Using Scalable Vector Graphics to evolve art

Eelco den Heijer*

Vrije Universiteit Amsterdam, Dept. of Computer Science, Amsterdam, Netherlands, E-mail: eelcodenheijer@gmail.com *Corresponding author

A.E. Eiben

Vrije Universiteit Amsterdam, Dept. of Computer Science, Amsterdam, Netherlands, E-mail: gusz@cs.vu.nl

Abstract In this paper we describe our investigations of the use of Scalable Vector Graphics as a genotype representation in evolutionary art. We describe the technical aspects of using SVG in evolutionary art, and propose and explain our custom, SVG specific operators initialisation, mutation and crossover. Furthermore, we compare the use of SVG with existing representations in evolutionary art. We perform two series of experiments and describe their setup and results. In the first series of experiments we investigate the feasibility of SVG as a genotype representation for evolutionary art, and evolve abstract images using a number of aesthetic measures as fitness functions. We found that SVG is suitable as a genotype representation for evolutionary art, but that the range of the visual output was limited by the design of our genetic operators. In order to increase the range of the visual output, and in order to evolve representational images, we performed a second series of experiments in which we used existing images as source material. We designed and implemented a new initialisation, crossover and mutation operator. We also designed and implemented an ad-hoc aesthetic measure for 'pop-art' and used this to evolve images that are visually similar to screen printing art and pop art. All experiments described in this paper are done without a human in the loop. All images and SVG code examples in this paper are available at
`http://www.eelcodenheijer.nl/research`

Keywords: Evolutionary computation, genetic programming, evolutionary art, SVG, scalable vector graphics

Reference to this paper should be made as follows: den Heijer, E., Eiben, A.E.. (xxxx) 'Using Scalable Vector Graphics to evolve art', *Int. J. Art and Technology* , Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Eelco den Heijer is a PhD candidate with the VU University Amsterdam, the Netherlands. His PhD research concerns

various aspects of evolutionary art, including the unsupervised evolution of aesthetically pleasing images using aesthetic measures, the use of alternative genotype representations, and the improvement of population diversity in evolutionary art systems. Other research interest include machine learning, machine vision, computer graphics and non-photorealistic rendering.
A.E. (Gusz) Eiben is Professor of Computational Intelligence on the VU University Amsterdam. He is one of the European early birds of evolutionary computing. Since his first publication in the area (1989) he has published numerous papers, book chapters and the first comprehensive text book, Introduction to Evolutionary Computing (together with J.E. Smith). His recent research interests include evolutionary art, parameter calibration and evolutionary robotics.

1 Introduction

Over the last two decades, evolutionary art (EvoArt) has developed from an experimental mix of computer art and evolutionary algorithms to an established research topic in evolutionary computation. Although there has been significant progress in various aspects of EvoArt (notably in the field of interactive evolutionary computation, or IEC (Takagi, 2001)) one cannot deny that some aspects of EvoArt appear to be stuck in a local optimum; perhaps the most visible aspect is that a lot of EvoArt looks like ... computer art.

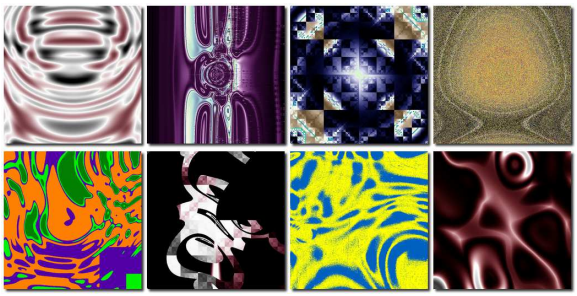


Figure 1: A portfolio of eight images evolved using symbolic expressions, from den Heijer et al (den Heijer & Eiben, 2010a,b, 2011; den Heijer, 2012)

In Figure 1 we see a number of images that are the result of previous experiments with the evolution of images using and expression based representation (den Heijer & Eiben, 2010a,b, 2011; den Heijer, 2012). We see a variety of images, but almost all images are abstract "textures". When we take a wider view, and regard different artworks of centuries, it is evident that artists over centuries have experimented with art materials, layouts, subjects, techniques etc. All this has resulted in a wide variety of visual output. If we project this observation onto the world of EvoArt, one could conclude that the field might benefit (in terms of variety of visual output) of new representations and new techniques. In this paper we want

to add a new technique to the world of EvoArt; the use of Scalable Vector Graphics or SVG and compare our new technique with an established technique (the use of symbolic expressions). SVG is a XML based markup language for vector graphics, maintained by the World-Wide Web consortium(W3C, 2005).
Many existing EvoArt system use the so-called 'raster paradigm' that was pioneered by Karl Sims in 1991 (Sims, 1991). In the 'raster paradigm' images are evolved by iteratively calculating the colour of each pixel in a raster using the outcome of a function tree (the next section gives a short explanation of the 'raster-paradigm').
Our motivation for investigating a new genotype representation is twofold; first of all, we think that the raster paradigm limits EvoArt systems in the range of their visual output; the visual output of these EvoArt systems is mostly limited to 'texture' images. The second motivation follows from the first; using symbolic expression with the 'raster-paradigm' it will be very difficult to evolve representational (i.e. non-abstract) images. There exist a number of alternatives to raster paradigm EvoArt systems, and we discuss several of these approaches in the next section on related work.
Our research questions are:

1. Is SVG a suitable representation for EvoArt? I.e. is it possible to implement all representation dependent components of an EvoArt system (mutation, crossover and initialisation) for SVG?

2. Are the resulting images different from the ones typically produced by EvoArt systems that use expression based representation?

3. Can we evolve *representational*, non-abstract images using SVG?

4. Can we evolve surprising new images using existing images?

This paper is organised as follows; in section 2 we perform a short literature overview of the use of several types of representation in EvoArt. In section 3 we describe Scalable Vector Graphics. In Section 4 we describe our genetic operators and experiments with evolving abstract images, and in Section 5 we describe our experiments and genetic operators in evolving representational images. We give conclusions and directions for future research in section 6.

2 Representation in Evolutionary art

Evolutionary art is a research field where methods from Evolutionary Computation are used to create works of art. Good overviews of the field are Romero & Machado (Romero & Machado, 2007) and Bentley & Corne (Bentley & Corne, 2001). Some EvoArt systems use Interactive Evolutionary Computation (IEC) or supervised fitness assignment (Sims, 1991; Rooke, 2001), and in recent years there has been increased activity in investigating unsupervised fitness assignment (Baluja *et al.* , 1994; den Heijer & Eiben, 2010a,b; Machado & Cardoso, 1998; Ross *et al.* , 2006; Unemi, 1999).
A number of different representations have been investigated for use in evolutionary art. We will briefly describe symbolic expressions, shape grammars, cellular automata and L-systems, vector graphics and representations that use an image as

a source.

'Raster paradigm' with Symbolic Expressions The most widespread representation within EvoArt is the symbolic expression employing the 'raster paradigm' (den Heijer & Eiben, 2010a,b; Greenfield, 2000; Machado & Cardoso, 2002; Rooke, 2001; Sims, 1991). The symbolic expression/ 'raster paradigm', pioneered by Karl Sims in 1991 (Sims, 1991) works roughly as follows; each genome is a symbolic expression (i.e. a Lisp function tree) that consists of functions from a predefined functions set and terminals from a predefined terminal set. Terminals can consists of variables like x and y (that correspond to the coordinates in the image grid) or constants. The phenotype is an image of size (w, h), and the expression of genotype into the phenotype is done using the following algorithm (which constitutes the 'core' of the raster paradigm);

for $x = 0$ **to** w **do**
  for $y = 0$ **to** h **do**
    $v \leftarrow calculate(x, y, tree)$
    $image[x, y] \leftarrow v$
  end for
end for
return image;

There are a number of variations on this theme. Some authors normalise the values of x and y between 0 and 1 or between -1 and 1 (Greenfield, 2000), some authors map the value v onto a colour index table (den Heijer & Eiben, 2010a,b; Greenfield, 2000) but the main idea is the same. There are a number of publications on the use of expression trees that evolve representational content (and thus do not follow the 'raster paradigm'); Machado et al (Machado *et al.* , 2012) evolve pictures of faces, whereby a face detection algorithm is used as a fitness function. There are several expression based representations that use NPR functions, and they are described in the paragraph labelled 'Using images as a source'.

Shape Grammars Although symbolic expressions have been a very popular form of representation in evolutionary art, other forms of representation have been investigated. The most notable other form is the shape grammar. A shape grammar is a formal description of a design and has been pioneered by Stiny and Gips in 1972 (G. Stiny, 1972). Shape grammars are especially useful in the context of design and architecture, since design heuristics can be coded into the grammar. Examples of the use of shape grammars in EvoArt and design are Schnier et al (Schnier & Gero, 1996) and O'Neill et al (O'Neill *et al.* , 2009). Machado et al (Machado *et al.* , 2010) describe the use of shape grammars using the Context Free language to evolve multiple artworks in a similar style. A very particular study by Eiben et al (Eiben, 2008; Eiben *et al.* , 2001) attempted to mimic artwork of M.C. Escher, using a representation based on the mathematical system behind his tilings . This system was described by Escher and Doris Schattschneider (Schattschneider & Escher, 2004) and transformed into an evolvable genetic representation that produced images with surprising similarity to the original. Lewis (Lewis, 2000) evolved cartoon faces using a template for a generic face, and a number of parameters for size, location etc. of various parts of the face.

Cellular Automata and L-Systems Other representations used in EvoArt are cellular automata, several types of fractals and L-systems. Ashlock et al (Ashlock & Tsang, 2009) used EC to evolve aesthetically pleasing images using

Cellular Automata. In related work, Ashlock et al (Ashlock, 2006) investigated the evolution of interesting appearing Julia Sets. Jerry Ventrella (Ventrella, 2008) has evolved 'tweaked' Mandelbrot functions to make the Mandelbrot figures look like a target image.

Vector Graphics Stephen Bergen and Brian Ross (Bergen, 2009; Bergen & Ross, 2012) have explored the use of vector graphics in their JNetic system. Their genotype representation consists of integer based chromosomes, where indices in the chromosome refer to a vector graphics primitive (e.g. a circle, a square), the colour (coded in r,g,b) and the (x,y) position of the vector graphics primitive. Baker et al (Baker & Seltzer, 1994) describe an approach that uses a custom and simple vector representation to evolve line drawings of faces . The vector drawing primitives strongly resemble the SVG 'path' element, in which elementary lines and curves can be drawn on the canvas. Furthermore, they added a number of simple symmetry markers, to duplicate elements to the opposite half of the canvas. Their approach was very much biased towards the evolution of line drawings of faces,

Using images as a source; Filters and NPR Whereas the previous approaches create images 'from scratch', some researchers have investigated the possibilities of manipulating existing images, whereby the manipulating function was subject to evolutionary computation. Collomosse (Collomosse, 2007) describes an approach that using non-photorealistic rendering or NPR (Gooch & Gooch, 2001) to produce synthetic oil paintings from images; the author uses a genetic algorithm to find suitable values for his NPR system. Neufeld et al (Neufeld *et al.* , 2007) describe the evolution of a NPR system using genetic programming, whereby the authors use a number of image filter primitives. DiPaola et al (DiPaola & Gabora, 2009) evolve renderings of portraits using a target image (in the paper they use a portrait of Charles Darwin), and Barile et al (Barile *et al.* , 2008) evolve novel NPR filters using genetic programming. Another recent example of combing NPR with GP expression trees is the work by Baniasadi et al (Baniasadi & Ross, 2013).

From this short overview we see that a few EvoArt systems use a representation that re-use existing images to evolve new images. For EvoArt systems that evolve images 'from scratch' it is very difficult, if not impossible, to evolve non-abstract art. The work in this paper is similar to the work by Bergen & Ross (Bergen & Ross, 2012), the Non-photo Realistic (NPR) work by Neufeld et al (Neufeld *et al.* , 2007), and the evolution of faces by Baker et al (Baker & Seltzer, 1994). The first series of experiments that we describe in Section 4 uses only SVG graphic primitives, and is similar to the work by Bergen & Ross (Bergen & Ross, 2012), although our goal is to evolve abstract images, and the goal in (Bergen & Ross, 2012) is to follow an NPR approach. The goal of the work by Baker et al (Baker & Seltzer, 1994) is to demonstrate the use of IEC in the search through face space. They constructed one vector image of a face manually, and used this as a starting point for their experiments. We initialise our populations using SVG images that we extracted from existing images. We also use colour in our images, whereas Baker et al use black and white line drawings.

3 Scalable Vector Graphics

Vector graphics operates on primitives like lines, points, curves and polygons and is complementary to raster graphics that operate on pixels. SVG is a graphics format developed and maintained by the World Wide Web Consortium (W3C) (W3C, 2005) and is an XML format for vector graphics. An important advantage of vector graphics over raster graphics is the possibility to scale an image without loss of image quality. Another important advantage of the use of SVG as a representation for EvoArt is the potential interoperability with the artist/ designer; an artist or designer can start with an SVG document in his or her vector graphics tool (like Inkscape or Adobe Illustrator) and use the output of his or her work as input for the EvoArt system. Next, the output of the EvoArt system can be used as input for the artist or designer. Both EvoArt system and designer tools speak the same language: SVG.

3.1 Basic layout of an SVG document

SVG is an implementation of XML and should comply to all basic XML rules; documents consists of elements and elements can have child elements. Furthermore, an SVG document must be well-formed; i.e. it should comply to all XML syntax rules. There are a number of specific rules to which SVG documents must comply and we will briefly describe the most important ones. First, the root element (the top level element) must be 'svg'. The SVG specification allows to nest 'svg' elements into lower level elements as well, but in our initial implementation we chose not to implement that (but we might do so in the future). Next, there can be zero or more definitions in a 'defs' element (SVG does not enforce a document to begin with a 'defs' element, but we do so in our implementation for reasons of simplicity). Definitions are like declarations of variables. Here we can clearly see a big difference with the symbolic expression representation; symbolic expressions are stateless, they have no state variables (only local variables in leaf nodes). A 'defs' element is merely a container of other elements. Elements that can be declared as 'variables' in a 'defs' container are

- `filter` - a filter in SVG alters the looks of a certain area of an image by applying an image filter effect on that particular area.
- `linearGradient` and `radialGradient` - gradients are transitions of colour over a certain area. SVG supports linear gradients (linear transition from one point to another) and radial gradients (colour transitions are circular/ ring shaped). Gradient definitions may refer to pre-defined filters.
- `style` - a style definition; a container for one or more css declarations. A declaration can define the foreground colour, the background colour, the stroke width, the stroke colour etc. In short, the css class determines the look and feel of a shape element. Shape elements refer directly to the css class definition (and not to the style container). A css class definition may refer to a linear or radial gradient definition.
- `mask` - a mask is an outline whereby everything on the inside of the mask is shown and everything on the outside is 'masked'. With a mask you can create

a 'hole' of a a certain shape. A mask is a container element; it contains other elements that define the shape of the mask.

- `pattern` - a pattern is container element that contains other elementary shapes (like 'rect' and 'ellipse') that are repeated such that they create a pattern (much like a wallpaper pattern).

Next to the 'defs' element, an SVG document can have a number of shape elements. In our implementation we have implemented the following shape elements:

- `rect` - a rectangle shape
- `ellipse` and `circle` - an ellipse; it has a centre coordinate, an x and a y radius. If the x and the y radius are equal, the result is a circle. The `circle` element is similar, but only has one radius.
- `path` - path is the most versatile element. A path defines a number of basic operations that are similar to turtle graphics; operations include move to, a number of basic line commands, and a number of Bézier curve commands. The path element is used extensively in our experiments with evolving abstract images (experiment 3 and 4, Section 4.6) and in evolving representational images (Section 5).
- `polyline` - a polyline is a collection of connected lines. A polyline does not fill an area (like a polygon does).
- `polygon` - a polygon is also a collection of connected lines, whereby the first and last point of the polygon are also (automatically) connected. The surrounding area is filled with the fill colour (if any) of the polygon.
- `group` - a group is a container element that holds one or more other elements (that can also be a 'group'). Groups are a simple way to implement complex constructs from a number of simple elements. A group can therefore occur both in the 'defs' part and in the 'shapes' part of an SVG document.

In the declaration of a shape element there can be references to declarations in the aforementioned 'defs' section. Elements can specify a filter, a css class, a mask, a pattern, a linear gradient or a radial gradient. For example, a rect element can have a reference to a CSS class in the 'defs' part, this css class may have a specification of the 'fill' property (that specifies how an element should be filled) that refers to a radial gradient element elsewhere in the 'defs' element, and this radial gradient element may have a reference to a filter. As we will see later, the interconnectedness of both 'defs' and shape elements with each other requires an elaborate bookkeeping process with the mutation and crossover operator; SVG parsers are usually very strict, and creating offspring that contains broken links (i.e. pointing to a filter element that no longer exists in the new offspring) will result in a SVG rendering error. Figure 2 shows an outline of an SVG as used in our system, and Table 1 shows a number of simple SVG example documents and their rendered images. The SVG specification is vast and complex, and we have not covered every aspect of it, nor have we implemented the entire SVG specification. Next to the elements described above, we have implemented 'use' and 'image', but we have not used them in the

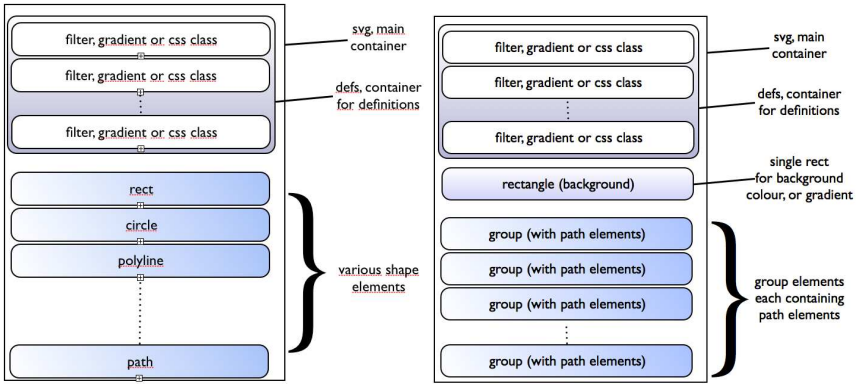


Figure 2: Two schematic outlines an SVG document in our system; the left outline is contains several SVG shapes an is used in Experiment 1 and 2 (Section 4.5), the right outline contains only 'path' elements, and is used in Experiment 3 and 4 (Section 4.6) and in Experiment 5 (representational images, Section 5).

experiments in this paper. In our current implementation we have skipped 'text' (rendering text labels), 'metadata' (specifying RDF metadata in an SVG element), javascript (mainly for animating svg elements and user interaction) and a number of SVG filters.

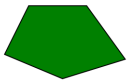
	
`<circle cx="100" cy="50" r="40" stroke="black" stroke-width="2" fill="blue" />`	`<rect x="20" y="20" width="50" height="25" fill="red" />`
	
`<polygon points="220,100, 300,210,170,250,50,200, 100,100" style="fill:green; stroke:black; stroke-width:2" />`	`<polyline points="50,50, 200,50,200,200,100,100, 50,200" style="fill:white; stroke:violet; stroke-width:4" />`

Table 1 Four simple examples of SVG code and their images

4 Evolving abstract images

As stated previously, the most used representation in EvoArt is the symbolic expression employing the 'raster paradigm'. The advantage of using symbolic expressions

SVG Configuration		
Number of	Minimum	Maximum
SVG shape elements	3	6
Linear gradients	1	4
Radial gradients	1	4
Masks	1	1
Patterns	0	1
Filters	4	5
CSS classes	3	4

Table 2 SVG initialisation parameters for the declaration part of the SVG document (defined in the 'defs' element of the SVG document).

is twofold; first, when using symbolic expressions, it is easy to create valid new trees from existing ones, since the trees are type-safe (i.e. the type of each sub expression tree is the same, so you can select any (sub)tree node as input for any other tree node). Second, symbolic expressions are stateless; they have no state variables (only local variables in leaf nodes), and this makes crossover and mutation relatively easy to implement. SVG does not have these advantages, so implementing genetic operators for SVG is more complex. In this section we will describe the genetic operators initialisation, crossover and mutation. All operators are SVG specific and all operators produce results that conform to the SVG standard. All declaration elements (the elements in the 'defs' element in the svg document) and all shape elements are potentially subject to mutation or crossover.

4.1 Initialisation

Initialisation uses a number of parameters to create new individuals. For example, there is a parameter 'number of svg shape elements' with a minimum of 3 and a maximum of 6. This means that between 3 and 6 shape elements are created. Table 2 has all the initialisation parameters and their minimum and maximum values. Initialisation also uses a weight distribution for shape elements; this way we can perform different experiments with different distributions of shape elements (e.g. we can do experiments with only 'path' elements). The initialisation procedure that we use for evolving representational images is different, and we will describe it in Section 5.1.

4.2 Mutation

The mutation operator for the experiments with abstract images processes an SVG document top-down, and (depending on the mutation probability) either copies or mutates each child element of the parent. There is a specific mutation operator for each type of SVG element. For instance, if the element is an ellipse, then the ellipse mutation operator is called, and the specific attributes of the ellipse are potentially subject to mutation (the mutation can change the coordinates of the ellipse, and/ or the horizontal/ vertical radius). There are a number of heuristics; each numeric attribute (x, y coordinate, radius etc.) is increased or decreased between 0 and 10% of the original value. For the 'defs' element, mutation is similar; each child element

in the 'defs' is potentially subject to mutation; a 'filter' element might change from a linear gradient filter to radial gradient filter, or the specific parameters of the filter (like colours, offsets) might be mutated. Css elements that are defined in the 'defs' element can also be mutated; attributes that can be mutated are colour, stroke etc.. The mutation operator that we use in the experiments with representational images is different, and we will describe it in Section 5.2. The mutation operator does not add new elements, nor does it remove existing ones. Our mutation is presented as pseudo-code in Algorithm 1. In Figure 3 we present a number of visual examples of our mutation operator.

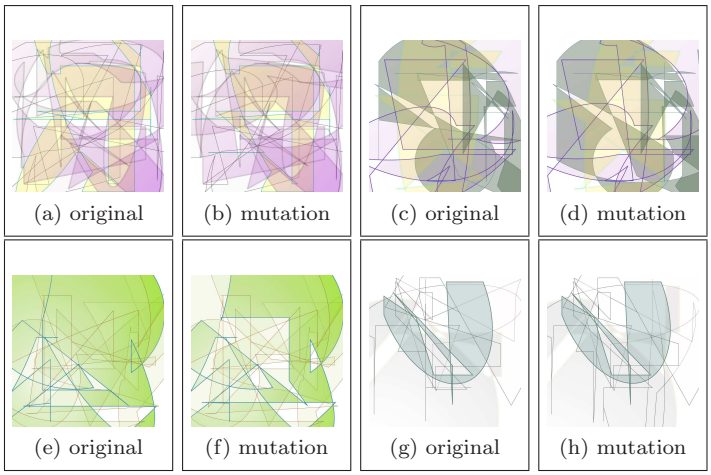


Figure 3: Four examples of mutation

4.3 One-Point Crossover

For the crossover operation in the experiments with abstract images, we implemented a one-point crossover operator specific for SVG. The crossover works on two parent svg documents and creates one child per operation. Each parent consists of a defs part and a shapes part. The 'defs' part contains only declarations of filters, css classes. For sake of simplicity, we define the shapes part as everything that comes after the defs part (and contains only shape elements). Crossover is implemented as follows: first we copy the defs part of one of the parents to the child. Next, we concatenate the first half of the shapes part of one parent with the second part of the shapes part of the other parent. Since shape elements have references to definitions that reside in the 'defs' element, the new child will have references in shape elements that do not exist in the child (since we only copied the 'defs' element of one parent, but we have shape elements of both parents). An SVG interpreter will not render such a document (with references to non-existing elements), so we have to fix the broken references; we traverse the shape elements, and check whether the references to a filter, css class, mask etc. are available. If not, the reference is replaced with an existing (new) reference from the child document. An example: suppose we have a father document that has a 'rect' element (a shape element) that refers to a 'cssClass' element (a 'defs' element) with id '123'.

Algorithm 1 Mutation

```
newChildren ← new List()
for all childElement in SVG do
  r ← random()
  if (r < mutationProbality) then
    child ← mutate(childElement)
  else
    child ← childElement
  end if
  newChildren.add(child)
end for
SVG.setChildren(newChildren)
return SVG
```

Algorithm 2 One-Point Crossover

```
p1 ← parents.get(0)
p2 ← parents.get(1)
r ← random()
if (r < 0.5) then
  defs ← p1.getDefs()
else
  defs ← p2.getDefs()
end if
shapes ← new List()
shapes.add(getFirstHalf(p1))
shapes.add(getSecondHalf(p2))
shapes ← repair(shapes, defs)
return new SVG(defs, shapes)
```

Now suppose we do a crossover and this 'rect' element in the child class is 'cut off' from this 'cssClass' with id '123' (because this cssClass definition is not copied to the child document), then we have to re-assign the 'cssClass' reference in the 'rect' element from '123' to '456' (or any other id that does exist in the 'defs' of the child document). This means that the 'rect' element will be rendered differently in the child element. Our crossover is presented as pseudo-code in Algorithm 2. In Figure 4 we present a number of visual examples of our crossover operator. The crossover operator that we use in the experiments with representational images is different, and we will describe it in Section 5.3.

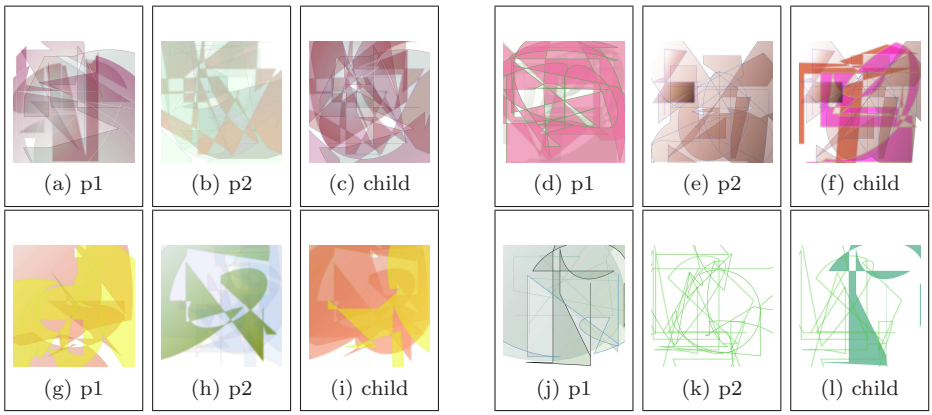


Figure 4: Four examples of crossovers; from left to right, the two parents (p1 and p2) and the resulting child

Aesthetic Measure	Various SVG Elements	Only 'path' element
Ross, Ralph & Zong	Experiment 1	Experiment 3
GCF	Experiment 2	Experiment 4
Custom pop-art		Experiment 5

Table 3 Overview of experiments

4.4 Experiments with evolving abstract images

In order to explore the potential of SVG as a representation for EvoArt we conducted four experiments; two experiments with a variety of SVG elements (polygons, polylines, circles and paths) and two experiments with only the 'path' element. Two of the experiments were performed with the Ross, Ralph & Zong aesthetic measure (Ross *et al.* , 2006) and two experiments were performed with the Global Contrast Factor aesthetic measure (Matkovic *et al.* , 2005); see Table 3 for more details. These aesthetic measures were used as fitness functions in an unsupervised EvoArt system; no human evaluation/ interactive evolution was involved.

The Ross, Ralph & Zong aesthetic measure is based on the observation that many fine art painting exhibit functions over colour gradients that conform to a normal or bell curve distribution. The authors suggest that works of art should have a reasonable amount of changes in colour, but that the changes in colour should reflect a normal distribution. The computation takes several steps and we refer to (Ross *et al.* , 2006) for details. Previous experiments with the Ross, Ralph & Zong aesthetic measure as a fitness function in an unsupervised EvoArt system have shown that the use of this measure often leads to images with rich colouring and smooth colour transitions (den Heijer & Eiben, 2010b). The global contrast factor computes contrast (difference in luminance or brightness) at various resolutions. Images that have little or few differences in luminance have low contrast and are considered 'boring', and thus have a low aesthetic value. Contrast is computed by calculating the (average) difference in luminance between two neighbouring regions. The contrast is calculated for several resolutions (2, 4, 8, 16, 25, 50, 100 and 200) and the average contrast is summed as

$$M_{gcf}(I) = \sum_{k=1}^{9} w_k \cdot contrast(n, p_k, r_k) \tag{1}$$

where r_k refers to the resolution of the region, w_k refers to the weight of the contrast of the region (the weight of the contrast differs per resolution) and p_k is a power factor. Both w and p were optimised using several experiments (Matkovic *et al.* , 2005). In our implementation we used all the settings from Matkovic et al (Matkovic *et al.* , 2005), and we refer to that paper for more details. In previous experiments with the global contrast factor as a fitness function it was shown that images that were evolved using GCF as the fitness function had a lot of alternating black and white areas (hence, a lot of contrast) (den Heijer & Eiben, 2010a)

Furthermore, we performed 10 runs per configuration, saved the images that had the highest fitness score, and selected a portfolio of 24 images from the 100 images. The portfolio for each experiment is shown in the next subsections.

Symbolic parameters	
Representation	Scalable Vector Graphics (SVG)
Initialisation	Custom SVG Initialisation (see 4.1)
Survivor selection	Tournament, Elitist (best 1)
Parent Selection	Tournament
Mutation	Custom SVG mutation (see 4.2)
Recombination	Two parent single point crossover (see 4.3)
Fitness function	Ralph & Ross or Global Contrast Factor
Numeric parameters	
Population size	200
Generations	10
Tournament size	2
Crossover probability	0.75
Mutation probability	0.25

Table 4 Evolutionary parameters of our EvoArt system used in our experiments

4.5 Experiment 1 & 2: multiple SVG elements

First we conducted two experiments with a variety of SVG elements. We initialised the SVG elements with circle, polygon, polyline and path elements (all with an initialisation probability of 0.25). The 'defs' part of the documents were initialised according to the specifications in Table 2.

4.5.1 Experiment 1: Ross, Ralph & Zong

In the first experiment we initialised the population with documents containing circle, poyline, polygon and path elements. We used the Ross, Ralph & Zong aesthetic measure as the fitness function. As said before, we did 10 runs using this setup and gathered the 10 fittest images of each run, and handpicked 24 images; these images are shown in Figure 5; Almost all images have rich and variable colouring which is

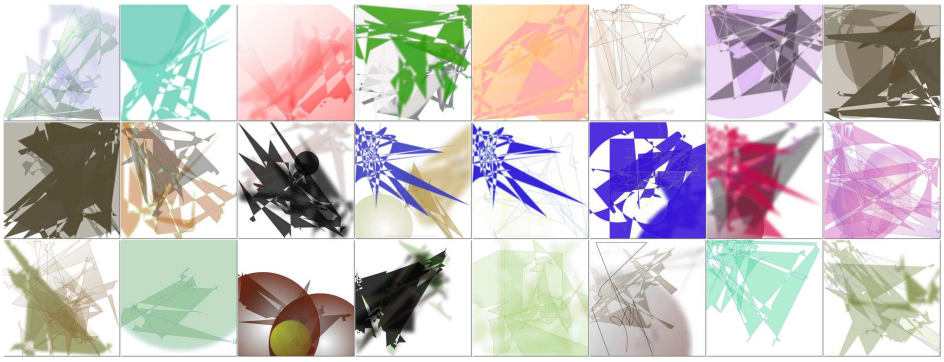


Figure 5: Portfolio of images gathered from ten runs with Ralph & Ross with various SVG elements (Experiment 1)

consistent with earlier experiments with the Ross, Ralph & Zong aesthetic measure (den Heijer & Eiben, 2010b) The polygon elements seem to dominate the look and feel of most images, and they make many images interesting, but they do tend to give them a slight 'computer art' flavour (although different from the images evolved using symbolic expressions employing the 'raster paradigm', see Figure 1).

4.5.2 Experiment 2: GCF

The second experiment uses the Global Contrast Factor as the fitness function, but is otherwise identical to Experiment 1. The images are shown in Figure 6. The

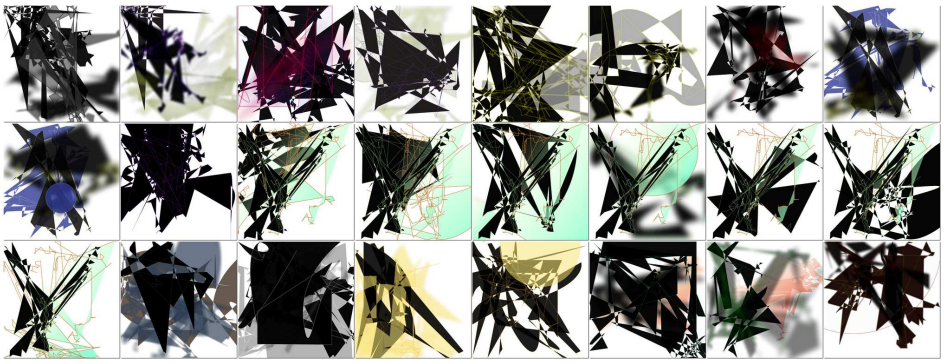


Figure 6: Portfolio of images gathered from ten runs with GCF with various SVG elements (Experiment 2)

images evolved using the GCF show a lot of contrast, and this is similar to earlier findings (den Heijer & Eiben, 2010a). The high level of contrast in the images have a very powerful effect, but it does give the images a certain 'harshness' that is not present in the images from Experiment 1 (with the Ross, Ralph & Zong aesthetic measure). Several images are reminiscent of 1960s computer art imagery by Michael Noll (Noll, 1967).

4.6 Experiments with the 'path' element

In the third and fourth experiment we initialised the population with genomes with only the 'path' element. The 'path' element is the most versatile SVG element; it contains a number of operations that closely resemble turtle-graphics (see Section 3 on a brief explanation of the 'path' element and see the appendix for an SVG document with many path elements). We initialised each document with 3 to 6 (see Table 2) 'path' elements, whereby each path element had between 10 and 80 path operations.

4.6.1 Experiment 3: Ross, Ralph & Zong

In the third experiment we evolved SVG document with just 'path' elements using Ross, Ralph & Zong as the fitness function. We present the images of this experiment in Figure 7. The first thing that is striking is the variety of the images; it is

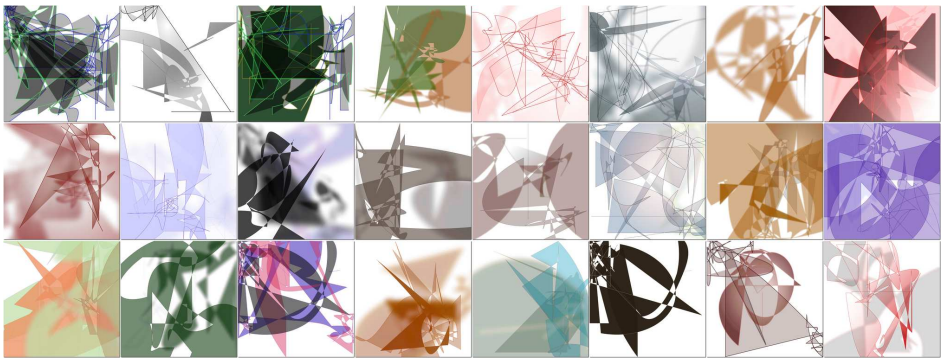


Figure 7: Portfolio of images gathered from ten runs with Ralph & Ross with the 'path' element (Experiment 3)

interesting to see that the 'path' element alone is versatile enough to create a wide variety of images, sometimes arguably more interesting than the circles, polygons, polylines and paths from Experiments 1 and 2. The addition of the curve operation in the 'path' element seems to have additional value over the standard polygons and polylines. Note that we initialise the points and operations of all 'path' elements randomly, there is no use of domain knowledge (e.g. from art theory). This randomness sometimes give the images a certain artificial flavour. In Experiment 5 we use path elements that were initialised using vector images that were 'extracted' from existing raster images, and this reduces the artificial flavour of the resulting images. The images from Experiment 3 are also varied in colour and this is consistent with Experiment 1 and previous work (den Heijer & Eiben, 2010b).

4.6.2 Experiment 4: GCF

The last experiment is identical to Experiment 3, except for the use of the Global Contrast Factor as the fitness function. We present the images of Experiment 4 in Figure 8. The images from Experiment 4 are also varied in shape (like Experiment 3) but again show a tendency towards black and white, which is similar to Experiment 2 and previous work (den Heijer & Eiben, 2010a).

5 Evolving representational images

In our second series of experiments, we clearly wanted to increase the potential visual output range of our evolutionary art system, and we wanted to evolve representational images. The genetic operators from the previous sections were not sufficient for this task, so we had to design and implement a new initialisation, mutation and crossover. In this section we will first describe these three operators. Next, we will describe a simple aesthetic measure for pop-art in Section 5.4. We will present our experimental setup and the results of the experiments in Section 5.5.

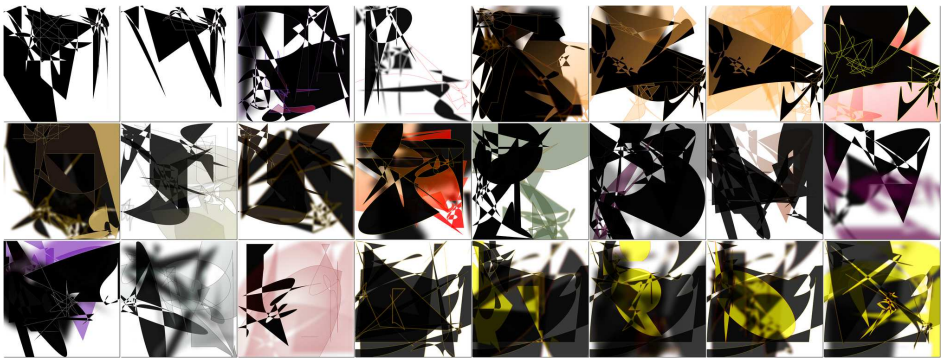


Figure 8: Portfolio of images gathered from ten runs with GCF with the 'path' element (Experiment 4)

5.1 Initialisation

In the previous section we initialised SVG genetic programs randomly with path elements and geometric SVG primitives. This approach produced some interesting images, although most images had an artificial, abstract flavour. In this section we intend to depart from evolving abstract art, and decided to use existing images as a starting point. Our initialisation process is represented in Figure 10.
In previous work (den Heijer & Eiben, 2012) we evolved SVG images using a collection of personal photographs of the first author. For this paper we created another, more diverse image set, consisting of 80 images from the RGBStock website (http://www.rgbstock.com, n.d.). We searched for rights free images that contained a single topic, preferably without any background clutter (white background). This way, it would be easier to combine numerous images into one new image. This process is analogous to using sample libraries in electronic music, where prepared audio samples are combined to create new work (audio sample libraries mostly contain samples from a single instrument, often played in a single key).
From the photographs (raster images) we create vector images. We used the publicly available program 'potrace' (Available at `http://potrace.sourceforge.net/`)(Selinger, 2003) to convert the raster images to our initial SVG sources. The 'potrace' program extracts the contours of a raster image and creates path elements with either lines or curves. One important aspect of this approach is that all colour is removed when extracting the contours, thus the resulting SVG images (that come out of 'potrace') are in black and white. Next to a collection of images, we also created a collection of colour schemes. A colour scheme is a list of colours that (ideally) combine well. We randomly generated 250 colour schemes with 2 to 5 colours per colour scheme.
To summarise, the steps of initialisation are;

1. randomly choose one colour scheme
2. sample 1 to 3 images from the aforementioned image collection, and create one group (`g` element) for each sampled image (each containing multiple path elements)

SVG Configuration		
Parameter	Minimum	Maximum
Number of svg sources	3	6
Number of linear gradients	1	2
Number of radial gradients	1	2
Number of filters	2	4
Number of CSS classes	2	4

Table 5 SVG initialisation parameters for the second series of experiments (representational images).

3. create one rectangle (`rect` element) that will act as the background; SVG does not support setting the background colour of the canvas itself.

4. create a random `defs` part using the sampled colour scheme; the 'defs' element may contain a css part, one or more gradients, and one or more filters.

5. assign filter and css class to the background rectangle and all path elements.

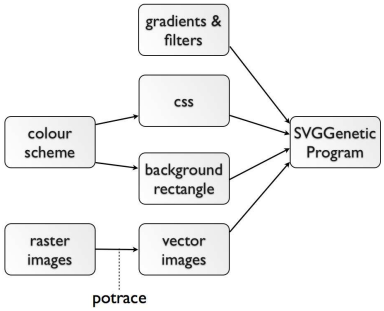
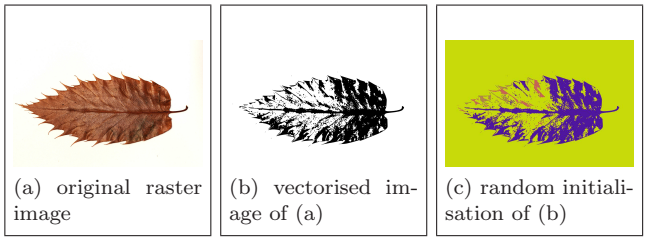


Figure 9: The outline of our SVG genotype initialisation process



(a) original raster image

(b) vectorised image of (a)

(c) random initialisation of (b)

Figure 10: The initialisation process in a nutshell; we start with a photo or raster image in (a), potrace converts this image into an initial SVG vector image (b), and our initialisation process adds one or more images (in this example only one) to the canvas, and adds and applies filters, gradients and css classes.

5.2 Mutation

We implemented several mutation operators that fall in two categories; macro and micro level mutation. The macro level mutation affects the entire composition and the micro level mutation operator operate at a single 'group' (a collection of 'path' elements) in the composition. The probability of macro and micro level mutation is 0.5. If macro-level mutation is performed, the mutation is done once on the entire program. If micro-level mutation is 'selected', a uniform randomly selected micro level mutation operator is performed for each group of path elements in the SVG document.

Algorithm 3 Our reproduction; we perform either crossover or mutation (not both). Within mutation, we do either macro-level mutation or micro-level mutation (not both)

```
r1 = random()
if (r1 < crossoverProbability) then
  doCrossover();
else
  d2 = random();
  if (r2 < 0.5) then
    doMacroMutation();
  else
    doMicroMutation()
  end if
end if
```

5.2.1 Macro level mutation

We have implemented the following macro level operators;

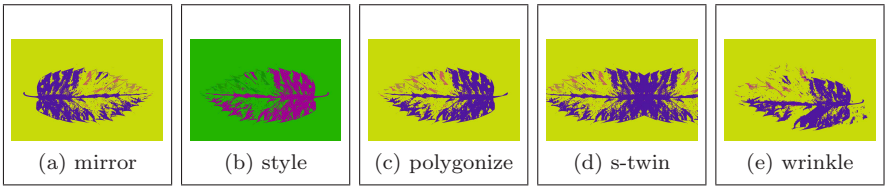
- `thicken` - this operator samples another image from the image collection and adds it at a random point in the composition
- `thin` - opposite of `thicken`; this operator removes a random chosen image from the composition (unless there is only one image on the canvas left; in that case the `thin` operator does nothing).
- `unclutter` - moves the images on the canvas in such a way that they do not overlap
- `updatestyle` - does a mutation on the css class definition of the `defs` part of the SVG document (affects the rendering of all elements that refer to a css class)
- `updatefilter` - does a mutation on the filter definitions of the `defs` part of the SVG document (affects the rendering of all elements that refer to a filter)

5.2.2 Micro level mutation

We implemented 11 micro level mutation operators. All operate on a group of path elements.

- `hideall` the 'hide all' mutation processes all the path elements in a group, and sets the attribute 'visibility' to 'hidden'. The effect is that the path will be present in the SVG document (the genotype) but will not be expressed in the image (the phenotype).
- `hidemore` the 'hidemore' mutation is similar to 'hideall' but the probability of a path to become invisible is 0.25.
- `mirror` the 'mirror' mutation creates a mirrored version (around the horizontal or vertical axis) of all the path elements in a group.
- `polygonize` replaces all curve operations (the operations with operator 'c', 't', 'a' and 'q') with a line operator ('l'). In many cases this mutation gives the images a simplified or 'compressed' look and feel, but if start and end point are close to each other, the effect is barely noticeable.
- `replace` the 'replace' operator resembles the subtree mutation operator in standard genetic programming; it replaces the entire group with a new initialized group (sampled from the image collection).
- `siamesetwin` the 'siamesetwin' operator is a complex mutation operator. It creates a horizontal or vertical mirror image of a group, moves the mirror image to the left (or up) and merges the result in the original group. This mutation operator creates images with symmetry, and sometimes the images resemble Rorschach ink blob tests (Figure 11d).
- `showall` 'show all' is the inverse of 'hide all'; it updates all the path elements in a group, and removes the 'visibility' attribute (which is equivalent to setting the visibility element to 'visible').
- `showmore` is similar to 'showall', but the probability of a path to become visible (if it was invisible) is 0.25.
- `updatefilter` this mutation alters the filter identifier of each path (if any) with a probability of 0.25.
- `updatestyle` this mutation alters the CSS class identifier of each path (if any) with a probability of 0.25.
- `wrinkle` the 'wrinkle' operator adapts all the parameters in all path elements in a group and adds or subtracts between 0 and 5% of the original value. The effects are different for the different path operators; for the SVG path 'move' operator ('M'), it may result in a displaced path element (sometimes it leads to an eye that appears somewhere on a cheek, somewhat like Picasso), and for the different curve operators it results in different curves, resulting in 'distorted' paths (Figure 11e). he effect on portrait images is sometimes funny, and sometimes unpleasant; some images are reminiscent of paintings by Francis Bacon.

Figure 11 shows five different mutations on the image of Figure 10c.

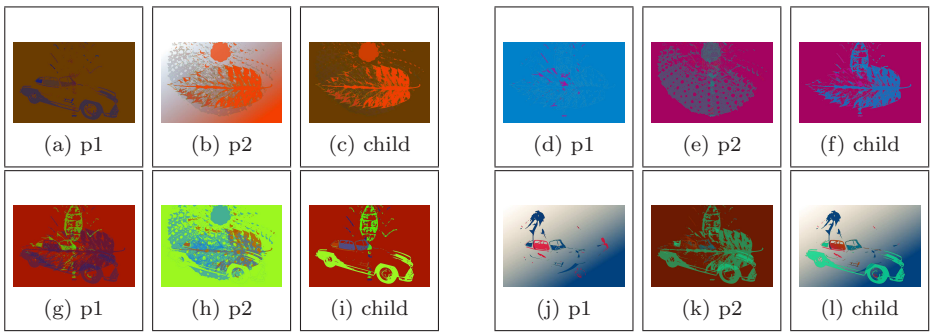


(a) mirror (b) style (c) polygonize (d) s-twin (e) wrinkle

Figure 11: Examples of 5 possible mutations of the original of Figure 10c; (a) mirror, (b) mutation of style, (c) polygonize mutation, (d) siamese twin mutation, (e) wrinkle mutation.

5.3 Uniform Crossover

We implemented a uniform crossover operator that creates a new SVG genotype from two parent SVG genotypes. Recall that an SVG document consists of two parts; the definitions or declarations, that reside in the `defs` element and the shapes, which are in the rest of the document (they are not contained in a separate container element). The crossover operator consists of 3 steps: first, we select the background rectangle randomly from one of the parents. Next, we select the colour scheme of one of the parents, and assign it to the new child (we do not perform crossover on the colour scheme itself). Next, we iterate over all elements of the defs part and the non-defs part, and randomly select an element from one of the parents. We present four examples in Figure 12.



(a) p1 (b) p2 (c) child (d) p1 (e) p2 (f) child

(g) p1 (h) p2 (i) child (j) p1 (k) p2 (l) child

Figure 12: Four examples of crossovers; from left to right, the two parents (p1 and p2) and the resulting child

5.4 A simple aesthetic measure for pop art

In previous work we have applied a number of aesthetic measures in EvoArt, and in our initial experiments with SVG we tried a number of them. Most of these aesthetic measures that we tried on SVG (most notably Benford Law (del Acebo & Sbert, 2005) and Ross, Ralph & Zong (Ross *et al.* , 2006)) assigned low scores to the

Algorithm 4 Uniform Crossover

```
p1 ← parents.get(0)
p2 ← parents.get(1)
defs ← newDefs()
shapes ← newList()
r ← random()
if (r < 0.5) then
  defs.setBackground(p1.getDefs().getBackground())
else
  defs.setBackground(p2.getDefs().getBackground())
end if
d ← (defs1.size + defs2.size)/2
for (i ← 0; i < d; i++) do
  r ← random()
  if (r < 0.5) then
    defs.add(defs1.get(i))
  else
    defs.add(defs2.get(i))
  end if
end for
s ← (p1.getShapes().size + p2.getShapes().size)/2
for (i ← 0; i < s; i++) do
  r ← random()
  if (r < 0.5) then
    shapes.add(p1.getShapes().get(i))
  else
    shapes.add(p2.getShapes().get(i))
  end if
end for
shapes ← repair(shapes, defs)
return newSVG(defs, shapes)
```

evolved images, including several images that we liked ourselves. We decided to create a simple aesthetic measure that favours contrast in hue, as is often seen in screen printing and pop art (Perry, 2011). Our aesthetic measure is a combination of two ideas; the first idea comes from the Global Contrast Factor or GCF (Matkovic *et al.* , 2005). This measure samples the contrast in brightness at various resolutions of the image and computes the amount of contrast. The other idea comes from colour harmony theory (Birren, 1987); there are several principles that suggest that particular combinations of colour are considered pleasurable, and one principle of colour harmony is the principle of opposing colours. This states that a combination of two colours that are opposed to each other on the colour wheel is preferable to other combinations. Although there are other principles on the harmony of colour, we will focus on the difference in hue. In a nutshell, our hue difference aesthetic measure works as follows: select two regions of the image (A and B), calculate the average hue for both regions, and calculate the difference be-

tween the average hues. Repeat this step for all regions of the image, for a number of different resolutions, and calculate the average hue difference.

$$M_{popart}(I) = \sum_{k=1}^{9} w_k \cdot hue\_difference(n, p_k, r_k) \quad (2)$$

where the $hue\_difference$ between two regions A and B is calculated as the difference in the average hue of the pixels of region A and B. Weight w_k, power factor p and resolution r are calculated in the same way as the global contrast factor (see Section 4.4 and Matkovic et al (Matkovic *et al.* , 2005) for more details).

5.5 Experiment 5: evolving representational images

We performed an experiment with our new initialisation, crossover, mutation and new ad-hoc aesthetic measure for pop-art for the aesthetic evaluation (there is no human in the loop). In the next subsections we will present the parameters of our EvoArt system, and present the resulting images.

Symbolic parameters	
Representation	Scalable Vector Graphics (SVG)
Initialisation	Custom SVG Initialisation (see 5.1)
Survivor selection	Tournament, Elitist (best 1)
Parent Selection	Tournament
Mutation	Custom SVG mutation (see 5.2)
Recombination	Two parent uniform crossover (see 5.3)
Fitness function	Colour contrast (hue)
Numeric parameters	
Population size	100
Generations	10
Tournament size	3
Crossover probability	0.5
Mutation probability	0.5 (within a mutation 'step', the probability for micro vs macro mutation is 0.5)

Table 6 Parameters of our EvoArt system used in experiment 5

Note that there are a few differences between the evolutionary parameters of the first series (abstract images, Section 4) and second series (representational images, this section) of experiments. First of all, we have implemented new initialisation, mutation and crossover. Next, we have increased the mutation probability (0.25 in the first series, 0.5 in the second series), since we want to see more influence of our wide array of mutation operators. Furthermore, the use of existing vector images can be very memory intensive. In some cases, a single individual in the population can be several megabytes in size, and since all operations (mutation, crossover) are done in memory, and since we use a generational setup (which means that every generation we build up a new population next to the existing one), we ran out

of memory at multiple occasions. That is the main reason why we lowered our population size from 200 to 100.

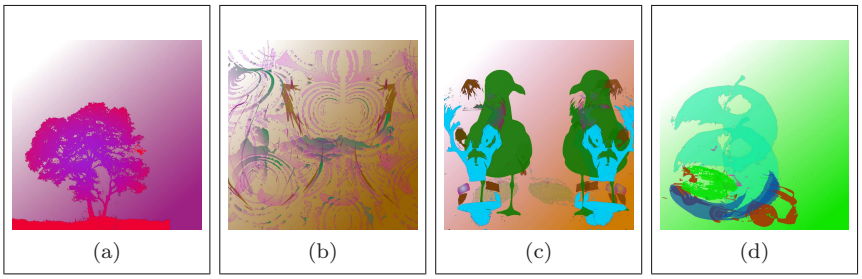
Experimental setup and results We performed 20 runs with our unsupervised genetic programming system using our aesthetic measure for pop art. The settings of our system are given in Table 6. In Figure 13 we show a portfolio of 35 images that we gathered from the 20 runs of our experiment, and Figure 14 shows a close-up of four images from this portfolio. Given the limited input image collection,

Figure 13: Portfolio of images gathered from twenty runs with SVG and Colour Contrast (hue) aesthetic measure

we think that the output is varied; varied in colour, composition, but also varied in the level of 'abstractness'. Most images contain representational content; parts of the image or the entire image refer in some degree to something recognisable, whereas some images have parts that are heavily processed by mutation and are less recognisable or not recognisable at all (and thus become abstract images).

6 Conclusions and Future Work

In this paper we have presented our investigations into the use of SVG or Scalable Vector Graphics in EvoArt. We have defined a number of research questions in Section 1, and we will answer them here. First, we wanted to know whether SVG is suitable as a representation for EvoArt. We have shown that we have successfully implemented SVG as a representation for EvoArt; we have implemented mutation, crossover and initialisation operators, both for abstract and for representational



(a) (b) (c) (d)

Figure 14: Close-up of four images from Figure 13; the first image 14a is a very simple image of the silhouette of a tree with a linear gradient background. Image 14b is very different; it contains a simple image of a group of seashells, but it has endured many mutation operations (most noteably the 'siamesetwin' operator) whereby the seashells have become barely recognisable. Images 14c and 14d are more recognisable but contain 'interesting' composition elements. Figure 14d contains a duplicate image of an apple over a 'wrinkled' car.

images. Implementing genetic operators for SVG is more complex than for symbolic expressions (due to several dependencies between the SVG elements), but it is certainly feasible.

Next, we wanted to know whether images evolved using SVG as a representation would result in images that are different from the 'typical' symbolic expression EvoArt systems (most notably those that employ the 'raster paradigm'). In Figure 1 we show eight images evolved in experiments using expression based representation (den Heijer & Eiben, 2011). We think it is safe to conclude that the images in Figures 5, 6, 7, 8, 13 and 14 are different in style from the ones in Figure 1. We think our images are also different from the image filter/ NPR approaches that we described in our section in Related Work (Baniasadi & Ross, 2013; Barile *et al.* , 2008; Bergen, 2009; Neufeld *et al.* , 2007); the described NPR/ Image filter approaches do not alter the main composition or outline of the underlying source image. Another difference is that our approach is able to combine multiple images into a new image, whereas the image filter/ NPR approaches usually operate on a single source image. We also think that our approach is different because our image operations, especially our mutation operators described in Section 5.2 are different from most image filters and NPR operations; our operators act on image fragment level (on fragments of SVG) whereas most image filters and NPR functions operate on pixel level, or on pixel block level. When we compare the visual output of our approach with the approach by Baker et al (Baker & Seltzer, 1994) we can safely conclude that our approach has a wider visual output; our output uses colour, gradient filters, several basic geometric shapes, line drawings and complex polygons, whereas the approach by Baker et al uses black and white line drawings.

In the first section we labelled many EvoArt as 'computer art'. An interesting question then could be 'Can EvoArt using SVG as a representation be labelled as computer art?' If we look at our first series of experiments, in which we evolved abstract images, we would probably have to answer 'yes' to that question. Several images in Figure 6 resemble early computer art by Michael Noll (Noll, 1967). However, if we look at our second series of experiments, in which we evolve rep-

resentational images, we could probably answer 'no' to that question. SVG does not prevent 'computer art-ness' per se, the difference clearly lies in the expressive power of the genetic operators initialisation, crossover and mutation.
Our third research question concerns the possibility to evolve representational (i.e., non-abstract) images using SVG. Our results confirm this. Most images from our experiments contain recognisable images or at least recognisable fragments. Although we used a small image collection, it will be trivial to repeat the experiments with bigger image collections. Clearly, having recognisable content in the final images was not a goal in itself. We achieved recognizability by using existing images as starting points.
As for the fourth and last research question regarding the evolution of surprising new images, our findings are positive as well. Many combinations of images and alterations of images result in images that are very different from the initial source images, sometimes leading to new and surprising images. Although we evolved pop-art in this research, we believe that SVG can be used for other, different categories of art and design, like collages of different kinds of images and shapes, the design of logos and album covers (SVG also supports the use of text elements).
We consider a number of possible routes for future work; first of all, we would like to improve the conversion of existing bitmap images to vector images. In our current setup we use potrace to convert bitmap images to SVG documents, but we think that a more elaborate image vectorisation algorithm will improve the quality of the SVG source material. Next, we think that there are several possibilities for new mutation operators. And we would like to exchange SVG documents with artists and designers, to blend the EvoArt process with the human Art process. The fact that SVG is already a standard among artists and designers is a clear advantage over many existing EvoArt genotype representations.

References

Ashlock, Daniel. 2006. Evolutionary Exploration of the Mandelbrot Set. *Pages 2079–2086 of:* Yen, Gary G., Lucas, Simon M., Fogel, Gary, Kendall, Graham, Salomon, Ralf, Zhang, Byoung-Tak, Coello, Carlos A. Coello, & Runarsson, Thomas Philip (eds), *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*. Vancouver, BC, Canada: IEEE Press.

Ashlock, Daniel A., & Tsang, Jeffrey. 2009. Evolved art via control of cellular automata. *Pages 3338–3344 of: IEEE Congress on Evolutionary Computation*. IEEE Press.

Baker, Ellie, & Seltzer, Margo. 1994. Evolving Line Drawings. *Pages 91–100 of: Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers.

Baluja, Shumeet, Pomerleau, Dean, & Jochem, Todd. 1994. Towards Automated Artificial Evolution for Computer-generated Images. *Connection Science*, **6**, 325–354.

Baniasadi, Maryam, & Ross, Brian J. 2013. Exploring Non-photorealistic Rendering with Genetic Programming. *Technical Report, CS-13-09, Department of Computer Science*.

Barile, Perry, Ciesielski, Vic, & Trist, Karen. 2008. Non-photorealistic Rendering Using Genetic Programming. *Pages 299–308 of: Proceedings of the 7th International Conference on Simulated Evolution and Learning*. SEAL '08. Berlin, Heidelberg: Springer-Verlag.

Bentley, P. J., & Corne, D. W. (eds). 2001. *Creative Evolutionary Systems*. San Mateo, California: Morgan Kaufmann.

Bergen, Steven. 2009. Evolving stylized images using a user-interactive genetic algorithm. *Pages 2745–2752 of: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*. GECCO '09. New York, NY, USA: ACM.

Bergen, Steven, & Ross, Brian. 2012. Automatic and interactive evolution of vector graphics images with genetic algorithms. *The Visual Computer*, **28**, 35–45.

Birren, Faber. 1987. *Principles of color: a review of past traditions and modern theories of color harmony*. Schiffer Publishing.

Collomosse, John. 2007. Evolutionary Search for the Artistic Rendering of Photographs. *In:* Romero & Machado (2007).

del Acebo, Esteve, & Sbert, Mateu. 2005. Benford's Law for Natural and Synthetic Images. *In:* Neumann *et al.* (2005).

den Heijer, E. 2012. Evolving art using measures for symmetry, compositional balance and liveliness. *Pages 52–61 of: Proceedings of the 4th IJCCI 2012*. Barcelona, Spain: ScitePress.

den Heijer, E., & Eiben, A. E. 2010a. Comparing Aesthetic Measures for Evolutionary Art. *Pages 311–320 of: Applications of Evolutionary Computation, LNCS vol. 6025*. Springer.

den Heijer, E., & Eiben, A. E. 2010b. Using Aesthetic Measures to evolve Art. *Pages 1–8 of: IEEE Congress on Evolutionary Computation*. IEEE Press.

den Heijer, E., & Eiben, A. E. 2011. Evolving Art Using Multiple Aesthetic Measures. *Pages 234–243 of: EvoApplications, LNCS 6625, 2011*.

den Heijer, E., & Eiben, A. E. 2012. Evolving Pop Art Using Scalable Vector Graphics. *Pages 48–59 of: EvoMusart 2012, Evolutionary and Biologically Inspired Music, Sound, Art and Design, LNCS 7247*. Malaga, Spain: Springer.

DiPaola, Steve, & Gabora, Liane. 2009. Incorporating characteristics of human creativity into an evolutionary art algorithm. *Genetic Programming and Evolvable Machines*, **10**(2), 97–110.

Eiben, A. E. 2008. Evolutionary Reproduction of Dutch Masters: The Mondriaan and Escher Evolvers. *Pages 211–224 of:* Romero, Juan, & Machado, Penousal (eds), *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*. Natural Computing Series. Springer.

Eiben, A. E., Nabuurs, R., & Booij, I. 2001. The Escher Evolver: Evolution to the People. *Pages 425–439 of:* Bentley, P.J., & Corne, D.W. (eds), *Creative Evolutionary Systems*. Academic Press.

G. Stiny, J. Gips. 1972. Shape Grammars and the Generative Specification of Painting and Sculpture. *Pages 1460–1465 of: Information Processing*.

Gooch, B., & Gooch, A. 2001. *Non-photorealistic Rendering*. A.K. Peters.

Greenfield, Gary R. 2000. Mathematical building blocks for evolving expressions. *Pages 61–70 of:* Sarhangi, R. (ed), *2000 Bridges Conference Proceedings*. Central Plain Book Manufacturing.

http://www.rgbstock.com. *Last accessed; 13th June 2013*.

Lewis, Matthew. 2000. Aesthetic Evolutionary Design with Data Flow Networks. *In: Proc. Generative Art*.

Machado, Penousal, & Cardoso, Amílcar. 1998. Computing Aesthetics. *Pages 219–229 of: Proceedings of the Brazilian Symposium on Artificial Intelligence, SBIA-98*. Springer-Verlag.

Machado, Penousal, & Cardoso, Amílcar. 2002. All the Truth About NEvAr. *Applied Intelligence*, **16**(2), 101–118.

Machado, Penousal, Nunes, Henrique, & Romero, Juan. 2010. Graph-Based Evolution of Visual Languages. *Pages 271–280 of: Applications of Evolutionary Computation, LNCS 6025*. Springer.

Machado, Penousal, Correia, João, & Romero, Juan. 2012. Expression-Based Evolution of Faces. *Pages 187–198 of:* Machado, Penousal, Romero, Juan, & Carballal, Adrian (eds), *Evolutionary and Biologically Inspired Music, Sound, Art and Design*. Lecture Notes in Computer Science, vol. 7247. Springer Berlin Heidelberg.

Matkovic, Kresimir, Neumann, László, Neumann, Attila, Psik, Thomas, & Purgathofer, Werner. 2005. Global Contrast Factor-a New Approach to Image Contrast. *In:* Neumann *et al.* (2005).

Neufeld, Craig, Ross, Brian, & Ralph, William. 2007. The Evolution of Artistic Filters. *In:* Romero & Machado (2007).

Neumann, László, Sbert, Mateu, Gooch, Bruce, & Purgathofer, Werner (eds). 2005. *Computational Aesthetics 2005*. Eurographics Association.

Noll, A.M. 1967. The Digital Computer as a Creative Medium. *IEEE Spectrum*.

O'Neill, Michael, Swafford, John Mark, McDermott, James, Byrne, Jonathan, Brabazon, Anthony, Shotton, Elizabeth, McNally, Ciaran, & Hemberg, Martin. 2009. Shape grammars and grammatical evolution for evolutionary design. *Pages 1035–1042 of: Proceedings of the 11th Annual conference on Genetic and Evolutionary Computation (GECCO)*. ACM.

Perry, Michael. 2011. *Pulled: A Catalog of Screen Printing*. Princeton Architectural Press.

Romero, Juan, & Machado, Penousal (eds). 2007. *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*. Natural Computing Series. Springer Berlin Heidelberg.

Rooke, Steven. 2001. Eons of genetically evolved algorithmic images. *In:* Bentley & Corne (2001).

Ross, Brian J., Ralph, William, & Zong, Hai. 2006. Evolutionary Image Synthesis Using a Model of Aesthetics. *Pages 1087–1094 of: IEEE Congress on Evolutionary Computation (CEC) 2006*.

Schattschneider, D., & Escher, M.C. 2004. *M.C. Escher: visions of symmetry*. Harry N. Abrams, Inc.

Schnier, Thorsten, & Gero, John S. 1996. Learning Genetic Representations As Alternative To Handcoded Shape Grammars. *In:* Gero, John S., & Sudweeks, Fay (eds), *Artificial Intelligence in Design '96*. Dordrecht, Netherlands: Kluwer.

Selinger, Peter. 2003. *Potrace: a polygon-based tracing algorithm*. http://potrace.sourceforge.net/potrace.pdf.

Sims, Karl. 1991. Artificial evolution for computer graphics. *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, **25**(4), 319–328.

Takagi, Hideyuki. 2001. Interactive Evolutionary Computation: Fusion of the Capacities of EC Optimization and Human Evaluation. *Proceedings of the IEEE*, **89**(9), 1275–1296.

Unemi, Tatsuo. 1999. SBART 2.4: breeding 2D CG images and movies and creating a type of collage. *Pages 288–291 of:* Jain, Lakhmi C. (ed), *KES*. IEEE Press.

Ventrella, Jeffrey. 2008. Evolving the Mandelbrot Set to Imitate Figurative Art. *Pages 145–168 of:* Hingston, Philip F., Barone, Luigi C., & Zbigniew, Michalewicz (eds), *Design by Evolution: Advances in Evolutionary Design*. Springer Berlin Heidelberg.

W3C, World Wide Web Consortium. 2005. *Scalable Vector Graphics (SVG) Full 1.2 Specification*. `http://www.w3.org/TR/SVG12/`.