added value to the system. The test uses datasets rated both by one and multiple users.

Datasets

To avoid confusing over which test uses which dataset, the three used datasets (*DS1, DS2, DS3*) are described here.

DS1 Single user

Dataset 1 was annotated (rated) by myself. It contains 1030 animations, and is used to train the neural network used in Test 1, as well as for the single-user dataset of Test 2. The animations in this dataset were generated randomly, without the use of the genetic algorithm, to increase the diversity of animation types in this dataset. This was done using the method to fill new populations for the genetic algorithm, as described in section 3.4.2, but without a neural network for filtering, or a long-term memory. The relative distribution of ratings can be found in Figure 15.

DS2 Multiple users

This dataset is the result of the ratings of users on the system without neural network in Test 1, and is thus annotated by multiple users. It contains 2281 animations. This dataset is used in both Test 2 and Test 3. The relative distribution of ratings in this dataset can be found in Figure 16. Note that something went wrong with the rating box in Test 1, causing most 'one' ratings to be registered as 'zero' ratings. Furthermore, it is clearly visible that the genetic algorithm with long-term memory produced more high-rated animations than if the animations are generated randomly like in DS1.

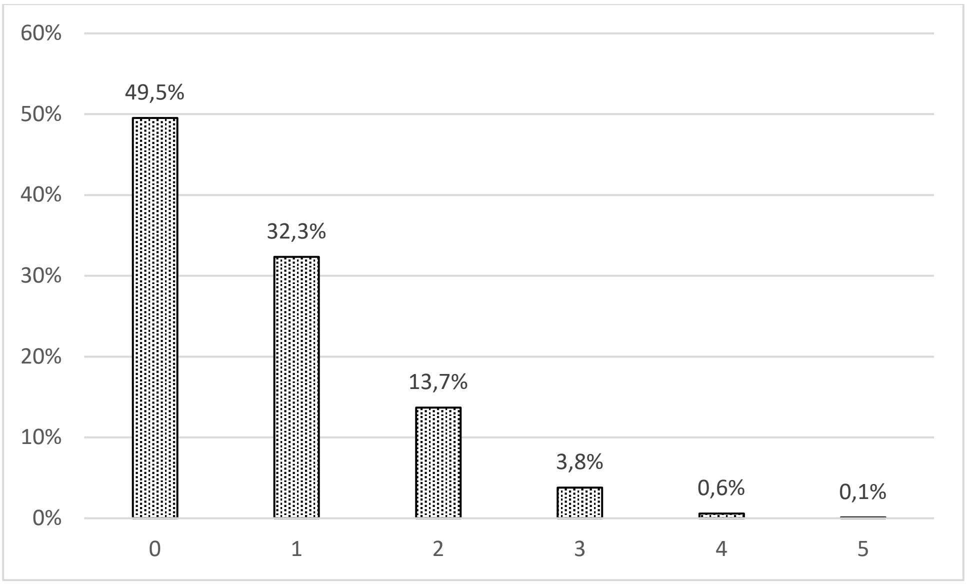


Figure 15: Relative distribution of ratings in DS1

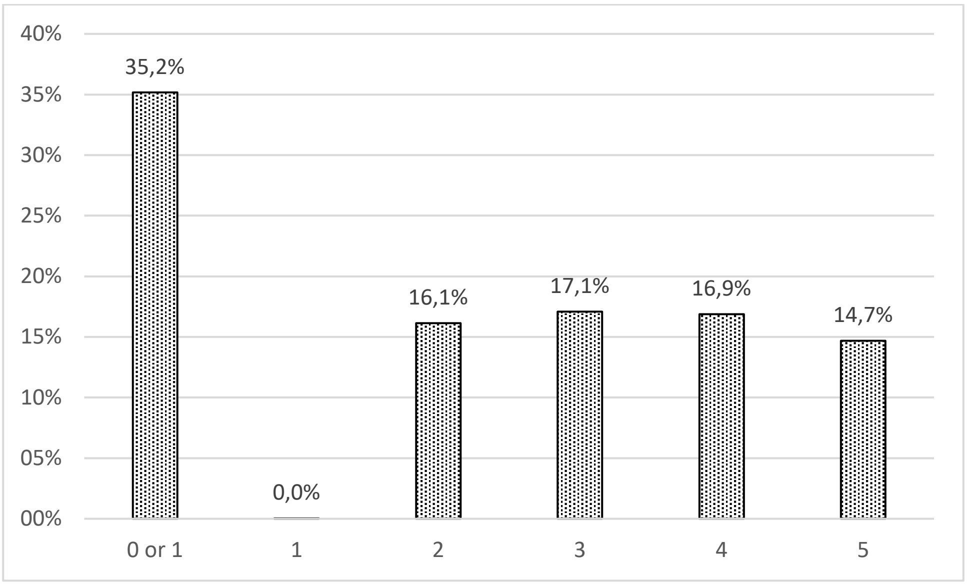


Figure 16: Relative distribution of ratings in DS2

DS3 Single user

Dataset 3 is not annotated by me, but by a male of 24 years old. It consists of 402 random animations from DS2, re-rated by this person. This was done this way to make sure the quality of the animations in both DS2 and DS3 is similar, as these datasets are both used in Test 3. The relative distribution of ratings in this dataset can be seen in Figure 17.

Furthermore, since the animations were rated in a random order and the ratings should be rather consistent (as they are all rated by the same person), this dataset could be used to see if the genetic algorithm actually improved the animations over time. This could not be seen in DS2, as the ratings are always relative to what the users are used to see; a certain animation might be rated better if the previous animations were bad than if the previous animations were good. An overall slow improvement over time would therefore be invisible in that dataset. Therefore, a moving average (10 samples) of the 402 ratings in DS3, in the genetic order of the animations, can be seen in Figure 18. There is no clear improvement visible.

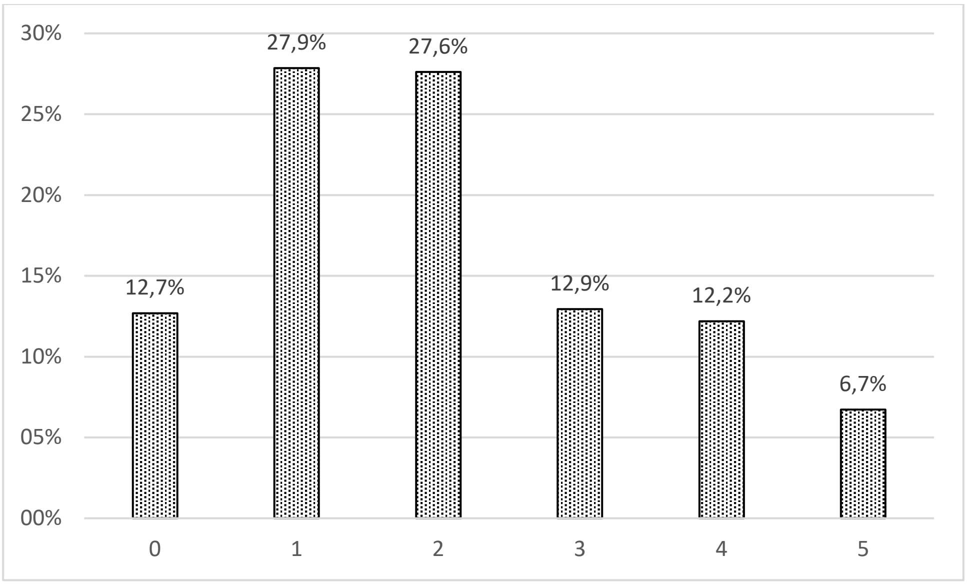


Figure 17: Relative distribution of ratings in DS3

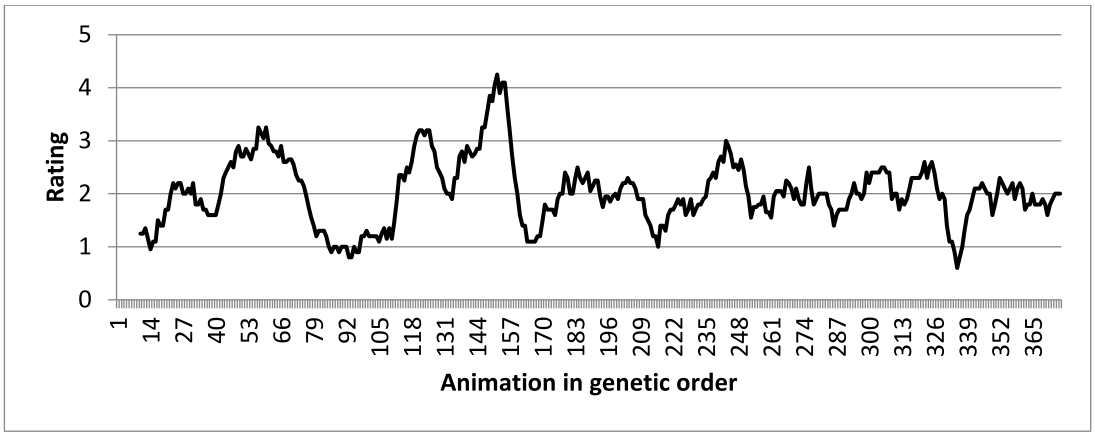


Figure 18: Moving average (10 samples) of the personal ratings of the animations from system 2, in genetic order

4.2 Test 1

The first test tests the performance and behaviour of the system in a public setting involving multiple users, while the neural network is already pre-trained before the test begins. This could give an idea of the performance of the system after operating in such a setting for a longer period of time. This test also investigates the influence of the neural network on the system's output. This is done by comparing a system as it was designed (with a neural network), with one where the neural network was omitted in the filtering component, and the filter thus only rejects animations with too little motion.

4.2.1 Method

Participants

The participants in this test were not selected, but could approach the installation freely. Therefore, there is no clear data on these participants. However, based on observations of the participants and the location of the setup, it could be said that most participants were between 18 and 25 years old and studied Creative Technology. However, there were also some (between 5 and 10) older people who used the installation.

Materials & setup

A computer was connected to a 42-inch TV screen with a resolution of 1360x768. Next to the TV was a high table with the rating box as described in section 3.5.2. A picture of this setup can be found in Figure 19.

The system works as described in the

Design & Implementation section. The long-term memory was seeded with 47 animations which received a rating of three or higher in DS1. The animations generated by the system were all stored with their rating, the date and time they got rated, and if they got rated by a human or not.

A choice had to be made about what neural network to use for this test. However, at the start of this test, the comparison of different networks (Test 2) was not performed yet. Since the execution of this comparison takes almost 48 hours to complete (this time is mainly spent on training the different networks), and there was a limited time to perform this test, it was decided to do a quick comparison with less networks. This was done using DS1, because it was the only dataset available at the time. Networks were compared on the accuracy of their classification. The chosen network had 28 neurons in the hidden layer, a sigmoid alpha of 2, a learning rate of 0.1, and a momentum of 0, with an accuracy of 86%.

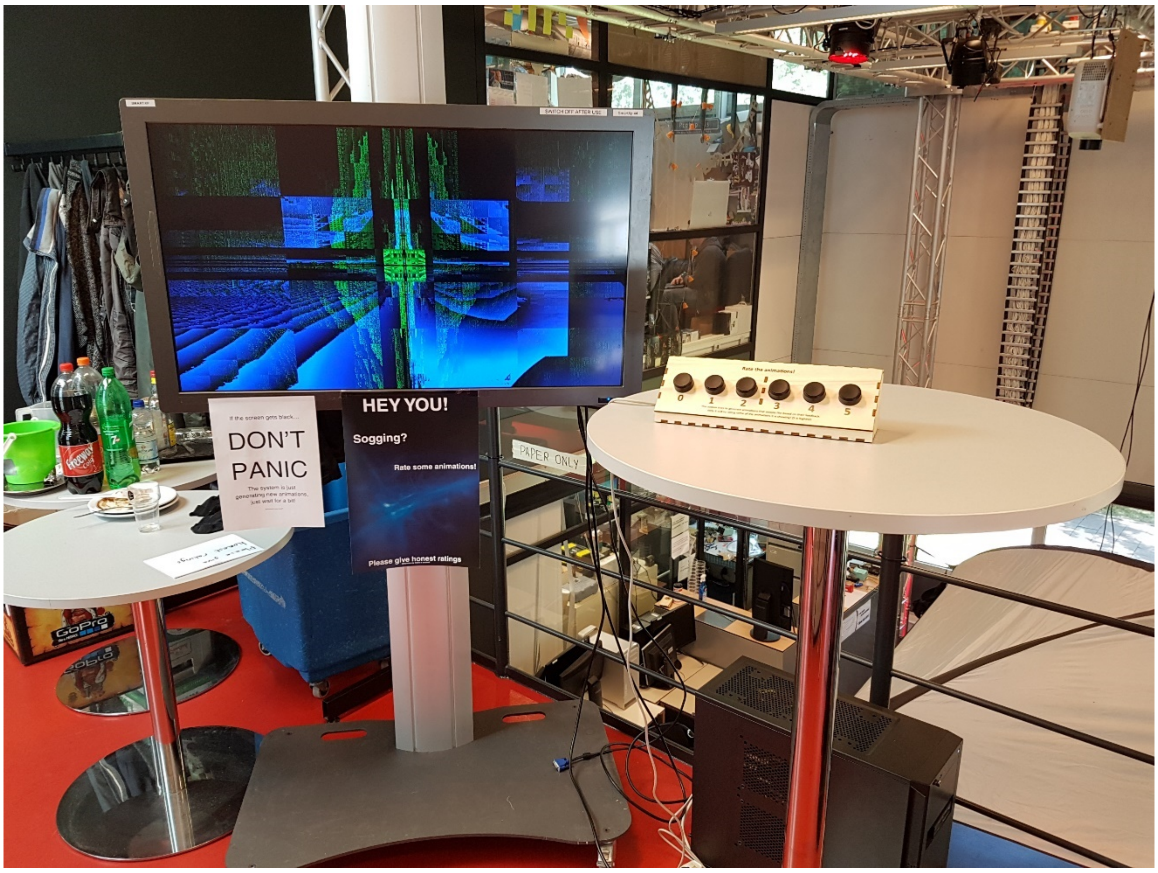

If the screen gets black...
DON'T PANIC
The system is just generating new animations, just wait for a bit!
HEY YOU!
Sogging?
Rate some animations!
Please give honest ratings
Rate the animations!
0
1
2
3
4
5
PAPER ONLY

Figure 19: The setup of the pre-trained test

Design

For two weeks (ten days, only business days), two different systems were tested:

- **System 1**: A system which works as described in the Design & Implementation section.
- **System 2**: A system which did not use a neural network in its filtering component, thus only rejecting animations with too little motion.

Initially, the systems were alternated every day. During the last three days, the system without neural network was tested for two and a half days, and the other system for the last half day. This was done to make sure each system got a similar amount of ratings.

Procedure

The setup was located in a place where people would often walk by and it could be seen well. Participants were passively encouraged to rate these animations using two posters: one underneath the TV and one in a break room that was close by. Participants could rate as many animations as they would like.

Animations were shown full screen, and the next animation was automatically shown after an animation was rated by a user. If the system was generating new animations, a message would be shown on the screen, telling the user to wait for a moment while the system generated new animations. Unfortunately, this did not always work, which resulted in a black screen being shown instead. To prevent participants walking away if they thought the system stopped working when this happened, a message was attached to the TV, telling them that a black screen probably meant the system was generating new animations.

Since the animation rejected by the neural network of system 1 were not shown to the users, these animations did not get a human rating. Therefore, the performance of the neural network in terms of precision, recall or F score cannot be calculated for this system. To get an idea of the performance of the network in this setting, the same neural network system 1 started with, was fed the rated animations from system 2 (DS2). This was done in chronological order, to simulate how the neural network would have behaved on these animations and ratings. Since all animations in this dataset are rated, this simulation could be used to find precision, recall and an F score of the neural network.

The users and system were observed during the test. No formal method was used to record their behaviour, although the observations were focussing on some questions which cannot be directly answered quantitatively:

- How diverse are the generated animations?
- What are the main reasons of users to rate an animation a certain way?
- Do users enjoy interacting with the system?
- Is there a common, shared opinion between users, or does preference differ?

4.2.2 Results

In total 4932 animations were generated; 2482 for system 1 and 2450 for system 2. System 1 filtered 529 animations, where system 2 filtered 169. This means system 1 got 1953 human ratings, and system 2 got 2281 human ratings. These numbers and the distribution of human ratings are summarized in Table 3.

Sadly, early in the test something happened with the rating box, so the wires of the '0' and '1' buttons were not isolated anymore, and touched each other. Therefore, if participants rated an animation as one, it got recorded as a zero. The zero and one ratings are therefore combined in the results. This should still be able to show if the system can filter 'bad' animations from its output, although it is unclear how much zero ratings it rejects compared to one ratings.

Table 3: Results of the pre-trained test

		Neural network (system 1)	**No neural network (system 2)**	**χ^2**	**Significance**
	Animations	2482	2450	-	-
	Filtered animations	529	169	210.884	$p < 0.005$
	Human ratings	1953	2281	-	-
Human ratings	Rating 0 or 1	470	802	61.620	$p < 0.005$
	Rating 1	32	1	34.598	$p < 0.005$
	Rating 2	361	368	4.080	$p < 0.05$
	Rating 3	434	390	17.628	$p < 0.005$
	Rating 4	364	385	2.237	-
	Rating 5	292	335	0.058	-

Human ratings

The relative human rating distributions can be seen in Figure 20. If the neural network would have no influence on the animations shown to the user, system 1 is expected to have the same human rating distribution as system 2. This hypothesis is tested using a Pearson's chi-square test, which resulted in a chi-square value of 96.902 ($p < 0.005$); the hypothesis can be rejected. This means there is a significant difference between the relative distribution of ratings of the two systems. To see how the distributions differ per rating, a chi-square test was performed for each rating. The results can be seen in Table 3. If the difference is significant, the significance is given. The biggest difference can be found in the amount of 0 or 1 ratings between both systems: system 1 got significantly less of these, while it got more ratings of 2 and 3. This means the neural network filtered more 'bad' animations than 'good' ones; if it would have filtered randomly (just rejecting an animation by chance) it would have had the same relative distribution of human ratings as the other system.

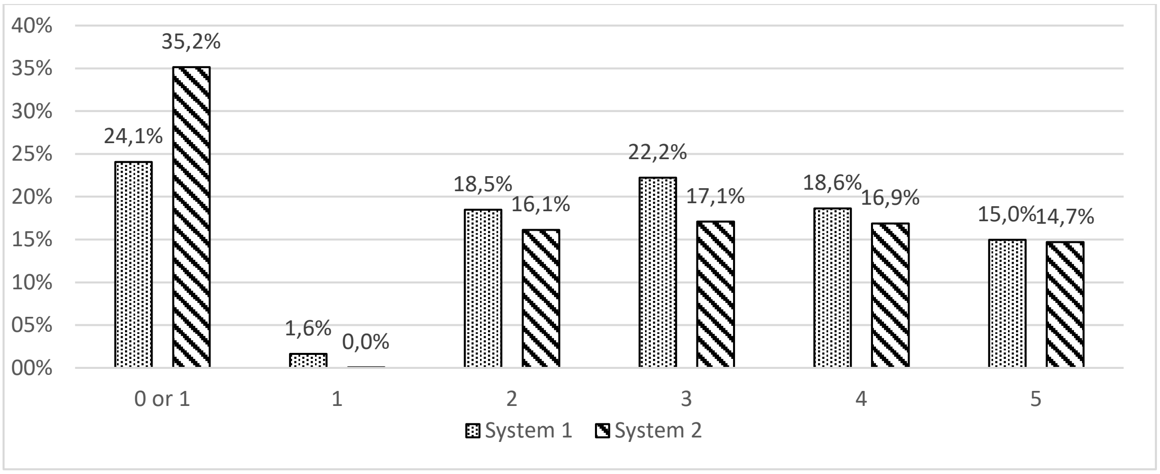


Figure 20: Relative distribution of ratings in the pre-trained test.

Simulated run

The simulated run of the same neural network as used in system 1, on the rated animations of system 2, resulted in the precision, recalls and F scores presented in Table 4. For comparison, these numbers were also calculated for the case where the network would not reject any animation. Note that there is only a very small difference in F_1 3+ score between these two situations; when using the neural network to filter, the precision is higher, but the cost on recall makes that the F_1 3+ score is still similar. The relative rating distributions of system 1 and the simulation are a bit different: the simulation got more 0 or 1 ratings, a bit less 2 and 3 ratings and a bit more 4 and 5 ratings. This can be seen in Figure 21.

Table 4: Precision, recalls and F_1 score of neural network on animations from system 2.

	Filter	**No rejections**
Precision	0.717404	0.648400
Recall	0.844490	1
Recall 3+	0.864865	1
F_1 score 3+	0.784263	0.786702

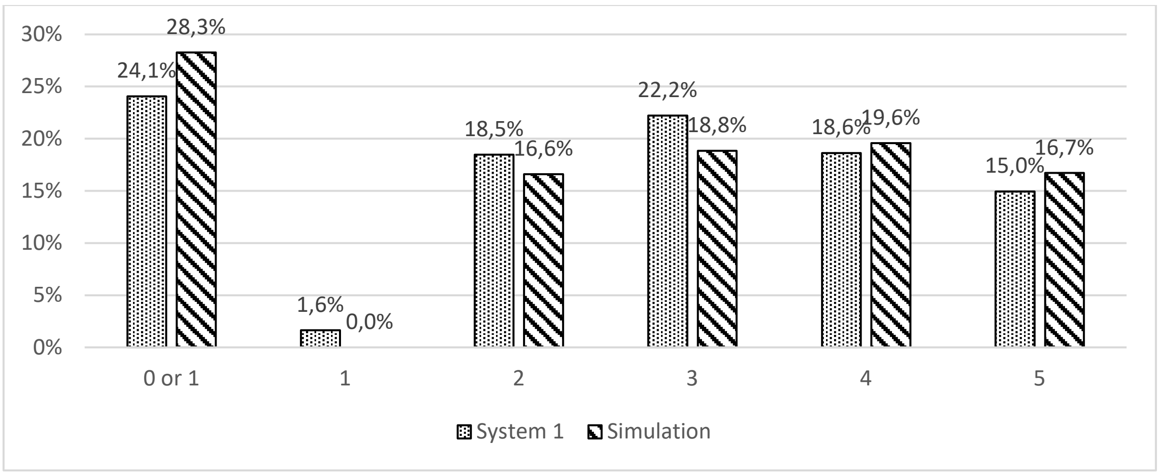


Figure 21: Relative rating distributions of system 1 and the animations after filtering in the simulated run

Observations

Not all users used the system in the same way: some users rated one or two animations while waking by, while others voluntarily used the system for 10-15 minutes. Especially this last group seemed to enjoy it. The system was used both individually and in small groups. This offered a good opportunity to get an insight in users' thinking process while rating the animations, since they would openly discuss their choices when using the system in groups. This showed that their tolerance and reasoning for 'bad' animations could differ much. Some users indicated that they did not like the flickering or lack of motion of a certain animation, but still gave it a high rating because they liked the colour use or a certain visual element in the animation. Others would give zero ratings to animations as soon as there was one 'flash' in the animation or if they did not see any movement within the first two seconds, regardless of other elements of the animation.

Users seemed to love diversity. They often gave higher ratings to animations which contained motion or visual elements that they did not see before. This also seemed to be a motivator for people that used the system for longer times; they liked being surprised by these 'new' animations.

There were still animations shown to users that, according to them, contained too little motion. "It did not move", was heard multiple times as a reason for a low rating. This indicates that the rules for filtering animations with too little motion can be stricter. Furthermore, many animations consisted of a small set of colours: red, green, blue, yellow, cyan, pink, black and white. This makes sense because of the way the animations are rendered, but it shows that diversity can still be improved. Users also noticed that some of the animations were variations of previous ones. Often, they assumed that it was the same animation as they saw before. Some users did not like this and gave low ratings because they "saw it so often already". This effect was probably increased by the reappearance of old animations from the long-term memory (some types of animations that were already generated at the beginning of the test still appeared close to the end).

4.3 Test 2

To determine how neural networks with different settings and layouts perform on filtering the animations the system generates, different neural networks were trained on two datasets: DS1 is used to test networks on single-user data, and DS2 to test on multiple-users data.

4.3.1 Method

Materials

A training set based on DS1 was constructed by including all animations from DS1, and then applying a small amount of noise on the features of the animations to create four more similar versions of each animation. This was done by adding or subtracting a random value of maximum 5% of the original value of the feature. This should make the trained networks more robust to noisy input, since it is expected that animations that have similar features will also have a similar rating. The final single-user training set therefore contains 5150 animations.

To get a similar amount of animations in the training set based on DS2 as in the single-user training set, the amount of animations in the set was doubled using the same method used for the single-user training set. This resulted in a training set with 4560 animations (two animations were removed to make a fair division of the training set possible for the k-fold cross-validation). Note that the dataset itself already contains some similar animations, since it was created using the genetic algorithm.

Procedure

For each of the variable parameters that could be varied in the implemented neural network (as described in section 3.6.3), certain values were chosen to be tested:

- Number of neurons in hidden layer: 14, 21, 28
- Sigmoid alpha: 1 and 2
- Learning rate: 0.1 and 0.01
- Momentum: 0, 0.1 and 0.5

All 36 possible combinations of these values were tested. A k-fold cross-validation was performed with k = 10 on each network setup. It was made sure that the earlier created noisy versions of the animations in the training sets, were not present in the validation sets, or the other way around. For each fold, the network was trained for 500 epochs, and after each epoch the network was validated on the validation set. This validation resulted in a precision score and two different recall scores; the overall recall and the recall of animations which scored ‘three’ or higher. This was done because the overall recall of a certain network could be moderate when it misclassifies a lot of animations with a rating of 1 or 2, while performing much better on the animations scoring higher. Since the recall is especially important for the higher rated animations (the generation of good animations can be quite rare), such a network would be preferred over a network with a good overall recall but a moderate recall on higher rated animations. These precision and recall scores were combined into two different F_1 scores (also one overall and one for animations scoring ‘three’ or higher), so the performance of the networks could be easily compared.

4.3.2 Results

Single user

For overview, only the results of the ten best performing setups are summarized in Table 5, excluding the overall recall and F_1 score. All results of all network setups can be found in Appendix D. If the filter would not filter anything, the precision would be 0.505, resulting in an F_1 3+ score of 0.671.

Table 5: Top 10 of neural networks trained on DS1, based on their F_1 3+ score after training

Network	Learning rate	Momentum	Alpha	Neurons	Precision	Recall 3+	F_1 3+
24	0.01	0.1	1	14	0.746	0.891	0.812
25	0.01	0.1	1	21	0.745	0.891	0.812
18	0.01	0	1	14	0.745	0.891	0.811
17	0.1	0.5	2	28	0.754	0.878	0.811
20	0.01	0	1	28	0.744	0.891	0.811
26	0.01	0.1	1	28	0.743	0.891	0.810
19	0.01	0	1	21	0.739	0.891	0.808
11	0.1	0.1	2	28	0.734	0.891	0.805
1	0.1	0	1	21	0.722	0.891	0.798
8	0.1	0.1	1	28	0.745	0.852	0.795

As can be seen, the F_1 scores are very close, there does not seem to be a network that is significantly better than the others, although network 24 also has a slightly better recall overall (not shown here, see Appendix D). It might be useful to look at the learning curve of each network, as these can still be different. The above described results were gathered without keeping track of the precision and recalls per, therefore a second run was done using the top 10 networks from Table 5. The F_1 3+ scores per epoch of this second run are plotted in Figure 22.

A clear difference is visible between the networks with a learning rate of 0.1 and the ones with a learning rate of 0.01. The networks with a higher learning rate reach their highest F_1 3+ score after around 100 epochs, after which it decreases again, probably because of overfitting. The final precision, recall 3+ and F_1 3+ of the networks are listed in Table 6, as they differ from the results of the first run. Note that the only different condition is the weights the networks are initialized with, as these are random, which thus can be the only cause of the differences with the results of the first run. Since the scores of the fast-learning networks decrease again after around 100 epochs, the maximum seen score during the learning process is given too in Table 6. Note that these maximum scores are very similar for all networks.

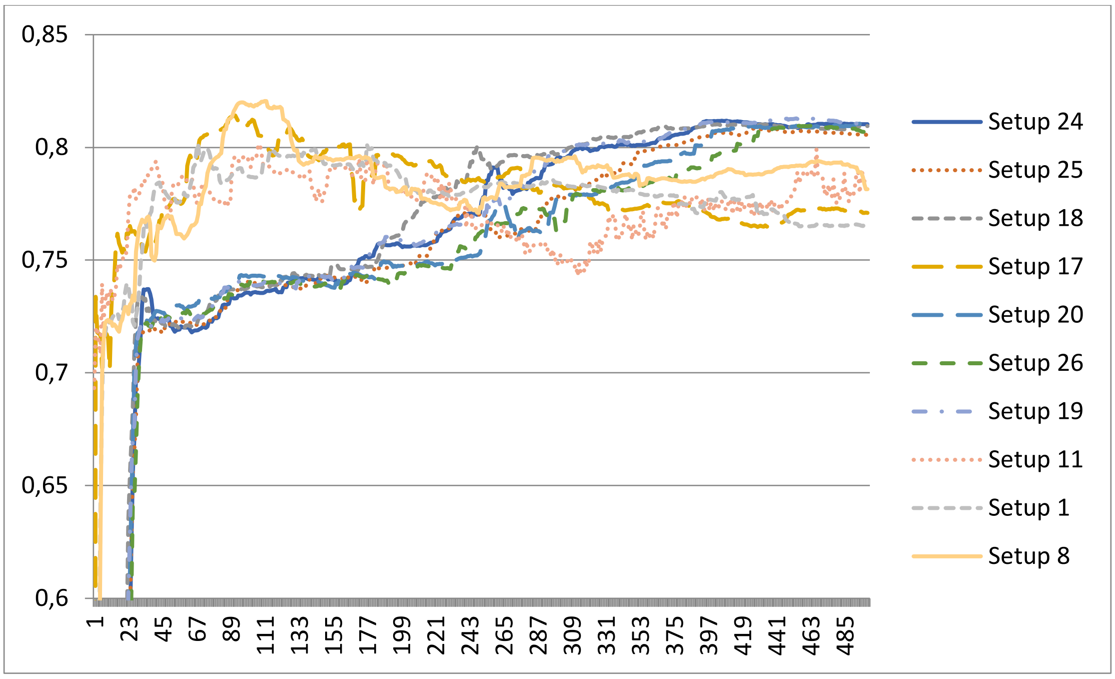


Figure 22: F_1 3+ score per epoch for the ten best performing networks, trained on DS1

Table 6: Final measures on the 2nd run of the top 10 networks, trained on DS1

Setup	Precision	Recall 3+	F1 3+	Max F1 3+ 2nd run
24	0.745452	0.887381	0.810248	0.811836
25	0.737464	0.887381	0.805507	0.808723
18	0.744152	0.887381	0.80948	0.810264
17	0.69876	0.859603	0.770881	0.81428
20	0.745644	0.887381	0.810361	0.810469
26	0.739492	0.887381	0.806714	0.80969
19	0.746756	0.887381	0.811018	0.813355
11	0.716184	0.872937	0.786829	0.801454
1	0.710118	0.830159	0.765461	0.80178
8	0.725178	0.847381	0.781531	0.820577

Multiple users

In the case of multiple users, for all networks the precision and recall after each epoch was stored. The single-user test revealed that taking the final F_1 3+ score might not be the best way to see the potential of each network setup: the networks with a learning rate of 0.1 all scored lower, while their F_1 3+ score earlier in the learning process was often comparable to the final F_1 3+ score of the slower learning networks. Therefore, the top 10 given here is not selected on the final F_1 3+ score, but on the maximum F_1 3+ score reached during the learning process. This top 10 and their scores can be found in Table 7. Their learning curves can be seen in Figure 23. The differences in performance between the networks are minuscule, also when looking at their learning curves. Without a filter, the precision of the system would be 0.648, resulting in an F_1 3+ score of 0.787.

Table 7: Top 10 of network setups on the multi-user training set, based on their maximum F1 3+ score while training

Setup	Learning rate	Momentum	α	Neurons	Precision	Recall 3+	F1 3+	Max F1 3+
34	0.01	0.5	2	21	0.756	0.908	0.825	0.826
21	0.01	0	2	14	0.753	0.890	0.816	0.825
29	0.01	0.1	2	28	0.755	0.898	0.820	0.825
14	0.1	0.5	1	28	0.758	0.895	0.820	0.825
33	0.01	0.5	2	14	0.757	0.903	0.824	0.825
22	0.01	0	2	21	0.754	0.891	0.817	0.825
2	0.1	0	1	28	0.758	0.867	0.809	0.825
12	0.1	0.5	1	14	0.756	0.883	0.815	0.824
8	0.1	0.1	1	28	0.754	0.881	0.812	0.824
35	0.01	0.5	2	28	0.755	0.906	0.824	0.824

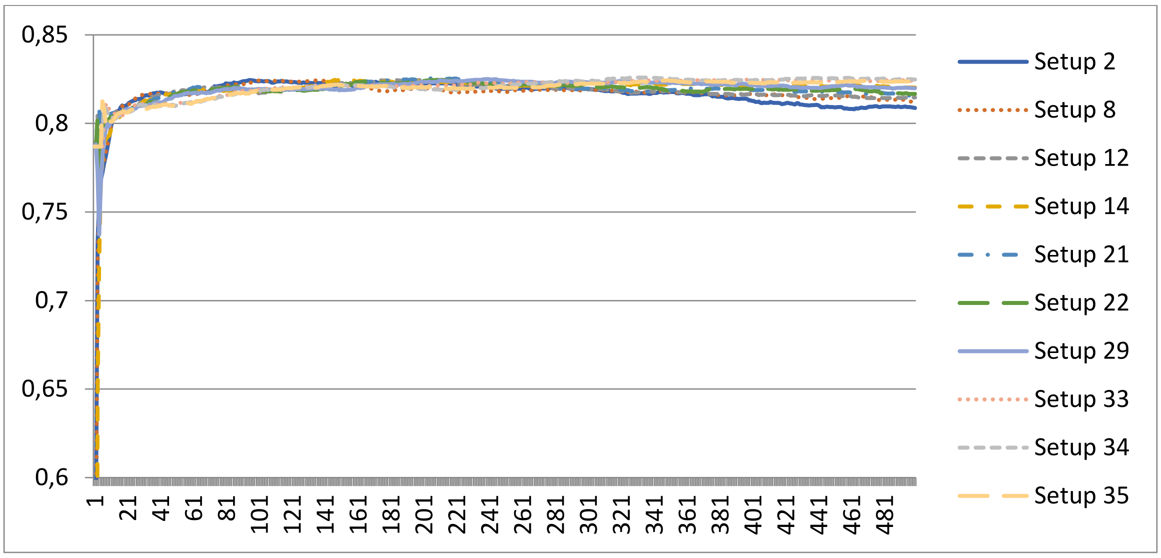


Figure 23: F1 3+ score per epoch for the ten best performing network setups on the multi-user training set

4.4 Test 3

To see how a system which starts with an untrained neural network would perform, two tests were done to see what the performance of the filtering component is when trained on different numbers of animations, and how this performance differs if the system has multiple users or just one user.

4.4.1 Method

Materials & setup

For the tests with multiple users, DS2 is used to build the training sets. For the tests with a single user, DS3 is used for the training sets.

The networks used in this all have the same settings as network 8 as it is presented in the results of Test 2, since that setup seems to learn fast and reaches a high score on the single user training set. On the multiple user training set it also performs well, although here the differences between the setups are much smaller.

Procedure

The network is trained on training sets of 25, 50,100, 200 and 300 animations, which are expanded to five times their size by adding four versions with noise for each animation. This was done using the same method as in Test 2. For each of the training set sizes, ten networks are trained on different parts of the datasets. Each training consists of 100 epochs. This number is derived from the learning curves as seen in Figure 22 and Figure 23; in both curves, network 8 seems to reach its best performance after 100 epochs. The animations in the dataset which are not used in the training set are used to verify the performance of the trained network. The mean performance of the ten trained networks is then taken as an estimation of the performance of the network.

4.4.2 Results

The different performance measures per training set size on both DS2 and DS3 can be found in Table 8 and Table 9. Also, for each training set size, the F_1 score is given of a system without a neural network.

The neural networks trained on DS2 performed better than a system without a filter, when trained on 100 or more animations. The improvement of the system seems to stall when trained on more than 200 animations. However, since 300 was the maximum number of animations in the training set, it is not possible to tell if this stall is local and thus the performance will continue to increase when trained on more animations, or if the performance will stay similar.

The neural networks trained on DS3 all perform similar to a system without a filter, regardless of the size of the training set. However, note that the performance of a system without a filter is already very high for this dataset, due to the lack of zero ratings in the dataset.

Table 8: Performance of networks trained on DS2 using different sizes for the training set

Training set size	Precision	Recall	Recall 3+	F1 3+	No filter F1 3+
25	0.688	0.803	0.816	0.746	0.787
50	0.726	0.786	0.803	0.762	0.787
100	0.726	0.850	0.870	0.792	0.787
200	0.729	0.884	0.905	0.807	0.787
300	0.747	0.843	0.868	0.803	0.787

Table 9: Performance of networks trained on DS3 using different sizes for the training set

Training set size	Precision	Recall	Recall 3+	F1 3+	No filter F1 3+
25	0.876	0.997	0.998	0.933	0.933
50	0.872	1.000	1.000	0.932	0.932
100	0.875	0.991	0.993	0.930	0.932
200	0.878	0.981	0.994	0.932	0.932
300	0.881	0.985	1.000	0.936	0.932

5 Discussion

In this chapter, the work of this project is discussed. The design of the system will be evaluated by looking at the design choices and the limitations of these choices, as well as the results of the tests and the limitations of the tests themselves. The design of the system will be discussed using the requirements stated in section 3.1.

5.1 REQ1

[**The system should generate visibly moving animations.**]

The animation rendering algorithm was originally designed to generate images, but it has been modified to output animations. It can still render animations that are not (visibly) moving, so the filtering component is designed to reject these animations. However, the features to detect these animations are not perfect, as discussed in section 5.6.

The filtering component filters all animations which contain no motion. The results of Test 1 imply that the filter is not strict enough on animations which contain too little motion; users still said they gave an animation a low rating because “it did not move”. However, some users would still give these animations a high rating because they liked certain visual elements in it, indicating that a lack of motion is sometimes not a “deal breaker” for the user. Note that none of these observations were a result of interviews, but were informal observations of the users’ behaviours. Their quotes come from discussions overheard between them. These observations can therefore not be used to draw solid conclusions, but can merely serve as indicators of where problems in the system might be present. This holds for the discussion of all of these observations, also for the other requirements discussed below.

5.2 REQ2

[**The generated animations should be diverse.**]

The used rendering algorithm is not bound to shapes and colours. The generated animations in Test 1 could be used to see how big the diversity of the rendered animations is in practice. However, diversity is hard to measure. In the test the diversity was judged by observing the generated animations personally. This is of course highly subjective and cannot be used to objectively measure the diversity. However, these observations suggest that the diversity could be improved. The algorithm did not always use the freedom it had in the colour domain, it seemed to often stick to more primary colours. Also, the genetic algorithm reduced the diversity; some users were observed giving low ratings because they already saw certain animations often before. This is unavoidable, since a genetic algorithm needs several variants of the same animation to steer towards improvement, but it seems the efforts to minimize this behaviour were not enough. This conflict between automatic improvement and diversity is a problem for the suggested system.

5.3 REQ3

[**The system should gather feedback on the animations from users.**]

The buttons did seem to invite users to use the system, although this was not compared with other interface options. None of the users openly complained about a lack of resolution of the rating scale. However, users were not asked about this after the test, so it is not clear if the rating scale does indeed give users enough room for their opinion. The rating scale's resolution seemed to be big enough for the system to function, shortcomings of the system did not seem to originate from a lack of resolution on the rating scale. However, it was not studied if the 'zero' rating meant the same to all users. There might be a difference in perception, which could result in extra noise on the user feedback if multiple users are involved.

5.4 REQ4

[**The system should improve the generated animations.**]

The ratings in dataset DS3 do not show an improvement of the animations, as can be seen in Figure 18. This lack of improvement could be caused by the diversity of opinions between the users of the system, as the genetic algorithm was only tested with multiple users in Test 1. It could also be caused by the modifications of the genetic algorithm compared with conventional genetic algorithms. Especially the 'long-term memory' could have had a significant impact on this. Animations are never deleted from this memory, so animations from the start of the evolution process can always be 'resurrected' later, putting a 'young' animation in a later stage of the evolution process. Every time this happens, the genetic algorithm is set back a couple of evolution steps. Furthermore, the initial seed of this long-term memory in Test 1 influences the genetic algorithm as well, as the genetic algorithm could render good animations from the beginning of its run. To test improvement properly, it would be better to start without a seed, or with a smaller probability of using animations from memory. Also, DS3 contained only a portion of all animations in DS2, and these animations were rated by one person. This person may have had a very different opinion than most users of the system, so the data is not very reliable.

Since the focus of this project is mainly on reducing the number of unpleasant animations in the system's output, a lack of improvement is not necessarily a problem. As long as the system generates enough high-scoring animations to properly train the neural network, the system can be tested adequately.

5.5 REQ5

[**The system should keep on exploring new animations.**]

The implemented method to realise this works well enough to satisfy this requirement; the algorithm continues with different animations after converging for a while. However, the way convergence is measured is very rough and can probably be greatly improved. The current method can in theory leave a population run and converge on for tens of generations, while the animations do not improve anymore.

5.6 REQ6

[**The system should filter out unpleasant animations.**]

This requirement expresses the main focus of this project. A neural network is used to classify the animations between animations that were predicted to receive a 'zero' rating, and all other ratings.

The features extracted from the animations were designed to be distinctive for the 'bad' animations, focussing on both motion and colour use. These motion features are not perfect. They are extracted by rendering a limited number of frames, so if motion occurs outside or only between these frames, it is 'invisible' in the motion features. Furthermore, the motion features do not look at colour, so a change of hue is also invisible in the motion features. The impact of this is unknown, as the rejected animations were not viewed. However, observations of the animations the system generated before the filtering component was implemented, suggest that the probability of animations that would be misclassified for these reasons is low.

The first results of Test 1 show that the filter, in combination with the rest of the system, did filter the 'bad' animations: when the neural network was in use, there were significantly less 0 or 1 ratings. When the neural network was tested on the data of the other system, it showed indeed an improvement in precision, but at a cost on recall: the network also rejected 10% of the good animations. Since the system can easily generate more animations, the precision is more important than recall in this case: losing 10% of the high-rated animations is not noticed by the user. Recall will only become a problem if it gets so low that it is noticeable in the speed at which the system generates new populations and generations; if only 5% of the good animations would pass the filter, this process would take around 20 times as much time.

The precision in Test 1 is found to be better than if the system would just accept all animations, so the neural network does seem to filter on features that belong to 'bad' animations. However, around 29% of the animations that passed the filter still got a low rating from the users, which in this case means it got either a 'zero' or 'one' rating. Without a filter, around 35% of the shown animations are low-rated. Note that these results do not talk about the performance of the neural network during the test itself, as this could not be measured. This were the results of a simulated run on DS2, so the real performance might be slightly different. Also, it is important to keep in mind that the long-term memory was seeded by animations from DS1, while the neural network was trained on this same dataset. Therefore, the neural network could have scored better on performance than it would have done in a system with other animations in the long-term memory.

In the other tests the precision got to around 0.75, meaning around 25% of the animations passing the filter were considered 'bad' by the users. Only the neural networks trained on DS3 got a higher precision, but the filter did not perform better than not filtering at all; this dataset contains few zero-rated animations by itself.

Test 2 showed that only the learning rate seemed to matter in the performance; varying other parameters of the neural network always resulted in a similar performance in terms of the F_1 3+ score. When trained on the dataset from multiple users (DS2), even the learning rate did not seem to matter much, and all networks seemed to learn faster on DS2 than on DS1. This difference in learning speed is striking. The main differences between the two datasets are the following:

1. DS1 got rated by one person, DS2 by multiple.
2. DS1 was filled with random animations, DS2 got filled with animations using a genetic algorithm.
3. DS2 got most 'one' ratings grouped together with the 'zero' ratings by accident.

The first difference could not explain the difference in learning rate; if anything, the networks would be expected to learn faster on DS1, since the ratings in that dataset are expected to be more consistent. The second difference could have an influence, since this caused DS2 to contain far more animations with high ratings, making it 'easier' for the networks to learn the difference between high-rated and low-rated animations. The third difference could also have some influence, since animations rated with 'one' can contain elements that normally would cause it to be rated 'zero', but other elements make that the user thinks it is still worth more than that; there is a grey area between the two rating options. This could disturb the learning process: if some 'good' animations (anything above a 'zero' rating, for the network) have elements normally found in 'zero' rated animations, the learned link between these elements and low-rated animations gets weaker. In DS2 this grey area between 'zero' and 'one' is gone, since they both got recorded as 'zero', improving the links between bad elements and bad animations. There is of course such a grey area between all neighbouring ratings, so in DS2 the grey area which influences the learning process is between 'one' and 'two' instead. This grey area might be smaller than between 'zero' and 'one', causing an increase in learning speed on this dataset. However, this cannot be proven with these results, it is merely a suggested theory.

Test 3 showed that when trained on animations from DS2, using only 50 animations already brings the precision close to the 0.75 that is seen often during other tests using much bigger training sets. Using 100 or more brings the F_1 3+ score to the same level or higher than the F_1 3+ score the system would have without a neural network. This indicates that an untrained system can probably be tailored quite fast to the tastes of new user(s) by letting those users first rate 50-100 animations, and then training on those animations. However, this is only based on the results of one dataset; other datasets might yield other results. DS3 proved to be a difficult dataset to train on, probably because of the few zero-rated animations in that dataset.

6 Conclusion

During this project, a system was designed based on RQ1: "How to design a system that generates a vast diversity of abstract animations, while minimizing the amount of unpleasant animations in its output?". This question was translated into six requirements for the system:

1. The system should generate visibly moving animations.
2. The generated animations should be diverse.
3. The system should gather feedback on the animations from users.
4. The system should improve the generated animations.
5. The system should keep on exploring new animations.
6. The system should filter out unpleasant animations.

The system can generate animations, although it sometimes still generates animations in which users do not observe the motion. However, users did not always see this as a bad thing, indicating that requirement 1 could be a bit too strict. The diversity between animations is quite big, but the diversity in colour use seems to be lacking, and the impact of the genetic algorithm on the diversity is not to be ignored, as users notice this and may get bored of seeing a lot of variations of the same animation. Observations of users suggest the feedback method fulfils its goal: users seem to like the interaction with the system, the interface gives the system enough information to work with, and the six rating options seem to satisfy the users' needs. The test data indicates that the genetic algorithm that was developed to improve the animations did not work well: there was no clear improvement seen in the animations. However, there are multiple reasons that could be the cause of the lack of improvement, also in the setup of the user test, which makes it impossible to draw clear conclusions from this data. On the other hand, the genetic algorithm was successful in exploring new animations, even with a rough measure for convergence.

The filter of the system was the main focus of this project. The question RQ2: "What is the performance of the filtering component in the designed system in terms of precision, recall and F-score?" summarizes the evaluation of this part of the system. The different tests showed that the filter does indeed filter bad animations from the output, while keeping most good animations. However, the expected precision is around 0.75, meaning that 1 in 4 animations are still considered unpleasant. The recall scores of animations rated 'three' or higher on the scale from 0-5 are in the range of 0.8-0.9, resulting in F_1 3+ scores (the F_1

score based on the recall of animations scoring 'three' or higher) around 0.8. This F_1 3+ score is not always much higher than it would have been in a system without any filter. However, the precision is the most important number in this case, since that is what the user will actually notice. The precision was always higher than the 'precision' of a system without a filter, which was on average around 0.66: around 1 in 3 animations was considered unpleasant.

Overall, this project showed that it is possible to design and develop a system which generates diverse abstract animations, guided by feedback from users. It can also, to a certain extent, filter unpleasant animations from its own output, increasing the mean quality of the animations it generates. This results in a relative high number of 'good' animations, considering computers have no concept of beauty by themselves, and the vast majority of possible animations are often considered 'bad' by users (as can be seen in the rating distribution of DS1 in Figure 15). The performed tests also showed that the system can be further improved on multiple points.

7 Future work and suggestions

The design process and test results show that there is still a lot of potential for interesting studies towards a system like the one realised here. This will be discussed here per component of the system. Also, suggestions are made for interesting points of improvement or adaptations for the developed system.

7.1 Animation generation algorithm

The implemented algorithm already does a good job at generating a diverse set of animations. However, Rooke showed that even more interesting images can be generated by adding fractal algorithms to the function set, for example. [24] It might be interesting to explore other visually interesting algorithms and adding those to the function set. Also, the currently generated animations often consist of basic colours, since the used functions work on the RGB values of each pixel. To improve the colour diversity, colour maps can be used, like Rooke did.

It might also prove worthy to explore how this algorithm can be used to create interactive animations; the same way the TIME variable is used now, variables like sound level or the x and y coordinates of a finger on a touch screen can be used to make the animation respond to its environment.

An observation during the various tests was that animations with iterative functions systems were often not part of the shown animations. This might be because the function needs a lot of arguments, and some of those quickly move the result of the IFS to a black or white image. This makes the potential motion in the animation invisible, causing it to be rejected. It might be better to make sure the arguments that are sensitive for causing this are kept within certain bounds.

7.2 Animation improvement

The implemented genetic algorithm did not seem to improve the animations over time. There are a lot of reasons why this could happen, since the implemented genetic algorithm differs a lot from a conventional genetic algorithm, and the setup of the user test was not ideal for testing this. Therefore, it is probably best to do a separate study towards a genetic algorithm, or maybe other methods, that can improve the animations generated by this system, while also keeping the animations diverse. Another option is to leave the improvement out completely: this would also increase the diversity of the animations. However, this requires a filter with a high precision, and a system that can analyse animations fast enough so the generation of animations does not take too much time, as a lot will be directly rejected (see the rating distribution of DS1 in Figure 15, which was generated without the use of a genetic algorithm).

The long-term memory never deleted older animations, impacting diversity and the genetic process. It might be better to limit the size of this memory, deleting old animations when new ones come in.

This project only looked at the animations generated by the system, but it would be very interesting to see if its animations can reach a similar quality as animations designed by human artists.

7.3 Filtering

Most neural networks that were tested showed some improvement over a system without a neural network, but the differences are small. Although these differences get bigger over time if a genetic algorithm gets involved, there is still lots of room for improvement. A future study could give more insights in how this could be improved. This project already found some possible room for improvement:

- The currently used features could be further improved or expanded; there are probably other features that influence the preference of users, that are currently not used. A study towards what people dislike about the animations that 'slip through' the filter might shed more light on this.
- The network could filter both 'zero' and 'one' ratings, while an animation rated by a human as 'one' is still used by the genetic algorithm. This is expected to improve the performance of the filter, since it deals with the grey area between the 'zero' and 'one' ratings: it is then probably more likely that a one-rated animation slips through the filter than a zero-rated animation, which is not a big problem. The network can also be made to numerically predict the rating, which should also deal with the grey area. It would be interesting to see how such a filter performs.

- It might also be a good idea, since the user ratings are often relative to what the user has seen before, to use the previous ten or twenty ratings as input to the neural network to get more accurate predictions.
- This project used a relatively simple neural network working with extracted features, but maybe other methods can perform better: a neural network analysing the animations directly (using the pixel value's as input), for example.
- The learning process of the neural network can also be changed. There exist other learn algorithms for neural networks, and there are multiple modifications that can be done to the used back-propagation algorithm; a dynamic learning rate or a method to detect overfitting, for example.

8 Bibliography

[1] K. Sims, "Artificial Evolution for Computer Graphics," in *ACM SIGGRAPH*, Las Vegas, Nevada, 1991.

[2] M. A. Boden and E. A. Edmonds, "What is generative art?," *Digital Creativity,* vol. 20, no. 1-2, pp. 21-46, 2009.

[3] H. Cohen, "The further exploits of AARON, painter.," *Stanford Humanities Review,* vol. 4, no. 2, pp. 141-158, 1995.

[4] H. Cohen, "Color, Simply.," Harold Cohen Online Publication, 2006.

[5] H. Munsinger and W. Kessen, "Uncertainty, structure, and preference.," *Psychological Monographs: General and Applied,* vol. 78, no. 9, pp. 1-24, 1964.

[6] P. C. Vitz, "Preference for different amounts of visual complexity," *Systems Research and Behavioral Science,* vol. 11, no. 2, pp. 105-114, 1966.

[7] H. Day, "Evaluations of subjective complexity, pleasingness and interestingness for a series of random polygons varying in complexity," *Perception & Psychophysics,* vol. 2, no. 7, pp. 281-286, 1967.

[8] J. W. Osborne and F. H. Farley, "The relationship between aesthetic preference and visual complexity in absract art," *Psychonomic Science,* vol. 19, no. 2, pp. 69-70, 1970.

[9] J. F. Wohlwill, "Amount of stimulus exploration and preference as differential functions of stimulus complexity," *Perception & Psychophysics,* vol. 4, no. 5, pp. 307-312, 1968.

[10] C. Martindale, K. Moore and J. Borkum, "Aesthetic Preference: Anomalous Findings for Berlyne's Psychobiological Theory," *The American Journal of Psychology,* vol. 103, no. 1, pp. 53-80, 1990.

[11] M. Jakesch and H. Leder, "Finding meaning in art: Preferred levels of ambiguity in art appreciation," *The Quarterly Journal of Experimental*

Psychology, vol. 62, no. 11, pp. 2105-2112, 2009.

[12] C. Martindale and K. Moore, "Priming, prototypicality, and preference.," *Journal of Experimental Psychology: Human Perception and Performance,* vol. 14, no. 4, pp. 661-670, 1988.

[13] O. Vartanian and V. Goel, "Neuroanatomical correlates of aesthetic preference for paintings," *Neuroreport,* vol. 15, no. 5, pp. 893-897, 2004.

[14] L. Bartram and A. Nakatani, "What Makes Motion Meaningful? Affective Properties of Abstract Motion.," in *Pacific-Rim Symposium on Image and Video Technology (PSIVT),* Singapore, 2010.

[15] D. Cosley, S. K. Lam, I. Albert, J. A. Konstan and J. Riedl, "Is seeing believing?: how recommender system interfaces affect users' opinions," in *Proceedings of the SIGCHI conference on Human factors in computing systems,* Ft. Lauderdale, 2003.

[16] H. H. Friedman and A. Taiwo, "Rating the Rating Scales.," *Journal of Marketing Management,* vol. 9, no. 3, pp. 114-123, 1999.

[17] R. Garland, "The Mid-Point on a Rating Scale: Is it Desirable?," *Marketing Bulletin,* vol. 2, no. 1, pp. 66-70, 1991.

[18] H. H. Friedman and L. W. Friedman, "On the Danger of Using too few Points in a Rating Scale: A Test of Validity.," *Journal of Data Collection,* vol. 26, no. 2, pp. 60-63, 1986.

[19] E. I. Sparling and S. Sen, "Rating: how difficult is it?," in *Proceedings of the fifth ACM conference on Recommender systems,* Chicago, Illinois, 2011.

[20] S. Draves, "The Electric Sheep Screen-Saver: A Case Study in Aesthetic Evolution," *Lecture notes in computer science,* vol. 3449, no. Evo workshops, pp. 458-467, 2005.

[21] S. Draves and E. Reckase, *The fractal flame algorithm,* 2004.

[22] O. S. Lawlor, "GPU-Accelerated Rendering of Unbounded Nonlinear Iterated Function System Fixed Points," *ISRN Computer Graphics,* vol.

2012, 2012.

[23] K. Sims, "Interactive evolution of equations for procedural models," *The Visual Computer,* vol. 9, no. 8, pp. 466-476, 1993.

[24] S. Rooke, "Eons of genetically evolved algorithmic images," in *Creative Evolutionary Systems*, San Fransisco, Morgan Kaufmann Publishers Inc., 2001, pp. 339-365.

[25] J. R. Koza, Genetic programming: on the programming of computers by means of natural selection., Cambridge, MA: MIT Press, 1992.

[26] J. H. Holland, Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence, Ann Arbor: University of Michigan Press, 1975.

[27] M. W. Schwarz, W. B. Cowan and J. C. Beatty, "An experimental comparison of RGB, YIQ, LAB, HSV, and opponent color models," *ACM Transactions on Graphics (TOG),* vol. 6, no. 2, pp. 123-158, 1987.

[28] K. Hornik, M. Stinchcombe and H. White, "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks,* vol. 2, no. 5, pp. 359-366, 1989.

Appendix A

Finding machine intuition criteria for generating aesthetically pleasing animated art

Wouter Deenik
University of Twente
Faculty of EEMCS
Human Media Interaction
Supervisor: Dirk Heylen
w.deenik@student.utwente.nl

ABSTRACT

This study tries to find the aesthetic preference of people about animated art. These preferences can be found in terms of complexity and meaningfulness of the art, colour, and speed or predictability of the animation. This study uses an online survey to find out preferences for animated art in terms of speed, predictability and blurriness by showing eight different animations and asking for a rating on preference. Musical taste is also considered as a predictor of visual preference. Slow moving, predictable and non-blurred animated art are found to be preferred. Musical taste does not have any correlation with visual preference. The time people watched the animations is positively correlated with preference rating, especially for the first animation people saw; when they were not biased by already having seen similar animations. These results can be used in an algorithm generating animated art to discard animations with a low chance of being aesthetically pleasing to humans.

CCS Concepts

•**Applied computing** → **Media arts;** •**Human-centered computing** → *Empirical studies in HCI;*

Keywords

Generative art; Abstract animation; Perception of motion; Computational Intelligence; Machine Intelligence

1. INTRODUCTION

Generative art, the type of art that is a result of some computer program being left to run by itself, with minimal or zero interference from a human being [3], is often guided by the creator of the algorithm by using rules and constraints that keep the different parameters of the generative algorithm within certain bounds. This makes sense, since without those rules the generated art tends to be completely random, which is not very interesting or pleasing to look at. The creator of the algorithm, also called the artist[1], uses these rules and constraints to influence the generated artwork(s) and thus makes sure the artwork will look a certain way. The goal is often to make the artwork interesting or pleasing to look at and engage the viewer. Basically all generative art up to now has this human influence from the artist that made the algorithm. AARON, an algorithm made by Harold Cohen, got very close to eliminating the role of a human artist in the process of generating art, at least it looks like it in the output. Every painting it generates can look much different than the previous one and no human is involved in the generation process itself. However, it is still rule-based; Harold Cohen programmed all the rules and constraints of the parameters that generate the art. It is just that Cohen worked for so long on the algorithm that this rule-based system got very complex, giving the resulting art a very high diversity. Furthermore, Cohen himself decided in the end which artworks to print and exhibit. [5] [4].

If an algorithm would create art independently — without the need of an artist creating rules and constraints or making a selection of the generated art — it would need to be able to do the same as the artist does when creating these rules and selections. You could say that it would be more like an artificial artist than a generative artwork that way. Since a computer can quickly generate a lot of different sets of parameters for its generative algorithm, restricting the output is a simple necessity for most algorithms. Automatically selecting the parameter sets that create 'good' art is a difficult design problem though and algorithms like Cohen's AARON do not seem to select at all; the first 'idea' results in an output. In the case of AARON, Cohen makes this selection *after* the generation took place. A human artist like Cohen has experience with past artworks he made, a personal preference, a certain intuition on what works or not and what people would like, and more subtle criteria he uses to select an idea. A computer has none of these things by default, so it would need other (maybe even similar) ways of selecting the 'right' parameters sets for the generative algorithm. The goal of this study is to find basic criteria to form an 'intuition' for an algorithm on which parameter sets will result in aesthetically pleasing art. These criteria will be mainly based on what people would (dis)like to see.

The criteria may depend on what kind of art the algorithm will generate. This study will use abstract animated 2D visual art for three different reasons:

1. *Abstract and 2D* art since it is simple to generate using an algorithm, and it is used most often in existing animated art installations, so will be most relevant.

2. *Animated* art since the time element introduces some parameters like speed, acceleration and predictability

[1]For the moment ignoring the interesting discussion about who the actual artist is in case of generative art. The creator of the algorithm will be referred to as the artist in this paper. More about this discussion can be read in an article by Boden [2].

that are easy to manipulate and have a big influence on the resulting artwork.

3. *Visual* art because although generating music would also qualify using the previous two arguments, the relevance of visual art is much higher towards art installations using LEDs and screens, which are rapidly increasing in popularity.

Unique aspects of animated visual art are factors like speed, predictability, colour and complexity. This study will focus on finding out what people prefer to see in terms of these factors. For example: do they prefer slow or fast movements?

2. RELATED WORK

Numerous studies already tried to answer the question of how people perceive art and what influences their preference. These can be grouped by where the factors influencing preference are found; in the (visual) art itself or the personalities of the observers. There are also numerous studies focusing on music.

2.1 Art

Several studies found that the complexity of generated images correlated with subjects' preferences of the images [18] [23] [6]. These studies used random polygons that differed in the amount of sides (more sides were interpreted as a higher complexity of the polygon), or visuals made by a random walk (where the length of the walk was interpreted as the complexity of the visual). Participants were asked which visuals they preferred. The results suggested that people have a preferred amount of complexity; preference increased with complexity up to a certain point, after which preference decreased again. Later studies used non-representational art instead of generated images. These studies used a small panel of participants to judge the complexity of each artwork. Results showed that preference increased with complexity, but did not find a very clear decrease of preference at the highest complexity ratings. It was suggested that this might be because the artworks with the highest complexity were simply not complex enough, which made it not possible to notice this effect [19] [24]. The latter study also found that the complexity of artworks positively correlates with the amount of times subjects would look at the artworks. This was to be expected since there are more visual stimuli in the artwork, so it would take more time to process everything visually.

Although studies using generated random polygons seemed to find a clear relation between complexity and preference, a study by Martindale found a different relation (a monotonic function instead of an inverted-U function) when the test circumstances were slightly changed [17]. They argued that the shape of the polygons did probably not just measure complexity but also other variables. Further experiments indicated that meaningfulness had a bigger influence on preference than complexity. Meaningfulness is interpreted here as how strong of a mental connection a viewer of the art can make to concepts he knows. For example: a cloud that looks like a cat is probably more meaningful to a viewer than a cloud with a much more random shape.

However, there is also ambiguity in art, which can be described as how many things a viewer might see in art. This makes that meaningfulness and ambiguity are quite related in terms of art. Most artworks with a high ambiguity will have a low meaningfulness and vice versa. Speaking in cloud's terms again: the cat-shaped cloud has a higher meaningfulness but at the same time is less ambiguous to a viewer since the similarity to a cat predominates other interpretations of the shape. The random-shaped cloud probably has a higher ambiguity since it is easier to see multiple similarities to known shapes in it, but these similarities are less strong and thus the cloud has a lower meaningfulness. Similar to complexity, research was done on a relation between ambiguity and preference in art. Jakesch and Leder performed a study that showed that subject's preference and interest towards artworks was highest for artworks with a medium amount of ambiguity [14]. Both artworks with higher and lower ambiguity were rated significantly lower on both interestingness and preference. This shows that there is probably a preferred level of ambiguity in art.

Several studies found a link between preference and colour. Ambiguity and liking ratings were found to be higher for coloured artworks compared to grayscale artworks [14]. Furthermore, the more prototypical colours were, the higher the preference was found for those colours [16] [17]. This might indicate that artworks using prototypical colours are preferred over artworks using less prototypical colours.

The type of art also seems to matter. Vartanian and Goel found that representational art was preferred over abstract art [22]. Another interesting find in their study was that normal artworks were preferred over filtered (blurred) artworks. This might be linked to earlier found correlations between preference, complexity and meaningfulness; a blurred artwork will lose complexity and meaning.

A study by Bartram and Nakatani on how motion is perceived showed that motion features like fast speed, an angular movement shape (vs. curvy), obtuse angles and NOT smooth motion were associated with negative terms such as angry, painful, threatening, disgust, rejecting, urgent, fear, and annoying [1]. Features like slow speed and a curvy motion shape were associated with calm terms such as reassuring, calming, unimportant, relax, boring, and relieved. This indicates that fast motion, motion that changes direction abruptly and jerky motion might not be preferred by people.

2.2 Personality

A lot of research has been done towards how taste in art differs between people and what causes these taste differences. This section will discuss these studies. There are different measures of personality that were found to influence people's preferences in art. Both the Sensation Seeking Scale and the 'Big Five' are used often.

2.2.1 Sensation Seeking

Sensation seeking is measured using the Sensation Seeking Scale (SSS) and generally includes four subscales: 'Thrill & Adventure Seeking' (TAS), 'Disinhibition' (Dis), 'Experience Seeking' (ES) and 'Boredom Susceptibility' (BS). Sensation seekers were found to prefer abstract art [11] [12], surreal paintings [10], pop-art [12], and tension evoking paintings, while being more tolerant of ambiguity in style than low sensation seekers [25]. The ES subscale was found to be the most significant subscale for some of these findings [11] [10]. A study using the same randomly generated polygons as Munsinger and Kessen used in their earlier mentioned research [18] found that sensation seeking correlates with a preference for higher complexity in these figures [20].

2.2.2 *The Big Five*

The 'Big Five' dimensions of personality originate in the NEO Five Factor Inventory: Neuroticism, Extraversion, Openness to Experience, Agreeableness and Conscientiousness. This measure was used less than the Sensation Seeking Scale to find relations between personality and art preference. Results differ between studies [10][12][9], but the one personality dimension with clearest and most consistent results seems to be 'Openness to experience'. People with a high openness tended to prefer both abstract, pop and representational art [12] more, which was very similar to the relation between sensation seeking and art preference in the same study [12]. Feist and Brady even concluded that Openness and the Experience Seeking subscale of the SSS appeared to be measuring much the same thing, and combined the two measures [9]. They found that this combination of measures correlated with a preference in art in general. The difference in preference between people scoring high and low on this measure got bigger towards abstract art (from realistic to ambiguous to abstract).

2.3 Music

Previous mentioned studies in this section looked mainly at static visual art. Animated art would introduce an extra dimension — time —, which can also be found in music. Characteristics such as speed and predictability have a great influence on the type of music and a big subject in existing research is to find out more about the link between personality, mood and preference in certain types of music.

Sensation seekers appear to like 'hard' music such as hard rock and techno and dislike 'soft-popular' music such as soft rock and easy listening [20]. Another study had the more general conclusion that sensation seekers tend to prefer more complex music [15]. It is notable that also for visual art a preference for more complexity was found in sensation seekers.

Just like with visual art the subscale 'Openness to Experience' of the Big Five seems to have the highest correlations with musical taste. Dunn, Ruiter and Bouwhuis did a quick review of four different studies on the relations between Openness to Experience and preference in music [8]. This review showed that the common correlations found in these studies were between Openness to Experience and Reflective & Complex music such as blues, classic, folk and jazz, and Intense & Rebellious music such as alternative, heavy metal and rock [21] [7] [13] [26]. Dunn, Ruiter and Bouwhuis only found a correlation between openness and a preference for jazz. However, they argued that the ambiguous nature of genre labels makes it very difficult to find robust correlations between musical genres and i.e. personality and demographics.

3. METHOD

An online survey was designed to find what elements that are unique in animated art influence how much people like the art.

3.1 Design

The unique aspects of movement are all time-based aspects. The survey focused on two of these: speed and predictability. Additionally, the survey also looked at the influence of the introduction of a blur filter to make the animations 'softer'. Since related studies (see section Related Work) found a clear preference for a moderate level of complexity, and the motion of the animations in a sense already is a form of complexity, it is to be expected that for other factors in the animation a lower amount of complexity will be preferred. Therefore there are three hypotheses related to speed, predictability and blur:

H1 Low speed will be preferred over high speed.
H2 High predictability will be preferred over low predictability.
H3 Blur will be preferred over no blur.

Since there are always taste differences between people, it could be possible that no overall preference could be found while in certain subgroups of people there would be an actual preference present. Previous research found a link between personality and visual preference as well as personality and musical preference, so musical preference might be an indirect predictor of visual preference. Also, since music is time-based as well, some features of music might directly relate to similar features in animations. These were both reasons to include musical preference in this study as well. There were two more hypotheses related to this:

H4 People who prefer faster music also prefer faster motion.
H5 People who prefer unpredictable music also prefer unpredictable motion.

3.2 Materials & procedure

3.2.1 *Survey*

An online survey was constructed, distributed and promoted online. Participants were asked to complete the survey on a reliable internet connection to make sure the animations loaded as fast as possible. They were told they did not have to watch each animation till its end, but instead watch it as long as they wanted before continuing with the survey. These instructions were given on the first page of the survey. After this initial page, eight animations were shown in a random order, one at a time. After each animation the participants were asked to indicate on a scale from one to five how much they liked the animation, where one was the lowest score. Every animation and question was shown on a separate page. Additionally, the time each participant spent on each page was recorded.

After having viewed and judged the animations, participants were asked to write down three of their most favourite songs that represented the range of music they were interested in. For example, if they would like rock, pop and classical music, they were asked to write down one of their favourite songs for each genre. This was seen as a better method than asking for their favourite music genres, since asking for genres seemed to lack accuracy and would have been greatly dependent on individual perceptions of those genres. For instance, within the genre of rock there is fast rock and slow rock, hard rock and soft rock, and within those subgenres there are still thousands of artists and bands with a great variety of musical style. How participants would answer the question would greatly depend on how they personally classify their musical taste. Furthermore, characteristics

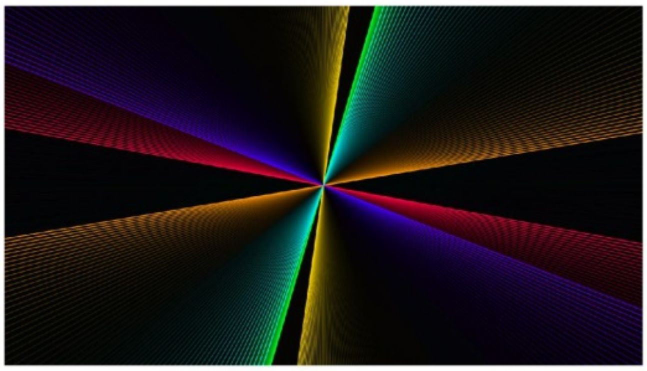


Figure 1: A screenshot of the animation with low speed, high predictability and no filter.

of music that directly relate to characteristics of animated art such as speed and predictability, could be compared directly by analyzing the songs. This would have been much more difficult if participants only chose a genre they liked.

At the end of the survey, participants were asked for their gender, age and field of study and thanked for their participation.

3.2.2 *Animations*

The animations had to be abstract, 2D and very simple, to make sure they would not evoke unwanted associations in the participants' minds that could influence their judgments. The concept of six lines rotating around the middle of the screen was chosen as a good option for this. Each line had a different colour, where the hue was chosen randomly for each line, but consistent through all variations of the animation. The saturation and brightness were at their maximum for all lines. The background was black to increase contrast. Since just having moving lines would leave a large portion of the screen black, each line was designed to leave a trail that would slowly fade to black. A sample of this can be seen in Figure 1.

The behavior of each line was determined by a formula for its angular velocity. This formula consisted of a sum of two sines and two cosines, each having a weight of 0.25. Each sine/cosine looked like this: $\sin(p1 * c * t)$. The variable *p1* was a parameter that was random and unique for each sine/cosine and this made that each line would have different and somewhat random behavior. The variable c had the same value for each sine/cosine and for each line. Increasing this would result in *less* predictable (so *more* **c**haotic) behavior since the angular velocity would change faster and switch direction more often. The speed of the lines was modified by multiplying the resulting angular velocity by a global speed value. To have a manner of changing the complexity and 'hardness' of the animation, a filter was implemented to be able to blur the resulting animation. Since the background was black, blurring the animation also decreased the brightness of the lines. The brightness levels of the blurred animations were adjusted such that the brightness of the brightest point matched the brightness of that same point in the non-blurred version of the animations. The effect of the blur filter can be seen in Figure 2.

The resulting animation could be adjusted on three points: speed, predictability and blurriness. For both speed and chaos a low and high value was chosen (for speed two and

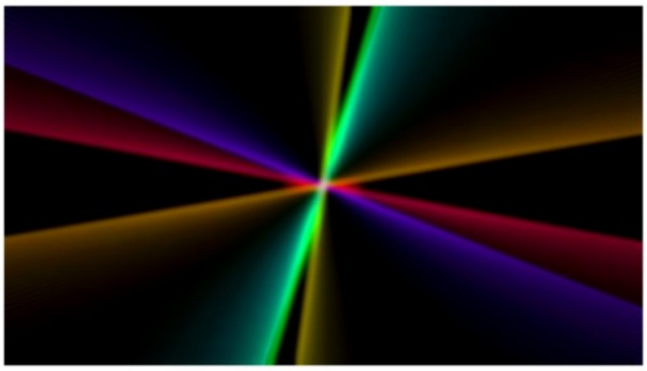


Figure 2: The same screenshot as seen in Figure 1, but with the blur filter applied.

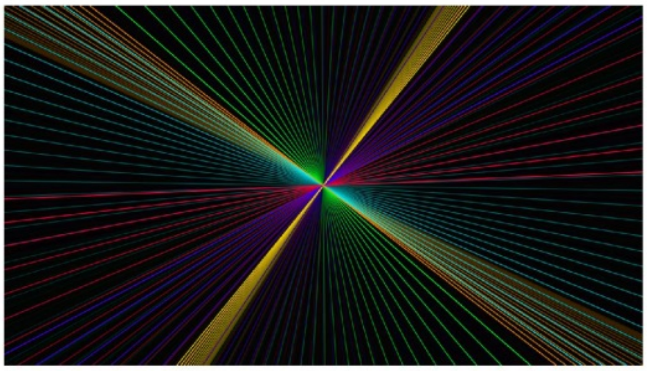


Figure 3: A higher speed results in frames that are more 'filled' and complex.

ten, for chaos one and ten). All combinations of low and high values and having a filter or not were rendered, resulting in eight variations of the animation with a length of 60 seconds. The links to these animations can be found in Appendix **??**. The title of the videos also codes the values used for the speed and chaos variables; the first number is the speed value, the second number the chaos value.

Changing the variables changed more than just what the variables were designed for. For instance, when the speed was increased, the trails of each line would also be more spread out, resulting in each frame being more filled; the coloured lines were more distributed across the screen. This could be seen as increasing complexity too. What such a frame looked like can be seen in Figure 3. Decreasing the predictability resulted in lines changing direction more often, reducing the total distance each line traveled too. Combining this with a low speed resulted in the lines 'wiggling' on their place instead of rotating. Adding high speed resulted in a highly chaotic and complex animation. Having a blur filter decreased this complexity and chaos somewhat since separate trails that were close together were fused, creating a gradient from one line to the next instead of a densely packed area of separate lines. This effect can be seen in Figure 4.

The animations were uploaded to YouTube to be able to embed them in the online survey, which introduced compression on the video files. The compression on the videos made individual lines somewhat harder to see, also on the highest possible quality settings. To reduce this loss of sharpness

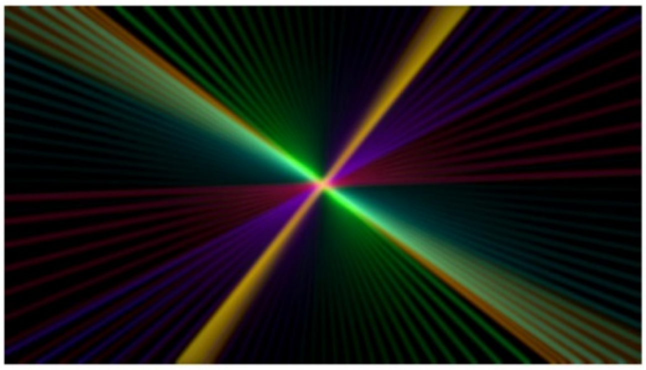


Figure 4: Blurred version of Figure 3. Blurring fuses trails that are close together, reducing complexity.

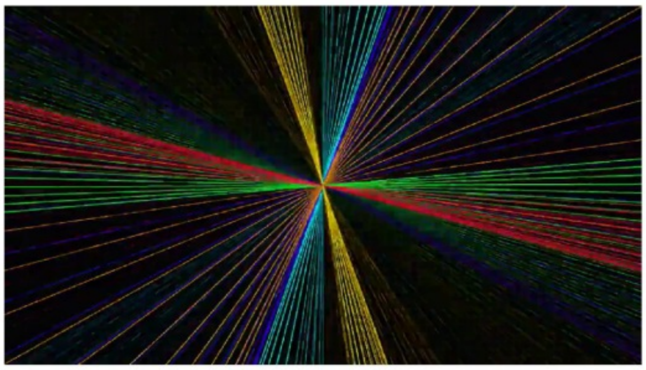


Figure 5: Quality loss due to YouTube compression.

and quality as much as possible, the animations were set to always play on the highest quality setting (720p). The quality loss can be seen in Figure 5.

3.3 Music Analysis

The three songs submitted by each participant needed to be converted into measures that could actually be used for analysis. Two measures were extracted from each song: tempo in beats per minute, and predictability on a scale from one to ten. This was done by the student researcher himself.

3.3.1 Tempo

Tempo was determined by manually 'ticking' the beats on an online BPM measure tool.[2] If there was any doubt if the actual BPM was half or double the measured BPM (for very slow or fast songs), an attempt was made to find sheet music of the particular song, which often reports the tempo at the top of the first page. If that attempt failed, the song was linked to similar music to get an idea in what range the tempo needed to be. For songs that changed tempo during the song the average tempo of the song was used.

3.3.2 Predictability

The predictability was also determined manually and was mainly based on the repetitiveness inside the song. Since a lot of the songs were well-known, any bias because the researcher already knew the song (making it easier to predict) were to be minimized as much as possible. This was done by basing the predictability on the repetitiveness of three different parts of the song: the rhythm, the melody and the lyrics, and initially looking at the spectrogram of the song instead of just listening to it. Examples of spectrograms can be seen in Figure 6.

The rhythm was often dictated by the drums. The more different drum patterns were used and/or the more tempo changes were heard (or seen in the spectrogram), the lower the predictability grade of the song. If there were no drums present, the instrument that mainly dictated the tempo of the song was used as an alternative. The melody was graded in a similar manual way. The main melody was analyzed on repetitive parts and the more unique parts were found in the melody, the lower the predictability of the melody. This was also rated on a scale from one to ten.

The lyrics were analyzed using a script that counted unique words and sentences. The amount of unique words was divided by the total amount of words and multiplied by ten. The same was done for each sentence; the amount of unique sentences was divided by the total amount of sentences and multiplied by ten. Ten minus the average of these two numbers was taken as a measure of predictability for the lyrics.

The overall predictability measure was the mean of the three separate grades for rhythm, melody and lyrics. This overall predictability measure was used in the analysis of the results.

3.4 Participants

Of the 85 participants that participated in the survey, 56 completed the survey completely. Of these participants, 33 were male. Age ranged from 17 to 60, with the majority (71%) being between 20 and 26 years old. Most of the participants in this majority were students at the University of Twente who had a technical background.

Eleven participants who did not finish the full survey did still finish the rating of the animations. These cases were added to the data set for the results that depended exclusively on this data. Participants that were probably distracted during the questionnaire were excluded. This was determined based on the time they spent on an animation page: if this exceeded 65 seconds for at least one animation, the full case was excluded. Participants who did not read instructions and watched every animation till its end were also excluded. This could be detected since the time they spent on an animation page was more than 60 seconds for every animation. Thirteen cases in total were excluded, resulting in a data set for the rating-dependent results of 54 cases.

For the results that depended on the music data, cases where it was not clear which song the participant meant were excluded. This could be because they only noted the song's title, they wrote down an album title instead of a song title, or they only reported genres. After these exclusions the data set for the music-dependent results consisted of 41 cases.

4. RESULTS

For each condition for speed, predictability and blur, there were four animations with that condition. For each possible condition, the mean rating of those animations was calculated for each participant, resulting in mean ratings for slow, fast, predictable, unpredictable, blurred and non-blurred animations. Paired t-tests were conducted on the condition

[2]http://www.all8.com/tools/bpm.htm

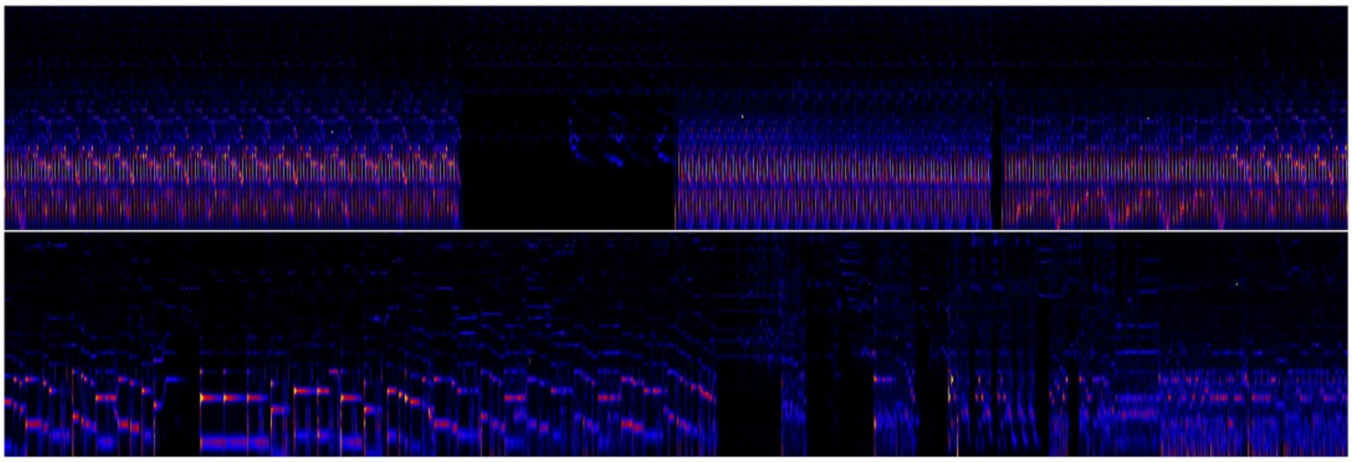


Figure 6: Spectrograms of two songs. The top got a high predictability rating (Daft Punk - Around the World), the bottom got a low predictability rating (Queen - Bohemian Rhapsody)

pairs [slow – fast], [predictable – unpredictable] and [blur – no blur] with the preference rating as dependent variable. These results can be seen in Table **??**. There was a significant difference between slow and fast, predictable and unpredictable, and non-blurred and blurred animations ($p < .01$). The same procedure was used on the time participants watched each animation. Watch times for slow and predictable animations were significantly longer than for fast and unpredictable animations ($p < .01$). These results can be seen in Table **??**.

Factor	Low	High	T
Speed	2.91	2.38	4.46**
Predictability	2.38	2.91	-7.10**
Blur	2.76	2.52	2.88*

Table 1: Means of preferences and t-factor per factor. *p < .01. **p < .005.

Factor	Low	High	T
Speed	15.41	13.46	2.84*
Predictability	13.58	15.29	-2.74*
Blur	14.91	13.96	1.24

Table 2: Means of watch times and t-factor per factor. *p < .01.

To further explore the relation between blur and preference, multiple paired t-tests were conducted on each animation pair where speed and predictability were constant but the blur differed. There was a significant difference in preference rating for the animations with low speed and high predictability with blur (M = 2.93, SD = 1.01) and without blur (M = 3.63, SD = .81). This was also the case for the animations with low speed and low predictability with blur (M = 2.28, SD = .96) and without blur (M = 2.81, SD = .91). Both results with $p < .005$. The animations with a high speed did not have a significant difference in preference between the blurred and non-blurred versions.

Independent	Preference	Watch time
Watch time	.04*	-
Order	.04	.09*

Table 3: Variation of preference (R^2) and watch time explained by watch time and place in order shown. *p < .005.

A linear regression was conducted with the time the participant watched the animation as independent variable and the preference rating as dependent variable. A significant positive correlation was found. To test whether the order in which animations were watched mattered, a linear regression was conducted with the preference rating and watch time as dependent variables and the position of the animation in the order they were shown to the participant as independent variable. A significant decrease in watch times the later an animation was shown was found, while no significant effect could be seen on the preference ratings. These results can be found in Table **??**. To explore how the unbiased (first) preference rating is related to the watch time, a linear regression was conducted on the preference ratings and watch times of the first animation shown to each participant. Here a positive correlation was found too ($r^2 = .17$, $p < .005$)

The mean music tempo and predictability were calculated over every participant's three songs. Multiple linear regressions with these means as independent variables and the ratings for each group of animations (slow, fast, predictable, unpredictable, blur, no blur) as dependent variables did not show any significant correlations.

5. DISCUSSION

The results of the preference ratings showed a clear preference for slow and predictable animations, agreeing with the first two hypotheses of this study. However, the effect of blur on the preference rating was opposite of what was expected. This may be explained by the conclusion of previous research that people tend to have a preferred complexity level; the non-blurred animations might have been preferred because these had a small increase in complexity compared to the blurred versions. The result of comparing all the blurred and non-blurred versions of the animations show that this might well be true. When the animations had a low complexity (because of a low speed and high predictability), non-blurred versions were preferred. However, when the animations had a high complexity because of the high speed and low predictability, this preference for non-blurred versions disappeared. The difference in preference was bigger for the animations with the lowest complexity (low speed and high predictability) than for the animations with a slightly higher complexity (low speed and low predictability), which makes sense in terms of the theory suggested here. The animation with lower predictability might already be closer to the preferred complexity level regardless of the blur. This assumes that the complexity does indeed increase when speed increases and predictability decreases, which was not measured in this study.

What is also interesting to see in the preference ratings is that participants tended to rate their preference rather low on the scale for all animations. This can be explained using earlier studies which showed that generally representational art is preferred over abstract art [22]. Since all animations were abstract and did not offer much opportunity for participants to give meaning to the abstract shapes (as can be done by for example watching clouds and recognizing shapes and animals in the clouds), it is to be expected that preference ratings can be lower.

Preference in music did not have any significant effect on the preference in animations, so the fourth and fifth hypotheses cannot be confirmed. Analyzing musical preference is a very difficult task because the existing categorical system known as genres is highly ambiguous and as experiences in this study show, extracting more objective data such as tempo and predictability is also hard to do in an unambiguous manner. Tempo can be measured in different ways (i.e. a tempo of 120 beats per minute can in some cases also be considered to be 60 beats per minute with 'shorter' notes), and predictability depends on an unknown amount of different factors. In this study the predictability was mainly determined by looking at repeating patterns, but a certain melody can also be predictable because it follows a 'logical' pattern or is similar to other well-known melodies. The main conclusion in terms of the link between music and animation preference should therefore be that a correlation could not be found between the speed, predictability or blurriness of an animation and tempo or predictability of the music *in the way these were measured in this study.*

An interesting find is the correlation between watch times and preference ratings. Vartanian and Goel found the same correlation for paintings [22], but it was never confirmed for animations. This could mean someone's preference of an animation can partly be predicted based on the amount of time they look at the animation, although the variance in preference explained by the watch time is small. However, since the watch times went down the more animations participants already saw, it might be more reliable to only look at the ratings and watch times of the first animation each participant saw, when they were still unbiased. In this case the variance in preference explained by the watch time was much bigger, although still a minor part of the total variance. It still shows that it might be very useful to include the time someone watches an animation as a predictor when predicting their preference of that animation. This could be used as feedback to a system generating animations without needing explicit interaction with people looking at the animation.

This study tried to find criteria that can be used by a machine creating animated art as an 'intuition' to select and create art that people like to see. It showed that people tend to prefer relatively slow moving and predictable art that still has some level of complexity. Existing studies already showed that indeed a certain level of complexity is preferred, as well as a certain level of meaningfullness. The latter can be quite hard for a machine to generate and this is an interesting topic for future studies: how can a machine generate art that is meaningful for humans? Coloured art is preferred over art in grayscale, and the more prototypical these colours are, the higher the preference. Another interesting finding was that the time someone watches an animation correlates with their rating of the art. These findings can form a great base for 'machine intuition' in terms of what kind of animated art will be appreciated by humans. The next challenge is to construct a system that actually uses such an intuition and possibly feedback to generate animated art.

6. REFERENCES

[1] L. Bartram and A. Nakatani. What makes motion meaningful? affective properties of abstract motion. In *Image and Video Technology (PSIVT), 2010 Fourth Pacific-Rim Symposium on*, pages 468–474. IEEE, 2010.

[2] M. A. Boden. Authenticity and computer art 1 2. *Digital Creativity*, 18(1):3–10, 2007.

[3] M. A. Boden and E. A. Edmonds. What is generative art? *Digital Creativity*, 20(1-2):21–46, 2009.

[4] H. Cohen. The further exploits of aaron, painter. *Stanford Humanities Review*, 4(2):141–158, 1995.

[5] H. Cohen. Towards a diaper-free autonomy, 2007.

[6] H. Day. Evaluations of subjective complexity, pleasingness and interestingness for a series of random polygons varying in complexity. *Perception & Psychophysics*, 2(7):281–286, 1967.

[7] M. J. Delsing, T. F. Ter Bogt, R. C. Engels, and W. H. Meeus. Adolescents' music preferences and personality characteristics. *European Journal of Personality*, 22(2):109–130, 2008.

[8] P. G. Dunn, B. de Ruyter, and D. G. Bouwhuis. Toward a better understanding of the relation between music preference, listening behavior, and personality. *Psychology of Music*, page 0305735610388897, 2011.

[9] G. J. Feist and T. R. Brady. Openness to experience, non-conformity, and the preference for abstract art. *Empirical Studies of the Arts*, 22(1):77–89, 2004.

[10] A. Furnham and M. Avison. Personality and preference for surreal paintings. *Personality and Individual Differences*, 23(6):923–935, 1997.

[11] A. Furnham and M. Bunyan. Personality and art preferences. *European Journal of Personality*, 2(1):67–74, 1988.

[12] A. Furnham and J. Walker. Personality and judgements of abstract, pop art, and representational paintings. *European Journal of Personality*, 15(1):57–72, 2001.

[13] D. George, K. Stickle, F. Rachid, and A. Wopnford. The association between types of music enjoyed and cognitive, behavioral, and personality factors of those

who listen. *Psychomusicology: A Journal of Research in Music Cognition*, 19(2):32, 2007.

[14] M. Jakesch and H. Leder. Finding meaning in art: Preferred levels of ambiguity in art appreciation. *The Quarterly Journal of Experimental Psychology*, 62(11):2105–2112, 2009.

[15] P. Litle and M. Zuckerman. Sensation seeking and music preferences. *Personality and individual differences*, 7(4):575–578, 1986.

[16] C. Martindale and K. Moore. Priming, prototypicality, and preference. *Journal of Experimental Psychology: Human Perception and Performance*, 14(4):661, 1988.

[17] C. Martindale, K. Moore, and J. Borkum. Aesthetic preference: Anomalous findings for berlyne's psychobiological theory. *The American Journal of Psychology*, pages 53–80, 1990.

[18] H. Munsinger and W. Kessen. Uncertainty, structure, and preference. *Psychological Monographs: General and Applied*, 78(9):1, 1964.

[19] J. W. Osborne and F. H. Farley. The relationship between aesthetic preference and visual complexity in absract art. *Psychonomic Science*, 19(2):69–70, 1970.

[20] D. Rawlings, F. Twomey, E. Burns, and S. Morris. Personality, creativity, and aesthetic preference: Comparing psychoticism, sensation seeking, schizotypy, and openness to experience. *Empirical Studies of the Arts*, 16(2):153–178, 1998.

[21] P. J. Rentfrow and S. D. Gosling. The do re mi's of everyday life: the structure and personality correlates of music preferences. *Journal of personality and social psychology*, 84(6):1236, 2003.

[22] O. Vartanian and V. Goel. Neuroanatomical correlates of aesthetic preference for paintings. *Neuroreport*, 15(5):893–897, 2004.

[23] P. C. Vitz. Preference for different amounts of visual complexity. *Behavioral science*, 11(2):105–114, 1966.

[24] J. F. Wohlwill. Amount of stimulus exploration and preference as differential functions of stimulus complexity. *Perception & Psychophysics*, 4(5):307–312, 1968.

[25] M. Zuckerman, R. S. Ulrich, and J. McLaughlin. Sensation seeking and reactions to nature paintings. *Personality and individual differences*, 15(5):563–576, 1993.

[26] R. L. Zweigenhaft. A do re mi encore: A closer look at the personality correlates of music preferences. *Journal of individual differences*, 29(1):45–55, 2008.

APPENDIX

A. LINKS TO ANIMATION VARIATIONS

S	P	B	Link
L	H	H	https://youtu.be/U7fbnxTydJw
L	H	L	https://youtu.be/y_nSTmWYN3I
L	L	H	https://youtu.be/nQP8T8pjsv4
L	L	L	https://youtu.be/mjGS2WasECE
H	H	H	https://youtu.be/qBZUwpnYzmk
H	H	L	https://youtu.be/V21Pd9bf-x8
H	L	H	https://youtu.be/GdlwRHvq9-o
H	L	L	https://youtu.be/giyAHJD-kqk

Table 4: Links to each animation. S=Speed, P=Predictability, B=Blur, H=High, L=Low

Appendix B

User feedback for a genetic algorithm generating animated art

Wouter Deenik
University of Twente
Faculty of EEMCS
Human Media Interaction
w.deenik@student.utwente.nl

ABSTRACT

A genetic algorithm that generates animated art needs human feedback to be able to improve the generated art; a computer is unable to measure how good the generated art actually is. This study tries to find the best method of gathering user feedback for such an algorithm. A genetic algorithm is developed that can generate many kinds of animated art based on given parameters and takes human feedback on the beauty of the animations as a fitness measure. A selection of two user feedback methods is tested on a small group of participants. No clear majority of participants preferred one method over the other, but it seems like each method has its strengths in certain situations. An informal test on the effectiveness of this feedback seems to indicate that the generated art indeed improves over time, but also shows some problems with the reliability of the feedback. Some improvements for the algorithm are suggested.

CCS Concepts

•**Applied computing** → **Media arts;** •**Human-centered computing** → *Empirical studies in HCI;*

Keywords

Generative art; Abstract animation; Perception of motion; Computational Intelligence; Machine Intelligence; Genetic Algorithm; Aesthetic Feedback

1. INTRODUCTION

Generative art, the type of art that is a result of some computer program being left to run by itself, with minimal or zero interference from a human being, has been around since not that long after the first programmable computers emerged [2]. Although the name 'generative art' suggests that the art is generated completely by a computer, this is often far from what actually happens. The algorithm is still designed by a human and the inner workings, limits and constraints of this algorithm determine what the resulting art will look like. So yes, the resulting art is generated by a computer and the creator of the algorithm does not have direct influence on the output, but his indirect influence is significant since he sets certain constraints for the algorithm. These constraints are often a necessity since an algorithm with complete freedom in what it displays will have a very low probability of producing pleasing art. This can be compared to the famous infinite monkey theorem, where an infinite amount of monkeys behind typewriters for an infinite amount of time will eventually write the complete works of William Shakespeare by chance. Similarly, an algorithm without any constraints will surely be able to produce art that is considered beautiful by people, but the probability will be so low that it will practically never happen.

This gives rise to a big challenge when designing an algorithm that does not just generate the art, but designs it too; an algorithm where there is no doubt that the algorithm itself is the artist and not the person that created the algorithm. An algorithm like that could change its own structure, limits and constraints to converge towards art that is considered 'good art', while also being able to explore new options in art. This would make it possible to have a piece of digital art that is constantly changing on its own without the need for a human artist designing the art. Constantly changing art is an interesting option for (semi-)permanent technological art pieces in public spaces or buildings, which will often be on display for several years. Having the content change over time will keep these art pieces more interesting through the years. Since these art pieces often use visual and animated art, this study first explores different ways of implementing an algorithm that can generate animated art and can adapt itself to be able to find good art.

Since the amount of parameters needed for the art generation is probably quite big, the 'solution space' for the generated art is vast. Furthermore, this is not a traditional optimization problem since there is not a clear definition of what good art is and thus there is also no one clear solution to find in the solution space. A genetic algorithm can deal with these problems though; it can explore vast solution spaces and does not stick to one local optimum. It can also be made in such a way to keep on exploring different options after finding good solutions. However, the algorithm would need a way to determine the 'fitness' of an animation; in this case related to how visually appealing an animation is. There is no way for a computer to measure this, so user feedback is necessary to know how good each animation is considered to be. However, this introduces new problems such as bias, consistency and ease of use.

This study tries to answer the following question: "What way of gathering user feedback on visual appeal can be used as a reliable fitness measure for animated art?". This question can be answered using the following subquestions:

1. What criteria determine if a feedback method can be used as a reliable fitness measure?
2. Which method is best according to these criteria?
3. Does this feedback method indeed improve generated animated art?

To answer these questions, an algorithm is developed that can generate animated art and can use a fitness measure to improve its art using a simple genetic algorithm.

The study is split into three sections. In the first section, the development of the algorithm is described both in terms of the visuals and the genetic algorithm. The second section is about feedback methods, and the first two sub questions are addressed and answered here. In the last section, the best feedback method is tested on the algorithm to see if the given feedback does actually improve the art. In both the second and third section the algorithm is already used, so these sections will also discuss improvement points for the algorithm itself.

2. ALGORITHM

This section describes the development of an algorithm that can generate and display animated art using a genetic algorithm. It first explains the background of genetic algorithms, then explores existing algorithms and finally describes how the algorithm used for this study works.

2.1 Background of Genetic Algorithms

Although ideas for applying evolution theory to find solutions to complicated problems have been around since the 1950s, most genetic algorithms we know today have their roots in the work of John Holland, who developed genetic algorithms (GAs) to design artificial systems based on natural adaptive systems [10]. Besides GAs there are other rather popular algorithms [1] which include: genetic programming, evolution strategies and evolutionary programming. However, GAs are by far the most popular used today and often the term Genetic Algorithm is used for the other kinds as well.

Although there are differences between these different kinds of evolutionary algorithms, they share the same principle and main components. In short, a GA mimics evolution as it occurs in nature: it maintains a population of individuals, which all get a score based on their performance. The best scoring individuals (which can be compared to organisms which survive longest and/or are most fertile) will reproduce to form a new population, which is tested again. One cycle like this is called a generation. After multiple generations, the individuals in the population improved compared to the individuals the algorithm started with. Because they all mimic evolution in some way, the different kinds of GAs share the same main components: a genetic representation, crossover and mutation methods and a fitness function. These components will be described below.

2.1.1 Genetic representation

In evolution we often talk about genotypes and phenotypes. In nature, DNA is the genotype and the phenotype is the creature carrying that DNA; the genotype encodes the phenotype. The same goes for most GAs: the program or problem that the algorithm is applied to is encoded in a certain genetic representation so the GA can evolve that representation. The phenotype resulting from that genotype can be tested by the fitness function. In some cases the genotype and phenotype are exactly the same: the fitness function can test the fitness of a solution directly on the genotype. In the classic GA as Holland developed it, the genetic representation is a string of ones and zeros. However, a genotype can also be a parameter set of different numbers, or a string of characters. In the case of genetic programming, the genotype is a program, often stored as a tree where each node is a statement.

2.1.2 Crossover

In nature, animals can mate to reproduce; they exchange DNA and this DNA is combined to form a new DNA sequence. In GAs this is called crossover. Depending on the genetic representation used there are many different ways of 'merging' two representations (called *parents* in this case) to form a new one. Holland used a method where a so-called 'crossover point' is chosen. Both parent strings of ones and zeros are cut at that point, and the first part of the first parent is combined with the second part of the second parent and the first part of the second parent is combined with the second part of the first parent. This means performing crossover on two strings produces two new strings (or *children*). This type of crossover can also be performed on parameter sets, but crossover on a parameters set could also be done by choosing for each parameter which parent to use as a source. In the case of genetic programming, whole branches of two parent program trees can be swapped to form children.

2.1.3 Mutation

Apart from crossover, genetic representations can also be altered by mutation. The probability of mutation occurring is often quite low. How this works really depends on the genetic representation. In Holland's representation a random one in the string representation could flip to a zero or the other way around. In parameter sets a parameter can get a random small addition or subtraction to change its value a bit. In genetic programming, a node can be replaced with a new randomly generated branch.

2.1.4 Fitness function

The intelligence in a genetic algorithm originates mainly from the fitness function. This function directs the evolution process towards the better solutions and is therefore essential. The fitness function itself is dependent on the application, but the result of the fitness function is a measure that represents the performance of a solution. An example: the fitness measure for an algorithm that should recognize circles in an image could for example be the percentage of false-positives and false-negatives it finds in a test set; the lower the better. For a GA that finds the optimal way of braking in a car, the fitness measure can be the braking distance; also the lower the better.

These are objective and explicit measures and this is preferable of course; the fitness measure should be as close as possible to the actual performance of a solution. However, this introduces a problem when the goal is to produce 'good art'. There is no objective and explicit measure for good art; taste and experience are personal and subjective. It is not possible to measure it by analyzing the art. The only way of getting an estimation of how good the art is, is to ask people about their opinion. The section 'Feedback Methods' will study what the best way of gathering this feedback is and how to deal with bias and inconsistencies in feedback in a genetic algorithm.

2.2 Existing algorithms

There are several existing algorithms that generate some form of animated art and some already use a genetic algorithm to generate the art. Several of them will be explored and explained here.

2.2.1 Electric Sheep

A famous one is Electric Sheep, a project that uses distributed computing to render animated art that is generated using a genetic algorithm [4]. Each frame in an Electric Sheep animation is a fractal flame, a member of the Iterated Function System (IFS) class of fractal algorithms [5]. This algorithm generates a fractal by iterating different spatial transformation functions and displaying the log density of the amount of times a point is 'hit' by the different transformation functions. Colour is introduced by coupling a colour to each function. This allows a big diversity of visual animations. An example of two fractal flames can be seen in Figure 1. Fractal flames are animated into Electric Sheep by varying some parameters of the fractal flame over time.

Each animation can be rated by users all around the world using a binary scale; each user can vote an animation 'up' or 'down'. If a user votes 'up', the rating for that sheep is incremented, voting 'down' decrements the rating. These ratings decay over time; every day the ratings are divided by four.

An Electric Sheep animation's genetic representation consists most times of 6 transforms which are constructed out of 27 parameters, so in total a genotype consists of 162 floating-point parameters [4].

Crossover can happen between two randomly chosen parents where the probability of being chosen is proportional to user rating. Crossover can happen in two ways, each way happening as often as the other: the new genome is created by alternating the transforms of both parents, or by doing a linear interpolation between the two parents using a blend factor between zero and one.

Mutation can happen by randomizing certain groups of parameters, adding noise to parameters, changing colours or adding symmetry. Which groups of parameters there are and how colours and symmetry work can be found in Scott Draves' work [4] [5]. It is good to mention that mutated sheep are rendered in a low resolution to see if they are not too dark or too bright before they are accepted.

Besides these genetic operators, the algorithm can add new randomly generated sheep and users can add sheep they developed themselves to the population.

A disadvantage of this algorithm is the time it takes to render a fractal flame. Iterated Function Systems take quite some computing time to execute, and an animation would need at least 20 fractal flames per second. Getting this kind of performance is possible nowadays using the GPU [11], but this can be quite a complicated process if the flames should be able to be modified by a genetic algorithm. Although the diversity between fractal flames is big, they still have a distinct visual style.

2.2.2 Milkdrop

Another existing algorithm that generates animated art is used as music visualization in the music player Winamp[1]. It is called Milkdrop[2] and uses audio wave forms, shapes, spatial transformations and pixel shaders to generate its animations. Unlike Electric Sheep, Milkdrop animations are made by people programming so-called 'presets'; it does not have a genetic algorithm. However, it should be possible to convert these presets to a genetic representation that can be modified by a genetic algorithm. There is one problem though: the presets can currently only be rendered by the music visualization plugin itself, which needs Winamp to be playing music. This makes it less ideal to use for this study, but the strategies used to form the animation can serve as inspiration for developing a stand-alone program. For example, wave forms and shapes can be generated from parameter sets using the 'Superformula' created by Johan Gielis [9]. The big advantage of Milkdrop over an algorithm such as Electric Sheep is that the algorithm was always designed to run in real-time and is therefore guaranteed to run fast enough to animate. An example of the output of a Milkdrop preset can be seen in Figure 2.

[1]http://www.winamp.com/

[2]http://www.geisswerks.com/milkdrop/

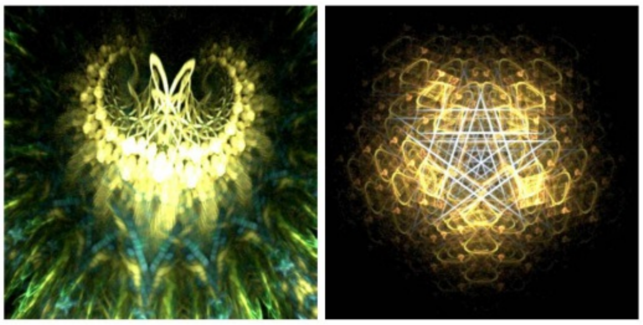


Figure 1: Two different fractal flames [5].

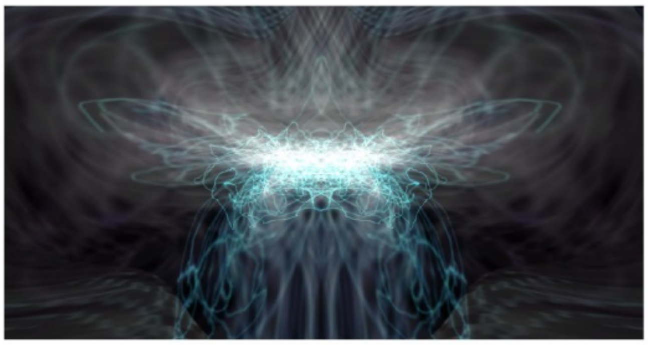


Figure 2: A Milkdrop preset (Inkblot by Geiss).

2.2.3 Karl Sims

Karl Sims developed a genetic algorithm that generates different images (not animations) by changing its own code based on human feedback [13]. The algorithm builds recursive lisp expressions using different functions. The used functions include normal lisp functions such as +, -, *abs()* and *sin()*, but also noise generation and image processing functions. Most of these functions can take images as input and give their output in the form of images too, so most can be seen as image processing functions. For example, the function *abs(X)* results in a gradient from white at the left of the image plane to black in the middle, back to white on the right of the plane again (X and Y coordinates range from -1 to 1). The algorithm can incorporate iterated functions systems in the expressions so in theory it should be able to generate the same visuals as the Electric Sheep algorithm. It can also generate waveforms and has numerous image processing functions available, so it should also be

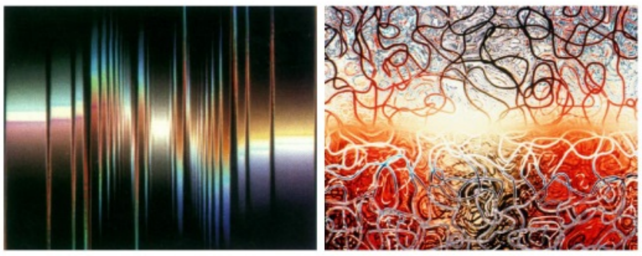


Figure 3: Two results of Sims' algorithm [12].

able to generate animations similar to Milkdrop presets. It could combine all this in one animation so in theory the diversity of the generated visuals should be superior to the other two algorithms described above. An example of the output can be seen in Figure 3. Karl Sims suggested several ways of animating the images:

- Allowing time to be an input parameter.
- Dissolving two different expressions of similar structure over time. This also makes it possible to gradually make a transition from one expression to another.
- Using an image as input in the expression, and changing this image over time.

Sims does not specify in detail how the fitness of the images is determined. It seems like each population as a whole is presented in a grid to the user. The user then selects which images should reproduce to form the next generation. Sims does not tell how many images the user can select or how many are presented in the grid, although an image in his paper [12] suggests a population of 20 images was used. If this is how Sims' selection of images actually works, he does not use a fitness score. Rather, some images form the 'seed' for the next generation while the other images simply 'die'.

The lisp expressions are represented as tree structures which can be traversed and changed by mutation and crossover methods.

Mutation can happen on each node with a probability inversely scaled to the size of the expression. Possible mutations are:

- Replacement by a new random expression.
- Random addition or subtraction.
- Replacement by another function (while retaining the function's arguments where possible).
- Jumping in or out of a function (so becoming the argument to a new function, or an argument of this node's function becomes this node).
- Become a copy of another child of the parent node.

Crossover can happen in two ways. One option is that a random node of one parent is replaced by a random node of the other parent. The other option is to copy the parent's trees where they are similar, and *dissolving* two nodes when they are different. This means that the node becomes a dissolve function with the two parents' nodes at that location as arguments. A linear interpolation is done between the result of the two nodes using a random blend factor between 0 and 1.

Estimations of the complexity of new expressions are made by summing pre-calculated computing times of each function in the expression. If this estimated execution time is considered to be too long, the expression is eliminated before being shown to the user.

It is unknown how fast this algorithm runs on current hardware. Sims needed a parallel supercomputer, but this was around the year 1990. It is not known how this translates to current hardware, although the fact that Sims was able to create a parallel implementation of his algorithm indicates that this algorithm could run on GPUs, which could make (a variation on) this algorithm fast enough to be animated in real-time.

2.3 Development

There are three main requirements for the algorithm that generates the animated art for this study:

- It should be able to generate a wide variety of animated art.
- It should be possible for the animation generation program to be described in a genetic representation (such as a parameter set, string representation or program tree).
- It should be able to render the animations in real-time. That is, it should be able to render at least 20 frames per second. Rendering in real-time makes it possible for the algorithm to adapt quickly to the given feedback instead of having to wait till the animations are pre-rendered.

In theory, Sims' algorithm seems to be able to generate the biggest variety of art, and it might be able to run in real-time. The disadvantage of Sims' algorithm is that Sims did not explain the used functions in detail. Rebuilding the algorithm in modern programming languages would thus be a complicated process and require lots of testing and experimenting.

The Electric Sheep algorithm can work in real-time with some work, but although it generates a wide variety of animations, the visual style of the generated animations is still very distinct. Therefore this study uses an algorithm that is inspired by the Milkdrop algorithm.

The base of Milkdrop are shapes and waveforms on top of which several image processing methods are used, which results in a big variety of animations. This same approach was used in this study. The shapes are formed using the Superformula of Johan Gielis [9], and several image processing methods such as blur, mirroring, fading, zoom and posterizing are used to add variety to the animations.

2.3.1 Superformula

The Superformula (1) created by Johan Gielis [9] is a formula that forms all kinds of shapes based on six parameters. The possible shapes range from circles and squares to stars and teardrops.

$$r(\theta) = \left(\left| \frac{cos(\frac{m\theta}{4})}{a} \right|^{n_2} + \left| \frac{sin(\frac{m\theta}{4})}{b} \right|^{n_3} \right)^{-\frac{1}{n_1}} \qquad (1)$$

The visual algorithm used for this study can generate multiple of these shapes and animate them by varying the six parameters of these shapes, as well as the position, rotation and colour of the shape over time. An example frame can

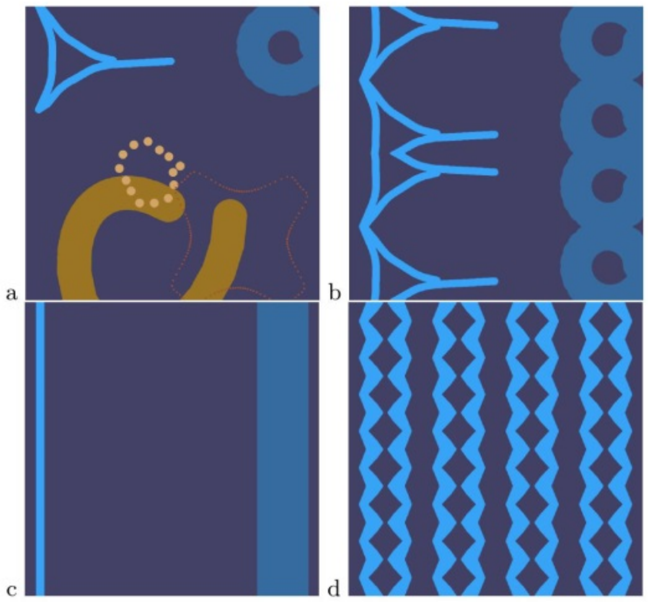


Figure 4: A frame with different Superformula shapes and its mirrored versions. a) Original frame. b) Mirrored two times along the y-axis. c) Mirrored ten times along y-axis. d) Mirrored three times along x-axis and four times along y-axis.

be seen in Figure 4a. Each shape can be drawn using a solid or dotted line and the width of the line is also variable. The fact that the shapes can be easily modified by varying these parameters makes it very suitable for use in a genetic algorithm.

2.3.2 Image processing

For each shape the algorithm can apply a certain amount of blur to the image. Doing this per shape makes it possible to have blurred shapes in the background with sharp shapes on top.

The algorithm can also mirror the image an arbitrary amount of times in two dimensions. For example, if the image is mirrored two times along the y-axis, it will contain four 'sections' that are all the same image but mirrored. This can be seen in Figure 4b. If instead the image is mirrored the maximum amount of times possible, only the first row of pixels is essentially shown along the whole image. This results in vertical lines covering the image (Figure 4c), while some combinations of mirror amounts along the X- and Y-axes can result in kaleidoscope-like visuals (Figure 4d)

Instead of completely redrawing each frame the algorithm can also fade the previous frames to a certain colour over time. This will introduce an extra sense of motion since the previous locations of shapes will still be visible for a certain time. The fade parameter can even become zero which means all previous frames will always be visible behind the current frame, creating some interesting effects.

The fading option can generate some kind of 'smoke' effect in combination with zoom. The algorithm can create a zoom effect by shrinking or enlarging the previous frame. This zoom can be varied in both horizontal and vertical directions and the center position of the zoom can also change. If the algorithm also fades the previous frame it creates a 'smoke' effect where a trail from each shape forms which 'drifts' towards or away from the zoom position. An example of this effect can be seen in Figure 5a.

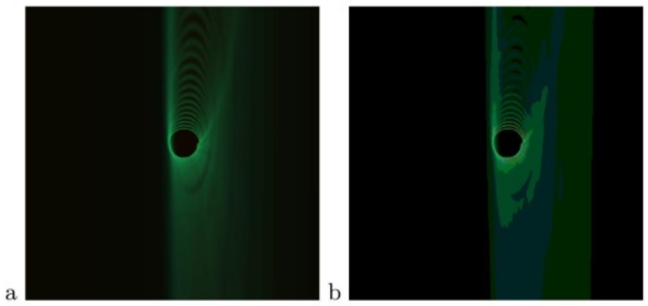


Figure 5: An animation with fade and zoom to create a smoke-like effect (a) and the same animation with a posterize effect applied (b).

The last image processing option the algorithm can apply is posterization, which reduces the colour resolution. For example, if each colour channel is reduced to four values, all colour values (on a scale from 0-255) will be converted to either 0, 85, 170 or 255. So an RGB value of (249, 94, 200) will become (255, 85, 170). This creates bigger areas of the same colour and increases contrast. An example can be seen in Figure 5b.

2.3.3 Parameters & genetic representation

The superformulas and image processing steps introduce quite some parameters. Some of these parameters are varied over time to animate the art. This is done by using five additional parameters to calculate a time-dependent parameter. This is done using the following formula:

$$P = p_1 - \frac{1}{2}p_2 \cdot (sin(p_3 t) + 1) + \frac{1}{2}p_4 \cdot (cos(p_5 t) + 1) \quad (2)$$

This means p_1 is a constant which is modified negatively by p_2 and p_3, and positively by p_4 and p_5. p_2 and p_4 set the maximum modification amount in negative or positive direction, while p_3 and p_5 set the period of the sine or cosine.

The parameters are still constrained, but just to make sure they do not make the program crash; the constraints are not used to steer the algorithm towards better looking art. When generating a random parameter set from scratch, the constraints are more strict to make sure the animation is not unnecessarily complicated and can be shown with a decent frame rate, but the genetic algorithm still has all freedom (within the non-crashing constraints) to change the parameters.

The set of all these parameters is directly used as a genetic representation of an animation. The genotype of each animation consists of 50 base parameters and 64 parameters per Superformula shape. The population is kept at a constant size of 10 individuals.

2.3.4 Crossover

Crossover is applied to two genotypes to create a new parameter set. The new parameter set is constructed by choosing randomly for each parameter which parent is the source for that parameter. For the parameters that are time-dependent the whole set of five parameters for calculating the time-dependent parameter are taken from the same parent.

2.3.5 Mutation

Each new genotype is mutated and each parameter in the set has a low probability to mutate. This can happen in two different ways: a 'soft' and a 'hard' mutation. A soft mutation means that the value of a parameter is slightly altered by adding or subtracting a small random amount. A hard mutation causes one of the bits in the parameter variable to flip. This can cause big differences in the actual value of the parameter. As suggested by Sims [13], the probability of a mutation decreasing complexity (lowering the value of a parameter) is slightly higher than the probability of increasing complexity. This prevents the parameter sets from drifting towards complex and hard to render animations without necessarily improving the results.

Selection.

The sets that are mated and/or mutated are chosen based on a fitness score. In this case this fitness score will be a direct result of the score the user gives to an animation, since a computer cannot rate the aesthetic beauty of art. How these scores are given is discussed and studied in the next section. The genotypes of the best scoring animations form a new population and are used to form new genotypes until that new population is filled again to a size of 10. The algorithm chooses just one genotype if it is the only one with a better score than the lowest score in the set. Otherwise it will choose the best two, and more if the second place in the fitness ranking is shared among multiple genotypes. If all genotypes have the lowest possible score, a complete new random population is generated; none of the genotypes survive in that case.

3. FEEDBACK METHODS

This section tries to answer the following two sub questions: "What criteria determine if a feedback method can be used as a reliable fitness measure?" and "Which method is best according to these criteria?". The first question is answered mainly by exploring studies done in the past in similar fields. The second question is answered by implementing and testing some feedback methods on users using the algorithm developed in the first part of this study.

3.1 Feedback scales

Research on preference feedback often focuses on recommender interfaces for things like movies, music or books on the internet [3]. These are interfaces on websites where people can rate these products to give other people an idea of what is good and what is bad. There are different ways in which this feedback is gathered, mainly differing on the scale. Facebook started with its famous unary scale, where people can only give 'thumbs ups'. Other common scales are binary (positive/negative), 5 or 10 point (often stars/half stars) or a 100 point scale. Another method that is less common works differently than the previous mentioned scales, because it presents the user with two options. The user then has to choose the best option of those two. Previous studies already tested which of these scales is best in several scenarios and according to several criteria. These studies can be used to find the criteria that determine if a feedback method is good or not and are discussed in the following two sections.

3.2 Reliability

It is possible to (accidentally) introduce a bias in feedback because of certain design choices for the feedback interface. Friedman and Amoo [6] mentioned different causes for a possible bias. Connotations can bias the results because the negative connotations might sound more negative to the participant than the positive connotations sound positive. This also applies to numeric values; there is a difference between having a scale range from -5 to 5 and from 0 - 10. People are more likely to use the lower end of the scale in the last case since the lower end is perceived more negative in the first case. Forcing a choice can bias the results if the amount of undecided people or people without an opinion is significant. Unbalanced scales, meaning there are more positive than negative points on the scale (or the opposite) can have a big influence on results. The order in which the scale is presented (does it start at the positive or negative side) also influences the reported answers, but it is difficult to say which one of the two has the smallest bias. This may also depend on the subject of the question. Garland [8] found that the absence of a mid-point on a scale can distort results, although it was not possible to tell which scale was most accurate.

The amount of points on the scale can also have an influence on the reliability of the scale. Too few points means a loss of information, and too many points increases variance while accuracy may not increase significantly. Friedman and Friedman [7] studied this problem and recommended the range between five and eleven points as most optimal. Friedman and Amoo [6] stated that this also depends on what the scale will be used for. If there is no reason to think participants will have a complicated opinion about something, even a three point scale might suffice.

3.3 Usability

Besides affecting the reliability of the scale, the amount of points also has an influence on users' behavior and satisfaction. Sparling and Sen [14] studied how different scales (unary, binary, five-point and 100-point) influenced the time it took users to rate an item, their cognitive load, and overall satisfaction with the scale. They found that overall the rating time of users increased with the amount of points on the scale. Cognitive load was estimated by measuring the reaction time of users on a secondary task: clicking a button as soon as it becomes red (on random intervals). The cognitive load was found to be lower for a unary scale compared to other scales, but differences between the other scale were not significant. Users preferred the binary and five-point scales, but liked the five-point scale best.

3.4 Criteria

Previous mentioned studies show there are two main criteria for feedback methods: reliability and usability. Reliability is influenced by different factors. For example:

- If the scale has connotations or not.
- If the user is forced to make a choice or not.
- If the scale is balanced or not.
- If the resolution of the scale is appropriate for the subject.
- If it is clear to every user how the scale should be used (is the scale interpreted the same by all users?)

The usability can also be influenced by a lot of factors, for example:

- The time it takes the average user to rate using the scale.
- The cognitive load the scale needs from the user.
- Users' understanding of the scale.
- If the scale distracts the user from the content.
- How the response is recorded (on a screen, paper, etc.).

3.5 Implementation

The possible feedback methods mentioned in section 3.1 were all implemented in the algorithm, except for the unary scale. The unary scale was considered to function a lot like a binary scale in this context and thus unnecessary to implement if the binary scale was also implemented. The implementation of each scale is described below.

3.5.1 Best of two

Best of two shows two animations at the same time, next to each other. New animations start with a score of one. Five random pairs are formed and are shown to the user one pair at a time. The user then chooses which one is preferred over the other by clicking on that animation. The score of the preferred animation will increase with one, the other animation's score is decreased by one. The maximum score is two. After all five pairs are shown, the mating process will start and a new population is formed which is again shown to the user.

3.5.2 Binary

The order of the population is randomized and the user is shown each animation one time. The user can indicate his preference by pressing the arrow up or arrow down keys on the computer keyboard, after which the next animation is shown. Pressing arrow up increases the animation's score by one, pressing down decreases the score by one. The maximum score is also two here. After all animations are shown, a new population is formed by the mating process.

3.5.3 Scale of five/ten

The order of the population is randomized and the user is shown each animation one time. The user can indicate his preference for each animation by pressing the number keys on the computer keyboard. The zero key represents a score of ten in case of a scale of ten points. The given amount of points translates directly to the score of each animation. After all ten animations are rated, the mating process is started to create a new population which can be rated by the user again.

3.5.4 Slider

The order of the population is randomized and the user is shown each animation one time. A slider is shown to the right of the animation, and starts at its middle value of 50. This value is not shown, the user only sees the position of the slider. Users can change the slider's position by clicking and dragging with the mouse or by scrolling with the mouse. The value represented by the slider's position is in the 0-100 range and this is directly translated to the animation's score. If the user is satisfied with the position of the slider, he can hit any key to be shown the next animation. After all ten animations are shown, a new population is generated by the mating process which can be rated again by the user.

3.6 Selection

A selection of the implemented feedback methods was made to narrow down the amount of methods to be tested on users. This was done by personally testing the different methods to get a basic idea of which ones would probably give the best results.

Best of two seemed to work quite well for the genetic algorithm, although sometimes the user was forced to choose between two very good animations, which often resulted in a good animation being thrown out of the population. Being shown two animations next to each other was found to sometimes be quite overwhelming. It might even be possible that some animations would have been rated better if they were shown on their own instead of together with another animation. Furthermore, slow rendering animations brought down the framerate of both animations. This could cause animations that were on themselves very fast rendered to be rated worse because of the low framerate. This could be solved by rendering each animation on a different machine. An advantage of this method was found in the speed of rating; it is often very easy and fast to determine which of two animations is preferred over the other.

The binary system does not have the disadvantage that users can be forced to down-rate a good animation. This method was also found to be quite quick in rating speed, although it seemed to be a bit slower than the 'best of two' method. Ratings seemed to be very much relative to the animations that were already seen; as other animations became better or the user got bored by seeing the same (or similar) animations multiple times, he started giving negative feedback on animations that he previously rated positive.

The scales were easy to use. The scale of five points already had enough points to express an opinion properly, the added resolution of the ten point scale was not used very much. The speed of rating was slower than the binary method, but did not differ a lot between both scales.

The slider did not work that well. The user found it difficult to relate the scores he was giving with the scores given to previous animations since he did not see the actual score that was given, just the position of the slider. It was therefore possible that an animation that was actually preferred over another was thrown out of the population because the user unknowingly rated it a couple of points lower than the other animation. Because of these problems with recalling scores given to other animations, the rating speed seemed to be the lowest of all methods.

It was decided that the binary and scale with five points were the best options to test on other test subjects. The binary method seemed to be slightly faster to use, while the scale offers more resolution for expressing an opinion properly. These two scales were also most preferred in Sparling & Sens's study [14]. In this study the five-point scale was most preferred of the two. The ten point scale did not have advantages over the five points scale and the other two methods were found to have too many disadvantages to be of good use for this application.

3.7 User testing

The two selected feedback methods were tested on users to find out which of the two has the best usability in this context.

3.7.1 Method

Each participant individually tests both scales by rating 30 animations using each scale. This means each participant will see and rate three complete generations of animations produced by the genetic algorithm. Two fixed sets of ten animations are used as the first generation. The order in which the scales are tested and which set of animations is used as first generation is randomly assigned. Before the participants start rating the animations, it is explained how to rate the animations using the scale that is being tested. After the first 30 animations, the participants get an explanation about the other scale and will rate 30 animations using that scale.

After testing both scales participants are asked which of the modes they preferred, and explain why. They are also asked what they thought about the resulting animations and if they thought one scale produced better animations than the other.

The test is performed on a laptop with a 14 inch screen with a resolution of 1600x900. The resolution of the animations is 768x768 and these are displayed in the middle of the screen surrounded by a black border.

3.7.2 Participants

Ten participants tested both feedback methods. These participants were all students at the University of Twente, and all were studying in a technology-oriented field. Three participants were female, seven were male.

3.7.3 Results

There was no majority of participants who preferred one of the two methods; five participants preferred the binary scale, the other five preferred the five point scale. Of the participants who preferred the binary scale, three said it allowed them to go with their 'gut feeling', whereas with the five-point scale they had to consciously think about their rating. One participant argued that he did not use the extra resolution of the five point scale anyway, but only used it to express a positive or negative rating and not much in between. Another participant did not use the extreme points of the scale (one and five) at all. One participant said that it allowed him to enjoy the animations more, with the five point scale he was really focused on his own tastes and rating behaviour.

The participants who preferred the five point scale all indicated that they thought they needed the extra resolution and could not express their opinion sufficiently with the binary scale. It seemed like these participants were more focused on their own voting behaviour, where the participants preferring the binary scale were more interested in the animations.

Whether or not participants thought one scale produced better animations than the other differed very much between participants and it did not seem to be related to their preference of one method.

Participants indicated that quite some of the generated animations were still a bit 'rough'; some animations had very fast movements, fast flickering colours, colour schemes with a very low contrast or did not show much at all. Some participants also indicated that they still missed some diversity and thought the animations were quite similar.

3.8 Conclusion & Discussion

The best feedback method.
Based on this study, it is not possible to say if a binary scale or a scale of five point is best for feedback on animated art. Exactly half of the participants preferred the binary scale while the other half preferred the five point scale and there is no big difference in terms of reliability.

Difference between the scales.
In terms of reliability both scales are quite similar; the biggest difference is that the binary scale has a clear negative part (you either rate something 'up' or 'down'), where the five-point scale only has positive ratings (from one to five points). This means the results of one scale might be shifted either positively or negatively relative to the other scale, but it is impossible to know which one gives the most reliable results.

Which scale scores best in terms of usability seems to depend on the context and the user. It seems like the two different scales both have their advantages and disadvantages, and depending on how much value someone gives to each (dis)advantage, their preference is different. The binary scale is considered fast, easy to use and requires little mental effort, but has a low resolution. The five point scale offers more room for people to express their opinion, but this also seems to introduce a higher mental effort and can make the decision on which rating to choose slower. Therefore, if speed and ease of use is most important, it would make sense to use a binary scale. If precision is more important, the five point scale seems to be the better choice. As one participant said, it would make sense to use the binary scale when exploring the animated art, while the five points scale is better suited for fitting the generated art to one person's tastes, or for finding out what that person's preference in art is about.

Since the subject group was small and because the random nature of the genetic algorithm makes it hard to make sure the test environment is exactly the same for each participant, these results can merely be an indication of which method to use in a certain situation. However, an earlier study already indicated that these two methods are indeed most preferred by people in a different context and similar conclusions were named in this study [14].

Users' opinions about the art itself.
It seems like the visual algorithm can still be improved. Users indicated that they thought there was a lack of diversity and too many 'rough' and 'ugly' animations. The lack of diversity can be improved by working with more than one population at the same time, or the algorithm can make bigger 'steps' between animations by increasing mutation chances or introducing random parameter sets that are not related to the previous population. In the first case the populations can even be split across multiple users, so one user rates one half of each population and the other user rates the other half. This should result in art that is preferred by both users, but making this work properly with users with a big difference in art preference might be an interesting case for future work.

4. ART IMPROVEMENT FROM FEEDBACK

As explained earlier, the developed visual algorithm uses a simple form of a genetic algorithm, so the feedback from users is already used to improve the art. This makes it possible to test how effective the feedback methods from previous section are in actually improving the art, and get an informal answer to the question "Does this feedback method indeed improve animated art?".

4.1 Method

Since the used genetic algorithm is still very simple, the effectiveness of the feedback is tested only informally in this study; a proper test of effectiveness needs a more sophisticated study of what the best genetic algorithm is for this application or if other methods such as machine learning need to be added. The effectiveness of the feedback is tested by letting one person rate 50 generations (populations) of animations using the five point scale. This scale is used since it gives a clear rating number that can directly be used to see if the rating of the art improves through generations. The first generation is generated randomly.

4.2 Results

The resulting ratings for each generation can be seen in Figure 6. Note that at generation 22, all animations were rated with 1, which means that generation 23 is a completely random generated generation, no animations from generation 22 survived.

The test subject noted that the algorithm had the tendency to sometimes focus on one particular animation and make all kinds of variations on that one. Seeing a lot of slight variations of the same animation all the time got the test subject bored which caused him to drop the rating of the variations of that animation. This was the main cause for a complete generation receiving a rating of 1.

New animations in a generation (originated from mating surviving animations) were often rated with 1 because they often contained flickering colours, very fast movement or only showed a background colour without any moving shapes.

4.3 Conclusion & Discussion

Effectiveness of feedback.

The feedback seems to be effective in improving the art, even with a simple genetic algorithm. The later generations got considerably more high ratings, and the last generations did not even receive ratings of two points.

Avoiding low ratings.

There were always some animations that received a rating of one. These were often the 'rough' kind of animations that were also found in the user tests of the feedback methods; without any movement, flickering or with fast repetitive movements. The results show that these animations have a big influence on the average rating: in every population there are some animations which receive a rating of one. To avoid this, animations that are rated negatively by (almost) all people should preferably not be shown at all. This could be done by letting the genetic algorithm form a concept of what people generally do not like (using machine learning, for example), and only show animations that pass this 'test'. However, this test should still allow the algorithm to show art that does not look like art it produced before. A genetic algorithm that can behave like that is probably far more advanced than the one used here and the design of such an algorithm could be a very interesting subject for future work.

Keep exploring art.

The results also show another important point that could be improved: the current algorithm converges quite fast to animations with good ratings and does not easily explore new directions once it gets 'stuck' with a certain type of animation. Exploring new directions when one or multiple forms of 'good art' are already found introduces the risk of generating art that is of much lower quality than the already found art. This makes it even more important to have a filtering system in place as described above.

Consistency.

The test showed that users are not consistent in their feedback. When the user was shown a lot of similar animations, he started to rate the animations lower and lower until he got shown new animations. Such a 'rating drift' can also be caused because other animations are becoming better; the ratings seem to be partly relative to the animations already shown to the user.

5. FUTURE WORK

The next step is to develop a fully working and advanced version of this (still rather primitive) algorithm. It should generate and keep on exploring a vast diversity of animated art while minimizing the presentation of art that is disliked by the public watching the art. The conclusions of this study can be used for this:

Feedback methods.

Depending on the application of the algorithm, one of the feedback methods can be used. For example, if the algorithm will be used in a physical art installation, the binary scale will be more useful to minimize the effort passersby have to put into rating the art. However, if the algorithm is used in a computer program or website that tries to show the user art that is best suited to that user's tastes (based on the tastes of people that voted similar to the user, for example), the five point scale is probably better.

Filtering.

Machine learning based filtering of the generated art is needed to minimize the amount of generally disliked animations that are shown to users. The challenge with a filtering system like this is to learn what art should be filtered in such abstract ways that art that is nothing like what has been generated before can also still pass this 'test'.

Diversity and exploration.

The diversity of the generated art needs to be increased and the algorithm needs to explore more. This can be done in several ways, for example:

- Using multiple populations.
- Split populations among multiple users.
- Increase mutation.
- Introduce completely new parameter sets that are not related to previous sets.
- Using a different algorithm to generate the visuals. For example, Sim's algorithm [13] is superior in terms of the diversity of its output. It might be worth it to see

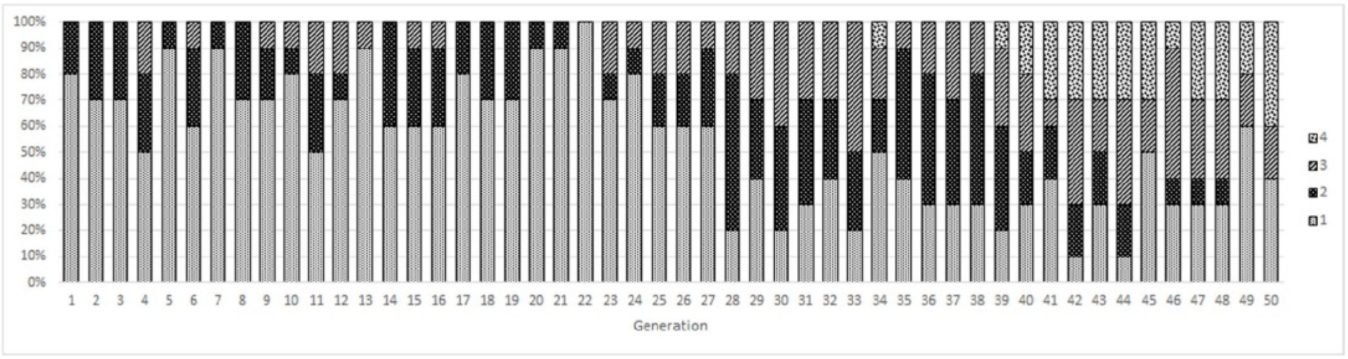


Figure 6: The resulting ratings per generation. A rating of 5 was never given.

if a modern implementation of the algorithm can run in real-time to generate animations.

Rating consistency and bias.

The ratings always being (partly) relative to the animations already shown makes the rating itself quite unreliable. The rating given to an animation is not always the same as the rating that would have been given to the animation when it would have been shown on its own. It is very hard to reduce this effect and make the rating more reliable, the rating will never be 100 percent reliable.

Also, there is probably always a bias between what the user thinks of the art and how he rates it, mainly introduced by how the user understands the rating scale. This is not a big problem if there is only one user using the algorithm since the bias is constant, but might become more troublesome if the system is used by multiple users. Two users giving the same rating to an animation does not necessarily mean they value its aesthetic beauty the same.

These two points make the rating less reliable but are both very difficult if not impossible to eliminate. How can a genetic algorithm deal with this unreliable feedback and still give good results? This is an interesting question for future work.

6. REFERENCES

[1] P. Bentley and D. Corne. *Creative evolutionary systems.* Morgan Kaufmann, 2002.

[2] M. A. Boden and E. A. Edmonds. What is generative art? *Digital Creativity*, 20(1-2):21–46, 2009.

[3] D. Cosley, S. K. Lam, I. Albert, J. A. Konstan, and J. Riedl. Is seeing believing?: how recommender system interfaces affect users' opinions. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 585–592. ACM, 2003.

[4] S. Draves. The electric sheep screen-saver: A case study in aesthetic evolution. In *Workshops on Applications of Evolutionary Computation*, pages 458–467. Springer, 2005.

[5] S. Draves and E. Reckase. The fractal flame algorithm. *Forthcoming, available from http://flam3.com/flame.pdf*, 2003.

[6] H. H. Friedman and T. Amoo. Rating the rating scales. *Friedman, Hershey H. and Amoo, Taiwo (1999)." Rating the Rating Scales." Journal of Marketing Management, Winter*, pages 114–123, 1999.

[7] H. H. Friedman and L. W. Friedman. On the danger of using too few points in a rating scale: a test of validity. *Friedman, Hershey H. and Friedman, Linda Weiser ((1986)." On the Danger of Using too few Points in a Rating Scale: A Test of Validity." Journal of Data Collection, 26 (2), 60-63*, 1986.

[8] R. Garland. The mid-point on a rating scale: Is it desirable. *Marketing bulletin*, 2(1):66–70, 1991.

[9] J. Gielis. A generic geometric transformation that unifies a wide range of natural and abstract shapes. *American journal of botany*, 90(3):333–338, 2003.

[10] J. H. Holland. *Adaptation in natural and artificial systems. An introductory analysis with application to biology, control, and artificial intelligence.* University of Michigan Press.

[11] O. S. Lawlor. Gpu-accelerated rendering of unbounded nonlinear iterated function system fixed points. *ISRN Computer Graphics*, 2012, 2012.

[12] K. Sims. *Artificial evolution for computer graphics*, volume 25. ACM, 1991.

[13] K. Sims. Interactive evolution of equations for procedural models. *The Visual Computer*, 9(8):466–476, 1993.

[14] E. I. Sparling and S. Sen. Rating: how difficult is it? In *Proceedings of the fifth ACM conference on Recommender systems*, pages 149–156. ACM, 2011.

Appendix C - Animation algorithm function set

All vector -> vector functions are executed elementwise unless indicated differently. A1, a2, a3, etc. are references to the input arguments of the functions.

Function name	Arguments	Output	Returns
abs	1 float	1 float	the absolute value of a1
abs	1 vector	1 vector	the absolute value of a1
and	2 floats	1 float	a1 & a2
and	2 vectors	1 vector	a1 & a2
blur	2 floats, 1 image, 2 vectors	1 vector	the pixel colour at coordinates (a1,a2) of the blurred image a3 with a blur size of a4 and sigma a5 (per RGB colour)
clamp	3 floats	1 float	a1, clamped by a2 as minimum and a3 as maximum value
clamp	3 vectors	1 vector	a1, clamped by a2 as minimum and a3 as maximum value
cos	1 float	1 float	cos(a1)
cos	1 vector	1 vector	cos(a1)
cosn	1 float	1 float	cos(a1) with input and output normalized to 0-1
cosn	1 vector	1 vector	cos(a1) with input and output normalized to 0-1
cross	2 vectors	1 vector	the cross product of a1 and a2
divide	2 floats	1 float	a1 / a2
divide	2 vectors	1 vector	a1 / a2
exp	1 float	1 float	the natural exponentiation of a1
exp	1 vector	1 vector	the natural exponentiation of a1
exp2	1 float	1 float	2^a1
exp2	1 vector	1 vector	2^a1
grad_dir	2 floats, 1 image	1 vector	the gradient direction at coordinates (a1,a2) of image a3 per RGB colour
grad_mag	2 floats, 1 image	1 vector	the gradient magnitude at coordinates (a1,a2) of image a3 per RGB colour
hsv_to_rgb	1 vector	1 vector	an RGB output for the HSV input
iff	3 floats	1 float	a2 if a1 <= 0.5, else a3

iff	3 vectors	1 vector	a2 if a1 <= 0.5, else a3
ifs	62 floats	1 vector	the pixel colour at coordinates (a1,a2) of a fractal flame rendered with a3 variations, exposure a4, texture scale a5, initial value a6, initial size a7, and for each used variation: (variation type a8, weight a9, scale a10, hue a11, saturation a12, X.x a13, X.y a14, Y.x a15, Y.y a16, O.x a17, O.y a18). These last inputs are repeated 4 more times for the other possible variations
log	1 float	1 float	logarithm of a1 (base 10)
log	1 vector	1 vector	logarithm of a1 (base 10)
log2	1 float	1 float	logarithm of a1 (base 2)
log2	1 vector	1 vector	logarithm of a1 (base 2)
logg	2 floats	1 float	logarithm of a1 (base a2)
logg	2 vectors	1 vector	logarithm of a1 (base a2)
max	2 floats	1 float	the maximum of both inputs
max	2 vectors	1 vector	the maximum of both inputs
min	2 floats	1 float	the minimum of both inputs
min	2 vectors	1 vector	the minimum of both inputs
minus	2 floats	1 float	a1 - a2
minus	2 vectors	1 vector	a1 - a2
mix	3 vectors	1 vector	the linear interpolation between a1 and a2 at point a3 (ranged between 0-1)
mix	3 floats	1 float	the linear interpolation between a1 and a2 at point a3 (ranged between 0-1)
mod	2 floats	1 float	a1 % a2
mod	2 vectors	1 vector	a1 % a2
mod	1 vector, 1 float	1 vectors	a1 % a2
normalize	1 vector	1 vector	the unit vector of a1
or	2 floats	1 float	a1 | a2
or	2 vectors	1 vector	a1 | a2
plus	2 floats	1 float	a1 + a2
plus	2 vectors	1 vector	a1 + a2

pow	2 floats	1 float	a1^a2
pow	2 vectors	1 vector	a1^a2
product	2 floats	1 float	a1 * a2
product	2 vectors	1 vector	a1 * a2
round	1 float	1 float	a1 rounded to the closest whole number
round	1 vector	1 vector	a1 rounded to the closest whole number
sign	1 float	1 float	1 if a1 is positive, else -1
sign	1 vector	1 vector	1 if a1 is positive, else -1
sin	1 float	1 float	sin(a1)
sin	1 vector	1 vector	sin(a1)
sinn	1 float	1 float	sin(a1) with input and output normalized to 0-1
sinn	1 vector	1 vector	sin(a1) with input and output normalized to 0-1
sqrt	1 float	1 float	the square root of a1
sqrt	1 vector	1 vector	the square root of a1
step	2 floats	1 float	0 if a2 < a1, else 1
step	2 vectors	1 vector	0 if a2 < a1, else 1
vec3	3 floats	1 vector	the vector containing the 3 input floats
xor	2 floats	1 float	a1 ^ a2
xor	2 vectors	1 vector	a1 ^ a2

Appendix D - Results of single user network setups

Recall 3+ means the recall of the animations that were rated 3 or higher.

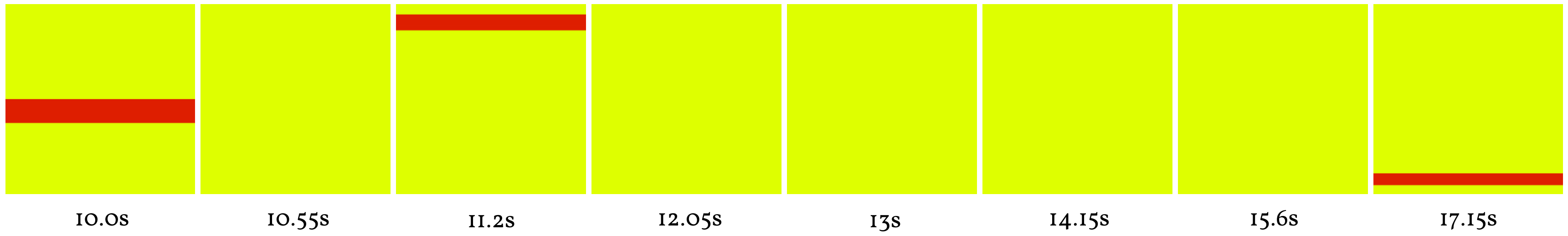
Setup	Learning rate	Momentum	Alpha	Neurons	Precision	Recall	Recall 3+	F_1 3+
0	0.1	0	1	14	0.717472	0.742308	0.830435	0.769832
1	0.1	0	1	21	0.722282	0.779231	0.891304	0.79794
2	0.1	0	1	28	0.696896	0.768462	0.847826	0.764987
3	0.1	0	2	14	0.737374	0.701923	0.847826	0.788752
4	0.1	0	2	21	0.764403	0.627692	0.804348	0.783867
5	0.1	0	2	28	0.749195	0.626154	0.830435	0.787726
6	0.1	0.1	1	14	0.700144	0.748077	0.895652	0.785922
7	0.1	0.1	1	21	0.716226	0.770769	0.891304	0.794231
8	0.1	0.1	1	28	0.744987	0.757308	0.852174	0.794984
9	0.1	0.1	2	14	0.755418	0.656923	0.778261	0.766669
10	0.1	0.1	2	21	0.732454	0.662308	0.865217	0.79332
11	0.1	0.1	2	28	0.734494	0.665	0.891304	0.805337
12	0.1	0.5	1	14	0.72952	0.760385	0.86087	0.789771
13	0.1	0.5	1	21	0.727764	0.777308	0.869565	0.79237
14	0.1	0.5	1	28	0.723451	0.754615	0.847826	0.780716
15	0.1	0.5	2	14	0.73353	0.668077	0.847826	0.786548
16	0.1	0.5	2	21	0.738854	0.669231	0.843478	0.787707
17	0.1	0.5	2	28	0.753697	0.627308	0.878261	0.811225
18	0.01	0	1	14	0.744534	0.785769	0.891304	0.811335
19	0.01	0	1	21	0.738703	0.779615	0.891304	0.80786
20	0.01	0	1	28	0.744039	0.768077	0.891304	0.811041
21	0.01	0	2	14	0.800307	0.400769	0.569565	0.665503
22	0.01	0	2	21	0.772873	0.370385	0.6	0.675552
23	0.01	0	2	28	0.7609	0.416154	0.569565	0.651475
24	0.01	0.1	1	14	0.746067	0.784231	0.891304	0.812244
25	0.01	0.1	1	21	0.745171	0.771538	0.891304	0.811713
26	0.01	0.1	1	28	0.742636	0.775769	0.891304	0.810207
27	0.01	0.1	2	14	0.785567	0.439615	0.626087	0.696818
28	0.01	0.1	2	21	0.790535	0.346923	0.604348	0.685015
29	0.01	0.1	2	28	0.789374	0.32	0.569565	0.661692
30	0.01	0.5	1	14	0.707598	0.709231	0.891304	0.788898
31	0.01	0.5	1	21	0.715614	0.710385	0.847826	0.77613
32	0.01	0.5	1	28	0.705069	0.706154	0.86087	0.775219
33	0.01	0.5	2	14	0.79064	0.370385	0.591304	0.676596
34	0.01	0.5	2	21	0.791251	0.396538	0.626087	0.699046
35	0.01	0.5	2	28	0.772152	0.398846	0.66087	0.71219

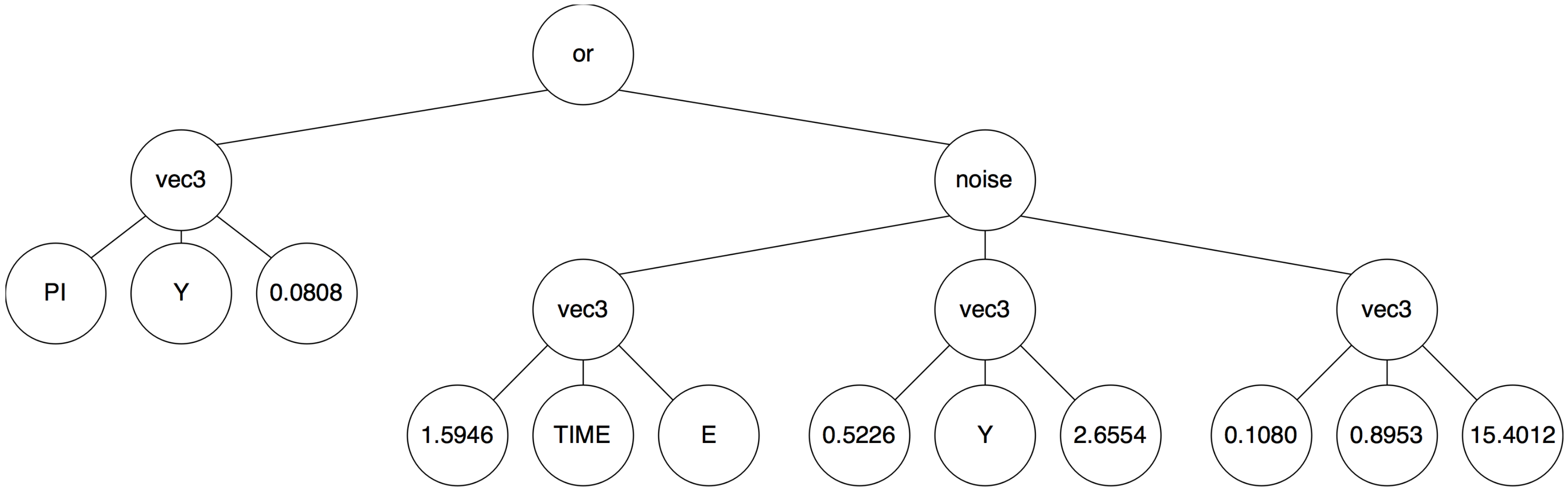
Appendix E - Results of multi user network setups

Recall 3+ means the recall of the animations that were rated 3 or higher.

Setup	Learning rate	Momentum	α	Neurons	Precision	Recall	Recall 3+	F_1 3+	Max F_1 3+
0	0.1	0	1	14	0.757	0.836	0.870	0.810	0.821
1	0.1	0	1	21	0.756	0.832	0.870	0.809	0.824
2	0.1	0	1	28	0.758	0.829	0.867	0.809	0.825
3	0.1	0	2	14	0.749	0.820	0.858	0.800	0.819
4	0.1	0	2	21	0.751	0.801	0.833	0.790	0.819
5	0.1	0	2	28	0.749	0.819	0.856	0.799	0.820
6	0.1	0.1	1	14	0.749	0.819	0.856	0.799	0.820
7	0.1	0.1	1	21	0.759	0.844	0.880	0.815	0.824
8	0.1	0.1	1	28	0.754	0.841	0.881	0.812	0.824
9	0.1	0.1	2	14	0.751	0.810	0.838	0.793	0.816
10	0.1	0.1	2	21	0.750	0.803	0.841	0.793	0.818
11	0.1	0.1	2	28	0.755	0.800	0.833	0.792	0.820
12	0.1	0.5	1	14	0.756	0.847	0.883	0.815	0.824
13	0.1	0.5	1	21	0.752	0.842	0.878	0.810	0.823
14	0.1	0.5	1	28	0.758	0.857	0.895	0.820	0.825
15	0.1	0.5	2	14	0.755	0.804	0.839	0.795	0.820
16	0.1	0.5	2	21	0.751	0.833	0.872	0.807	0.823
17	0.1	0.5	2	28	0.759	0.826	0.859	0.805	0.820
18	0.01	0	1	14	0.755	0.875	0.903	0.823	0.823
19	0.01	0	1	21	0.754	0.872	0.900	0.821	0.821
20	0.01	0	1	28	0.754	0.870	0.898	0.820	0.821
21	0.01	0	2	14	0.753	0.856	0.890	0.816	0.825
22	0.01	0	2	21	0.754	0.855	0.891	0.817	0.825
23	0.01	0	2	28	0.754	0.864	0.900	0.821	0.824
24	0.01	0.1	1	14	0.754	0.876	0.903	0.822	0.822
25	0.01	0.1	1	21	0.754	0.874	0.902	0.821	0.821
26	0.01	0.1	1	28	0.755	0.873	0.901	0.821	0.822
27	0.01	0.1	2	14	0.755	0.863	0.896	0.819	0.823
28	0.01	0.1	2	21	0.756	0.859	0.898	0.820	0.824
29	0.01	0.1	2	28	0.755	0.861	0.898	0.820	0.825
30	0.01	0.5	1	14	0.748	0.879	0.904	0.818	0.820
31	0.01	0.5	1	21	0.750	0.878	0.903	0.819	0.819
32	0.01	0.5	1	28	0.751	0.877	0.902	0.820	0.820
33	0.01	0.5	2	14	0.757	0.868	0.903	0.824	0.825
34	0.01	0.5	2	21	0.756	0.872	0.908	0.825	0.826
35	0.01	0.5	2	28	0.755	0.870	0.906	0.824	0.824

Appendix F - Example animations and their function trees



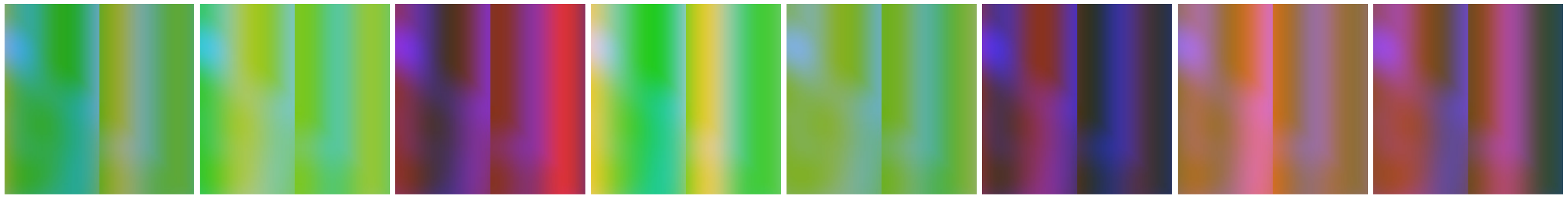


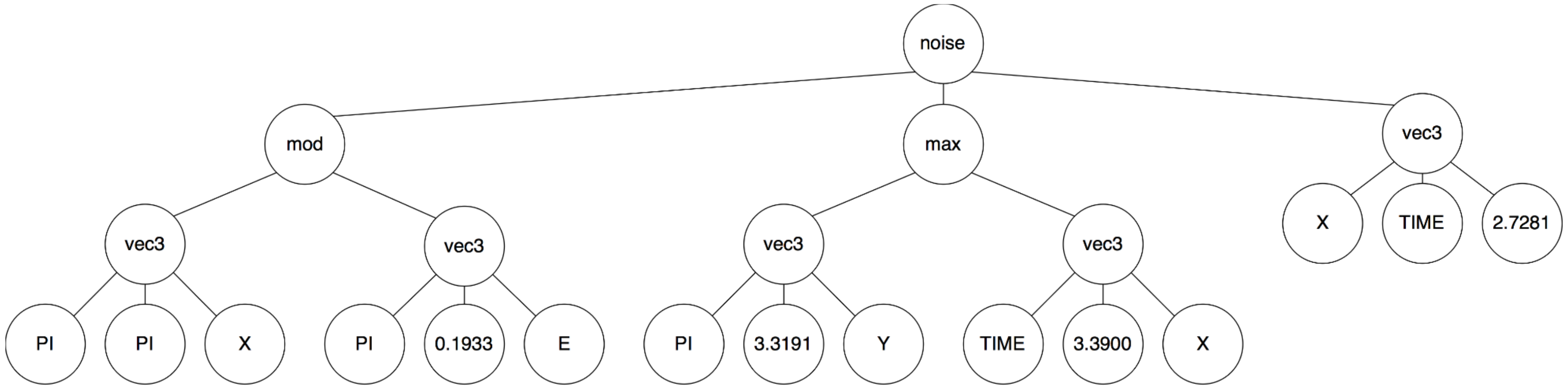
Functions

'or' – A bitwise 'OR' operating on the bits of the floating-point values in the vectors of its input.

'noise' – 3D simplex noise. The inputs are the coordinates for the noise.

'vec3' – Creates a vector from three floating-point values.


10.0s 10.55s 11.2s 12.05s 13s 14.15s 15.6s 17.15s



Functions

'noise' - 3D simplex noise. The inputs are the coordinates for the noise.

'max' – Returns the maximum value of each vector value (elementwise).

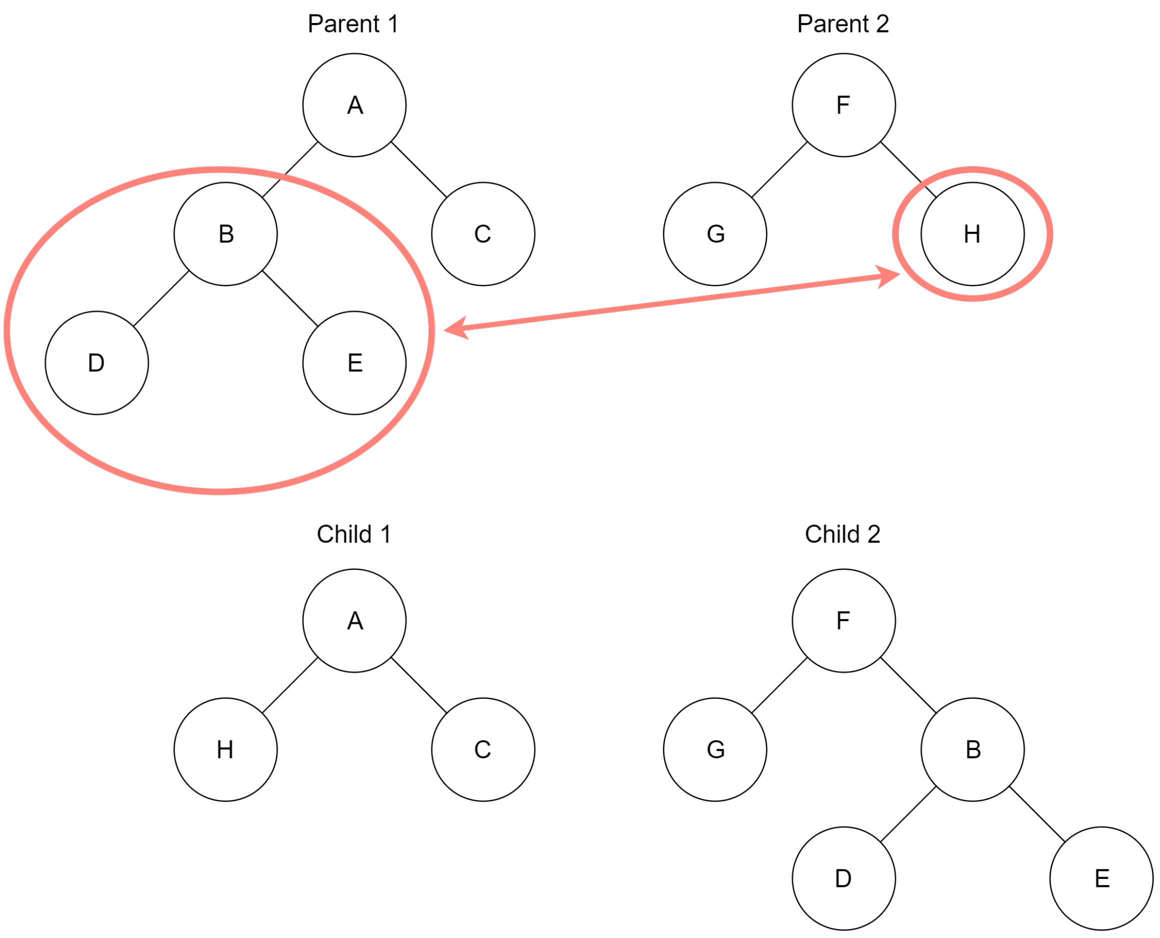
'mod' – The modulo operation. Also works elementwise.

'vec3' - Creates a vector from three floating-point values.

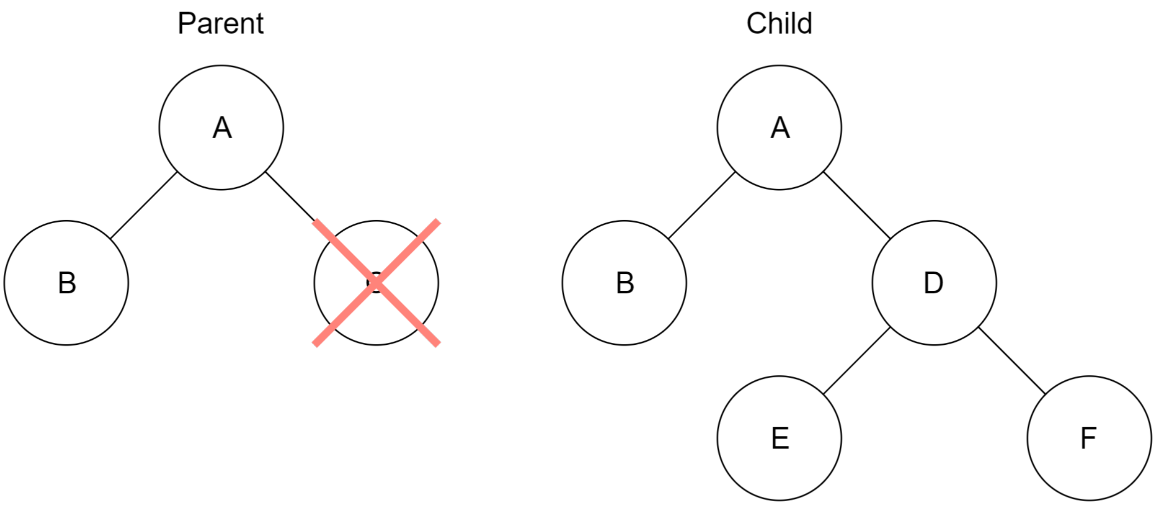
Appendix G - Examples of genetic operations

The three basic genetic operations are shown below on example trees

Crossover



Mutation



Reproduction