Evolving Art with Scalable Vector Graphics

E. den Heijer
Objectivation and Vrije Universiteit Amsterdam
Dept. of Computer Science
Amsterdam, Netherlands
eelco@objectivation.nl

A.E. Eiben
Vrije Universiteit Amsterdam
Dept. of Computer Science
Amsterdam, Netherlands
gusz@cs.vu.nl

ABSTRACT

In this paper we introduce the use of Scalable Vector Graphics (SVG) as a representation for evolutionary art. We describe the technical aspects of using SVG in evolutionary art, and explain the genetic operators mutation and crossover. Furthermore, we compare the use of SVG with existing representations in evolutionary art. We performed a number of experiments in an unsupervised evolutionary art system using two aesthetic measures as fitness functions, and compared the outcome of the different experiments with each other and with previous work with symbolic expressions as the representation. All images and SVG code examples in this paper are available at
`http://www.few.vu.nl/~eelco`

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Miscellaneous; I.3 Computer Graphics [**I.3.3 Picture/Image Generation**]: Line and curve generation

General Terms

Experimentation

Keywords

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

1. INTRODUCTION

Over the last two decades, evolutionary art has developed from an experimental mix of computer art and evolutionary computation to an established research topic in evolutionary computation. Although there has been significant progress in various aspects of evolutionary art (notably in the field of interactive evolutionary computation (IEC) [18]) one cannot deny that some aspects of evolutionary art appear to be stuck in a local optimum; perhaps the most visible aspect is that a lot of evolutionary art looks like ... computer art. 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 [5]. We see a variety of images, but

GECCO'11, July 12–16, 2011, Dublin, Ireland.

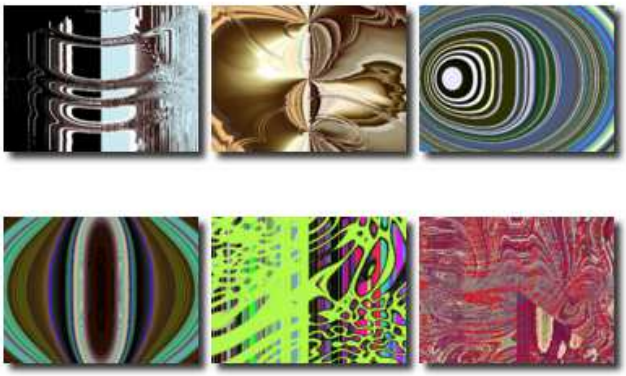


Figure 1: A portfolio of six images evolved using symbolic expressions, from [5]

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 evolutionary art, 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 evolutionary art, and compare our new technique (the use of SVG) with an established technique (symbolic expressions). An important starting point of many evolutionary art systems is the choice for symbolic (Lisp) expressions as the representation for the genotypes. The choice for symbolic expressions and the construction of the function set is a very important, but also a potentially restrictive choice in the design of an evolutionary art system. As Jon McCormack wrote [10] *"While there might exist a symbolic expression for generating the Mona Lisa for example, no such expression has been found by aesthetic selection"*. The research questions that we would like to address in this paper are; 1) is SVG a suitable representation for evolutionary art? We want to investigate whether it is feasible to implement SVG as a genetic representation, i.e. is it possible to implement all representation dependent components of an evolutionary art system (mutation, crossover and initialisation)? 2) are the resulting images different from the images produced by evolutionary art systems that use symbolic representation?
The rest of this paper is structured as follows; first we discuss a number of representations in evolutionary art in Section 2. Next, in Section 3 we briefly describe Scalable Vector Graphics. In Section

3.2 we describe SVG as a representation in an evolutionary art system that uses genetic programming. Section 4 describe a number of experiments and their results, and in Section 5 we present the conclusions and directions for future research. In the appendix we show an example of an evolved SVG document.

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 [13] and [2]. Some evolutionary art systems use IEC or supervised fitness assignment [17] [14], and in recent years there has been increased activity in investigating unsupervised fitness assignment [1] [4] [3] [15].

A number of different representations have been investigated for use in evolutionary art. We will briefly describe symbolic expressions and grammars below (other representations like L-Systems, Cellular automata etc. are out of scope of this paper).

Symbolic Expressions The most widespread representation within evolutionary art is the symbolic expression [17] [14] [4] [3]. The symbolic expression paradigm, pioneered by Karl Sims in 1991 [17] 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 as follows;

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 [8], some authors map the value v onto a colour index table [8] [4] [3] but the main idea is the same.

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 [7]. Shape grammars are especially useful in the context of design and architecture, since domain rules can be coded into the grammar. Examples of the use of shape grammars in evolutionary art/ design are [16] and [11]. [12] describes the use of shape grammars (using the Context Free language) to evolve multiple artworks in a similar style.

3. SCALABLE VECTOR GRAPHICS

Vector graphics deals with primitives like lines, points, curves and polygons and is complementary to raster graphics that deals with pixels. SVG is a graphics format developed and maintained by the World Wide Web Consortium (W3C) [19] [6] 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 evolutionary art 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 evolutionary art system. Next, the output of the evolutionary art system can be used as input for the artist or designer. Both evolutionary art 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[1]. 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

- `cssClass` - a Cascading Stylesheet class definition; a css class is 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. Basically, the css class determines the look and feel of a shape element.
- `filter` - defines a 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. Currently we have implemented the gaussian blur filter and the colour matrix filter. SVG specifies more filters but since each filter can have a large number of specific parameters, they are not easy to implement. We intend to implement more (if not all) filters from the SVG specification in the future.
- `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.
- `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).
- `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:

[1]SVG does not enforce a document to begin with a 'defs' element, but we do so in our implementation for reasons of simplicity

- `rect` - a rectangle shape
- `ellipse` - an ellipse; it has a centre coordinate, and x radius and an y radius. If the x radius and y radius are equal, the result is circle.
- `circle` - a special case of 'ellipse'; there is only 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 (see the appendix for a comprehensive example of an SVG document with many path elements.
- `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.
- `use` - the 'use' element is a special type of element, since it does not define a shape itself, but refers to a predefined element (defined in the 'defs' section of the SVG document) and performs a transformation on it. Transformations include scaling, rotating and translating.
- `image` - SVG also supports the use of raster image using the 'image' element. The image element is different from the other elements, since it does not constitute a vector graphics element, so many aspects of SVG do not apply (like styling using CSS, gradients etc.).

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. 'use' elements can also refer to a 'group' element. As we will see later, the interconnectedness of the shape elements with the declared elements can pose complex dependencies that may occur when performing a crossover or mutation on one or more SVG documents. Figure 1 shows a number of simple SVG example documents and their rendered images. The appendix shows an evolved SVG document with many 'path' elements and its rendered image.
The SVG specification is vast and complex, and we have not covered every aspect of it, nor have we implemented the entire SVG specification. 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.

3.2 SVG in Evolutionary Art

As stated previously, the most used representation in evolutionary art is the symbolic expression. The advantage of using symbolic expressions 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 will be more complex. In this section we will describe the genetic operators crossover and mutation, and we will describe the initialisation process. 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.

3.3 Mutation

The mutation operator processes an SVG document top-down, and (depending on the mutation rate) 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.. Currently, we have not implemented the insertion, replacement and removal of child elements upon mutation (but we intend to implement this in the near future).

3.4 Crossover

For the crossover operation, we implemented a one-point crossover operator specific for SVG. The crossover works on two parent svg documents and create 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). Since a SVG interpreter will refuse to render such a document (with references to non-existing elements), 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 a reference that does exist in the child document. 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'. 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.

3.5 Initialisation

Just like mutation and crossover, our initialisation adheres to the SVG standard, and produces only valid SVG documents. Initialisation uses a number of parameters to create new individuals. For

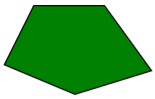
 `<circle cx="100" cy="50" r="40" stroke="black" stroke-width="2" fill="blue"/>`	 `<rect x="20" y="20" width="50" height="25" fill="red"/>`
 `<polygon points="220,100,300,210 170, 250,50,200,100,100" style="fill: green; stroke: black;stroke-width:2"/>`	 `<polyline points="50,50,200,50,200,200, 100,100,50,200"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

Table 1: Four simple examples of SVG code and their images

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).

	Various SVG Elements	Only 'path' element
Ross & Ralph	Experiment 1	Experiment 3
GCF	Experiment 2	Experiment 4

Table 3: Overview of experiments

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).

4. EXPERIMENTS

In order to explore the potential of SVG as a representation for evolutionary art 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 Bell Curve aesthetic measure [15] and two experiments were performed with the Global Contrast Factor aesthetic measure [9] (see Table 3 for more details). These aesthetic measures were used as fitness functions in an unsupervised evolutionary art system; no human evaluation/ interactive evolution was involved.

The Ross & Ralph Bell Curve 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 (hence the name 'Bell Curve'). The computation takes several steps and we refer to [15] for details. Previous experiments with the Ross & Ralph aesthetic measure as a fitness function in an unsupervised evolutionary art system have shown that the use of this measure often leads to images with rich colouring and smooth colour transitions [4]. 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 superpixels. Superpixels are rectangular blocks in the image. The contrast is calculated for several resolutions (2, 4, 8, 16, 25, 50, 100 and 200) and the average contrast is summed as

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

where r_k refers to the resolution of the superpixel, w_k refers to the weight of the contrast of the superpixel (the weight of the contrast differs per resolution) and p_k is a power factor. Both w and p were optimised using several experiments in [9]. In our implementation we used all the settings from [9], 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) [3]

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.

4.1 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.

4.1.1 Experiment 1: Ross & Ralph

In the first experiment we initialised the population with documents containing circle, poyline, polygon and path elements. We used the Ross & Ralph 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 im-

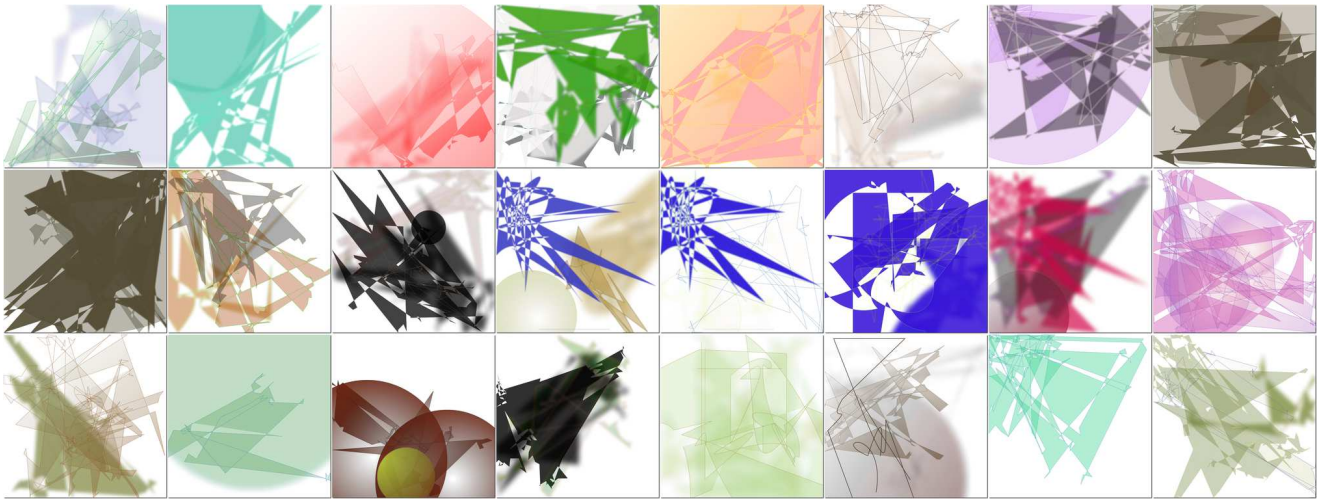


Figure 2: Portfolio of images gathered from ten runs with Ralph & Ross with various SVG elements (Experiment 1)

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

Table 4: Evolutionary parameters of our evolutionary art system used in our experiments

ages are shown in Figure 2; Almost all images have rich and variable colouring which is consistent with earlier experiments with the Ross & Ralph aesthetic measure [4] 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 in Figure 1).

4.1.2 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 3. The images evolved using the GCF show a lot of contrast, and that is similar to earlier findings [3]. 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 aesthetic measure).

4.2 Experiment 3 & 4: 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.

4.2.1 Experiment 3: Ross & Ralph

In the third experiment we evolved SVG document with only 'path' elements using Ross & Ralph as the fitness functions. The images of this experiments are in Figure 4.

The first thing that is striking is the variety of the images; it is 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. We think we can improve this by creating initialisation methods that use heuristics from art theory. The images from Experiment 3 are also varied in colour and this is consistent with Experiment 1 and previous work [4].

4.2.2 Experiment 4: GCF

The last experiment is identical to Experiment 3, except for the use of the Global Contrast Factor as the fitness function. The images of Experiment 4 are in Figure 5. 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 [3]

5. CONCLUSIONS AND FUTURE WORK

In section 1 we defined a number of research questions for this paper, and we will answer them here. First, we wanted to know whether SVG is suitable as a representation for evolutionary art. We think we have shown that we have successfully implemented SVG as a representation for evolutionary art; we have implemented a mutation, crossover and initialisation operator, and we intend to implement variants of these operators in the near future. Imple-

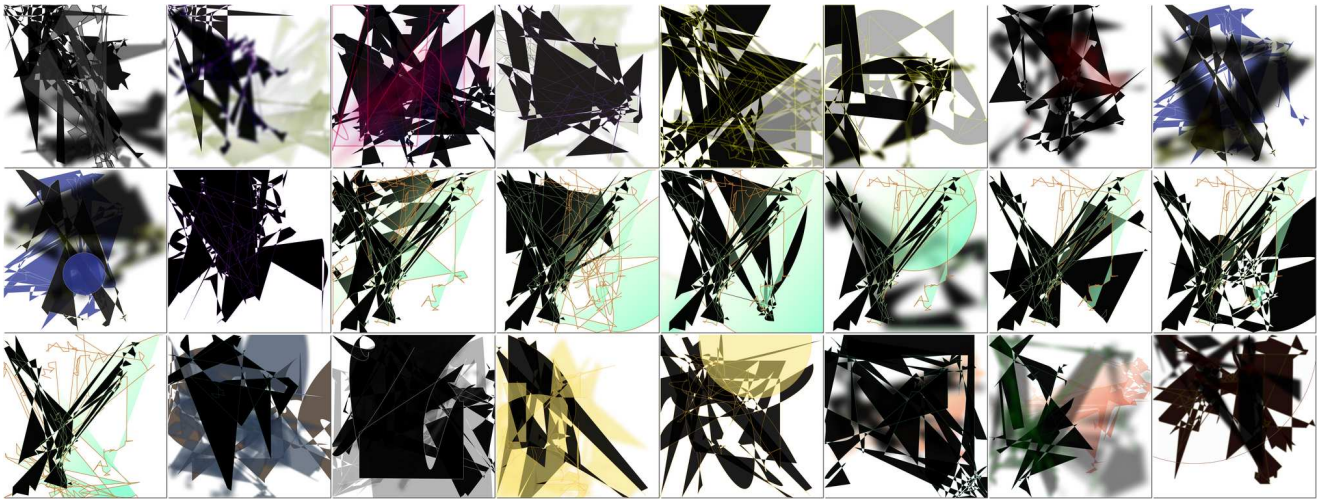


Figure 3: Portfolio of images gathered from ten runs with GCF with various SVG elements (Experiment 2)

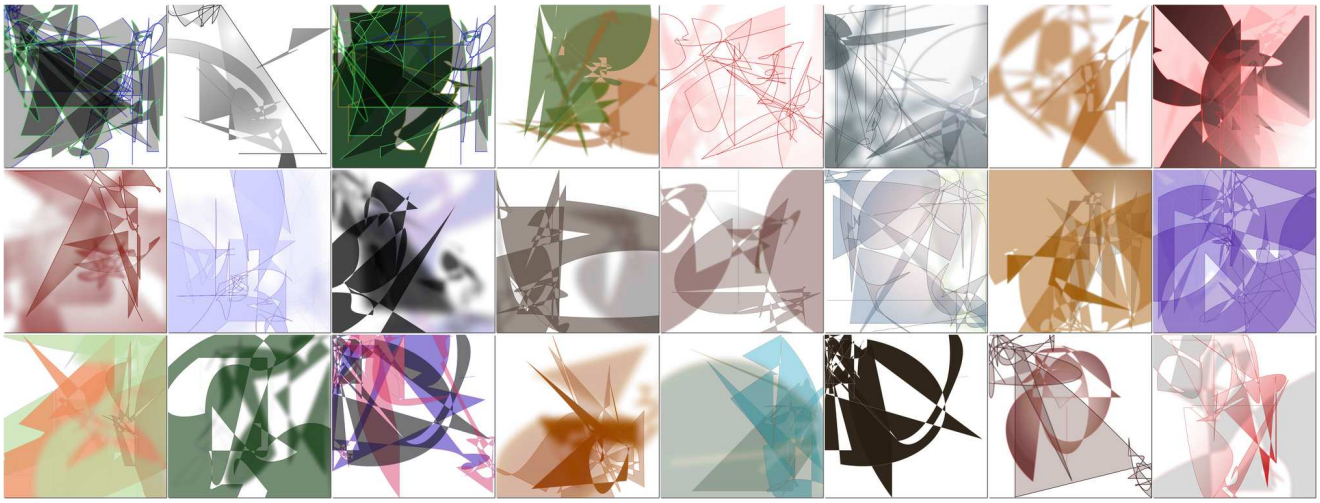


Figure 4: Portfolio of images gathered from ten runs with Ralph & Ross with the 'path' element (Experiment 3)

menting SVG as a representation was not easy, but it is 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 evolutionary art systems. Although we have barely touched the surface of the possibilities of SVG as a representation, we think that the images that we have evolved are different from the ones we evolved in earlier work using symbolic expressions. In Figure 1 we show six images evolved in experiments using expression based representation [5]. We think it is safe to conclude that the images in Figures 2, 3, 4 and 5 are different from the ones in Figure 1. In the first section we labelled many evolutionary art as 'computer art'. An interesting question then could be 'Can evolutionary art using SVG as a representation be labelled as computer art?' The answer to that question is probably 'yes'; the initial experiments result in images that are different from previous images, but could nevertheless be labelled as computer art. We do think, however, that SVG has a lot of potential. First, we think there are many ways to improve the interestingness of the images; first of all, we mainly used a uniform 'random' function to initialise various attributes of the SVG elements; we think we can improve the quality of the images by using heuristics from art theory, especially heuristics concerning the composition of an image (for example, the golden ratio, the rule of thirds, etc.). Second, we think there are many possible points for improvement; we intend to implement a number of additional SVG features; more SVG filters, the use of images, text etc.

6. REFERENCES

[1] Shumeet Baluja, Dean Pomerleau, and Todd Jochem. Towards automated artificial evolution for computer-generated images. *Connection Science*, 6:325–354, 1994.

[2] P. J. Bentley and D. W. Corne, editors. *Creative Evolutionary Systems*. Morgan Kaufmann, San Mateo, California, 2001.

[3] E. den Heijer and A. E. Eiben. Using aesthetic measures to evolve art. In *IEEE Congress on Evolutionary Computation (CEC 2010)*, Barcelona, Spain, 18-23 July 2010. IEEE Press.

[4] E. den Heijer and A.E. Eiben. Comparing aesthetic measures for evolutionary art. In *Applications of Evolutionary Computation, LNCS 6025, 2010*, pages 311–320, 2010.

[5] E. den Heijer and A.E. Eiben. Evolving art using multiple

Figure 5: Portfolio of images gathered from ten runs with GCF with the 'path' element (Experiment 4)

aesthetic measures. In *EvoApplications, LNCS 6625, 2011*, pages 234–243, 2011.

[6] J. Eisenberg. *SVG Essentials*. O'Reilly Media, 2002.

[7] J. Gips G. Stiny. Shape grammars and the generative specification of painting and sculpture. In *Information Processing*, pages 1460–1465, 1972.

[8] Gary R. Greenfield. Mathematical building blocks for evolving expressions. In R. Sarhangi, editor, *2000 Bridges Conference Proceedings*, pages 61–70, Winfield, KS, 2000. Central Plain Book Manufacturing.

[9] Kresimir Matkovic, László Neumann, Attila Neumann, Thomas Psik, and Werner Purgathofer. Global contrast factor-a new approach to image contrast. In László Neumann, Mateu Sbert, Bruce Gooch, and Werner Purgathofer, editors, *Computational Aesthetics*, pages 159–168. Eurographics Association, 2005.

[10] Jon McCormack. Open problems in evolutionary music and art. In Franz Rothlauf, Jürgen Branke, Stefano Cagnoni, David W. Corne, Rolf Drechsler, Yaochu Jin, Penousal Machado, Elena Marchiori, Juan Romero, George D. Smith, and Giovanni Squillero, editors, *EvoWorkshops*, volume 3449 of *Lecture Notes in Computer Science*, pages 428–436. Springer, 2005.

[11] Michael O'Neill, John Mark Swafford, James McDermott, Jonathan Byrne, Anthony Brabazon, Elizabeth Shotton, Ciaran McNally, and Martin Hemberg. Shape grammars and grammatical evolution for evolutionary design. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 1035–1042, New York, NY, USA, 2009. ACM.

[12] Henrique Nunes Penousal Machado and Juan Romero. Graph-based evolution of visual languages. In *Applications of Evolutionary Computation, Lecture Notes in Computer Science, 2010, Volume 6025/2010*, pages 271–280, 2010.

[13] Juan Romero and Penousal Machado, editors. *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*. Natural Computing Series. Springer Berlin Heidelberg, November 2007.

[14] Steven Rooke. Eons of genetically evolved algorithmic images. In Bentley and Corne [2], pages 339–365.

[15] Brian Ross, William Ralph, and Hai Zong. Evolutionary image synthesis using a model of aesthetics. In *IEEE Congress on Evolutionary Computation (CEC) 2006*, pages 1087–1094, 2006.

[16] Thorsten Schnier and John S. Gero. Learning genetic representations as alternative to handcoded shape grammars. In *Artificial Intelligence in Design '96*.

[17] Karl Sims. Artificial evolution for computer graphics. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, volume 25, pages 319–328. ACM Press, July 1991.

[18] Hideyuki Takagi. Interactive evolutionary computation: Fusion of the capacities of ec optimization and human evaluation. *Proceedings of the IEEE*, 89(9):1275–1296, 2001.

[19] World Wide Web Consortium (W3C). Scalable vector graphics (svg). `http://www.w3.org/Graphics/SVG/`.

APPENDIX

In this appendix we show a code example of an evolved SVG document; the document was evolved in experiment 3 (see Section 4). In order to fit the code on one page, we removed (unreferenced) code from the 'defs' element.

Figure 6: One of the images from Experiment 3; the SVG code is on the next page

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
  version="1.1" width="1000" height="1000" viewBox="0 0 1000 1000"
  viewPort="0 0 1000 1000">
  <defs>
    <filter id="filter_20110111T194020_972_053">
      <feGaussianBlur in="SourceGraphic" stdDeviation="12" />
    </filter>
    <filter id="filter_20110111T194020_972_019">
      <feGaussianBlur in="SourceGraphic" stdDeviation="19" />
    </filter>
    <filter id="filter_20110111T194020_972_082">
      <feGaussianBlur in="SourceGraphic" stdDeviation="17" />
    </filter>
    <linearGradient id="linearGradient_20110111T194020_972_042"
       filter="url(#filter_20110111T194020_972_082)" x1="12%" y1="12%" x2="95%" y2="79%">
      <stop offset="0%" style="stop-color : rgb(255,0,0); stop-opacity : 0.00; " />
      <stop offset="100%" style="stop-color : rgb(255,255,255); stop-opacity : 1.00; " />
    </linearGradient>
    <linearGradient id="linearGradient_20110111T194020_973_003"
      filter="url(#filter_20110111T194020_972_019)" x1="12%" y1="9%" x2="84%" y2="77%">
      <stop offset="0%" style="stop-color : rgb(0,0,255); stop-opacity : 0.00; " />
      <stop offset="100%" style="stop-color : rgb(255,255,0); stop-opacity : 1.00; " />
    </linearGradient>
    <style id="style_20110111T194020_974_035" type="text/css"><![CDATA[.c4314694 {
    fill    : #ff;
    stroke    : #ff0000;
    stroke-width    : 3;
    fill-opacity    : 0.90;
  }
  .c4610956 {
    fill    : url(#linearGradient_20110111T194020_972_042);
    stroke    : #ff;
    stroke-opacity    : 0.6;
    fill-opacity    : 0.74;
  }
  .c6957277 {
    fill    : url(#linearGradient_20110111T194020_973_003);
    stroke    : #ff;
    stroke-linejoin    : miter;
    fill-opacity    : 0.02;
  }]]></style>
  </defs>
  <path width="215" height="683" d="M757 787 L115 459 H89 Q20 394 0 328 S125 303
     7 259 V0 M654 216 H514 A490 378 49 1 0 478 465 Q447 500 423 454 T488 373 S457
     486 503 373 Q1000 399 656 295 M611 462 L930 67 H960 V697 Z" fill-rule="nonzero"
     class="c4314694" />
  <path width="883" height="562" d="M462 25 V567 H503 T346 113 H270 M799 236 A404
     687 96 1 0 490 815 T415 790 A456 427 169 0 1 157 555 V207 V558 V0 S352 327 257
     469 V262 Z" fill-rule="nonzero" class="c6957277" />
  <path width="846" height="893" d="M633 373 A660 812 65 0 1 432 590 S462 592 413
     1000 V659 H384 L660 1000 L575 548 L303 852 V642 A339 1000 20 1 0 660 874 L504
     652 L539 899 V90 A526 624 45 0 0 428 938 C660 598 509 799 656 799 Z"
     fill-rule="nonzero" class="c4314694" />
  <path width="930" height="34" d="M635 308 C492 775 346 714 416 756 S34 575 371 874
     T866 971 S1000 856 617 748 H806 S649 182 483 287 S721 298 637 177 H805 T863 172
     L602 271 A753 0 136 1 0 730 145 S954 0 686 231 V559 H516 M888 42 T852 84 A711
     399 88 0 0 888 0 L674 165 V708 C787 255 457 0 692 257 H592 L668 319 A954 277
     8 0 0 606 0 T518 178 M332 624 Q673 112 767 189 Z" fill-rule="nonzero"
    class="c4610956" />
  <path width="784" height="219" d="M290 188 H178 C615 1000 630 1000 614 1000 S843
     941 958 1000 V658 M664 715 M345 610 V194 L137 272 Q217 225 70 350 Q165 356 47
     324 M46 915 L260 314 T260 269 H166 C217 260 260 262 168 207 S209 321 219 176 Z"
     fill-rule="evenodd" class="c4610956" filter="url(#filter_20110111T194020_972_053)" />
  <path width="646" height="811" d="M925 752 A470 728 177 1 0 431 605 T547 650 M553
     565 M324 247 T263 534 M485 581 S381 588 407 534 A518 553 117 1 0 547 608 A463 762
     167 1 0 378 715 A498 732 6 0 0 479 819 L404 570 T371 613 T344 651 A459 820 15 0
     1 447 591 H547 S547 680 466 611 T547 769 H487 V471 V545 Z" fill-rule="nonzero"
     class="c4610956" />
</svg>
```

Listing 1: SVG Code example