Evolving pop art using Scalable Vector Graphics

E. den Heijer[1,2] and A.E. Eiben[2]

[1]Objectivation B.V., Amsterdam, The Netherlands
[2]Vrije Universiteit Amsterdam, The Netherlands
eelco@objectivation.nl, gusz@cs.vu.nl ,
WWW home page: http://www.cs.vu.nl/~gusz/

Abstract. In this paper we present our findings of our continued investigation into the use of Scalable Vector Graphics as a genotype representation in evolutionary art. In previous work we investigated the feasibility of SVG as a genetic representation for evolutionary art, and found that the representation was very flexible, but that the potential visual output was somewhat limited by the simplicity of our genetic operators. In this paper we extend on this work, and introduce various new, more expressive genetic operators for SVG. We show that SVG is a flexible and powerful representation for evolutionary art, and that the potential visual output is only limited by the design of the genetic operators. With the genetic operators that we describe in this paper, we are able to evolve art that is visually similar to screen printing art and pop art.

1 Introduction

In this paper we explore the possibilities of evolving representational art using Scalable Vector Graphics as the genetic representation. In the last decades, several types of evolutionary art systems have been designed and built, and many of them evolve abstract or non-representational art. We believe that the field of evolutionary art will benefit from techniques that will enable the evolution of representational art. In previous work we introduced the use of Scalable Vector Graphics (SVG) as a representation in evolutionary art, and although we showed that SVG is a viable representation for evolutionary art, we did not evolve representational art, mainly because of the limitations of our genetic operators. In this paper we want to extend on that work, and use SVG to evolve representational images. In our approach we re-use existing images to create a new image. Our system has to be able to manipulate the existing images at various levels, since we do not want to merely copy/ paste existing images. The existing images should be a starting point, or an inspiration, but the resulting image should be new and surprising. The research questions of this paper are:

1. Can we evolve representational, non-abstract art using SVG?
2. 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 evolutionary art. In

section 3 we describe Scalable Vector Graphics and we describe our new genetic operators for SVG in section 4. We performed experiments with our new genetic operators and we describe them in section 5. We give conclusions and directions for future research in section 6.

2 Representation in Evolutionary Art

Evolutionary art is a field where methods from Evolutionary Computation are used to create works of art [2, 19]. Some evolutionary art systems use IEC or supervised fitness assignment [20, 24], and in recent years there has been increased activity in investigating unsupervised fitness assignment [1, 6, 7, 21]. A number of different representations have been investigated for use in evolutionary art. We will briefly describe symbolic expressions, grammars and image filters.

Symbolic Expressions The most widespread representation within evolutionary art is the symbolic expression [7, 11, 12, 20, 24]. The symbolic expression paradigm, pioneered by Karl Sims in 1991 [24] works roughly as follows; each genome is a symbolic expression (i.e. a Lisp function tree) that consists of functions from a function set and terminals from a 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 calculation of the phenotype from the the genotype is done by calculating the colour for each pixel by calculating the function value of the expression for each (x, y) coordinate. 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 [11], some authors map the value v onto a colour index table [6, 7, 11] but the main idea is the same.

Grammars A shape grammar is a formal description of a design and has been pioneered by Stiny and Gips in 1972 [9]. 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 [22] and [16]. [17] describes the use of shape grammars (using the Context Free language) to evolve multiple artworks in a similar style.

Using images as a source 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. [4] describes an approach that using non-photorealistic rendering or NPR [10] to produce synthetic oil paintings from images; the author uses a genetic algorithm to find suitable values for his NPR system. In [14] the authors describe the evolution of a NPR system using genetic programming, whereby the authors use a number of image filter primitives.

Other Other representations used in evolutionary art are cellular automata, several types of fractals and L-systems.

From this short overview we see that only a few evolutionary art systems use a representation that re-use existing images to evolve new images. For evolutionary art 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 NPR work by Neufeld et al [14].

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) [25] 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 (or gradient), the background colour (or gradient), the stroke width, the stroke colour etc. In short, the css class determines the look and feel of a shape element.
- `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.

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

– `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).

There are also the elements `mask` and `pattern` that can be defined inside a 'defs' element, and although we have implemented them, we have not used them in our experiments in this paper. Next to the 'defs' element, an SVG document can have a number of shape elements, like a rectangle, an ellipse, a circle, a polygon etc. In this paper we only use the 'group' element and the 'path' element;

– `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.
– `path` - path is the most versatile SVG 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 four different curve commands.

4 Genetic operators for SVG

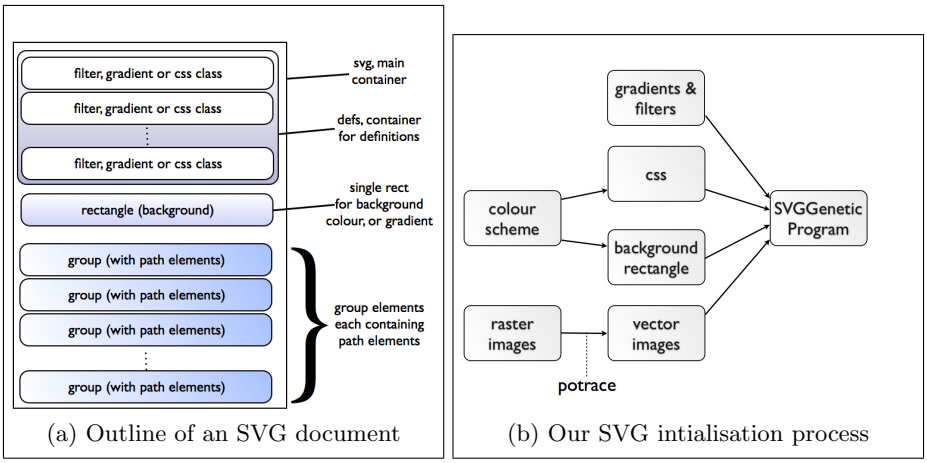
In this section we describe our operators initialisation, mutation and crossover.

4.1 Initialisation

In previous work we initialised SVG genetic programs with path elements and polygons using random initialisation [8]. This approach produced some interesting images, although most images had an artificial, abstract flavour. In this paper we intend to depart from evolving strictly abstract art, and decided to use existing images as a starting point. Our initialisation process is represented in Figure 1b. The images were taken from a collection of personal photographs of the first author. We also did experiments with a number of image collections that we downloaded from the Internet, but we chose not to publish these results, since we want to avoid copyright issues. From the photographs (raster images) we create vector images. We used the publicly available program 'potrace'[2][23] 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 100 colour schemes with 2 to 5 colours per colour scheme. To summarize, the step of initialisation are;

1. randomly choose one colour scheme

[2] Available at http://potrace.sourceforge.net/

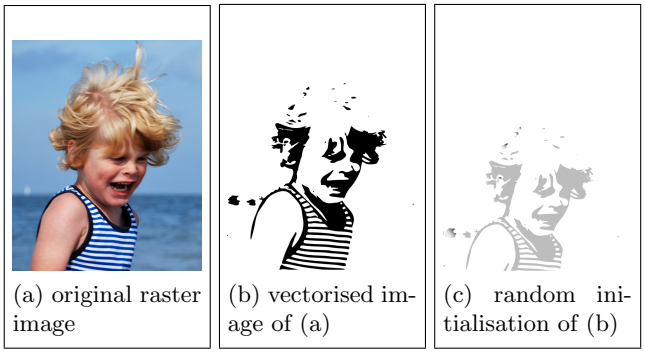


(a) Outline of an SVG document

(b) Our SVG intialisation process

Fig. 1: The outline of a typical genetic art program based on SVG (1a) and the schematic outline of our SVG genotype initialization process (1b)

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)
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, css class to all path elements



(a) original raster image

(b) vectorised image of (a)

(c) random initialisation of (b)

Fig. 2: 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.

All the elements are combined into an SVG document. The SVG document and the colour scheme together form the genotype of our evolutionary process; both are subject to mutation, and the SVG document is also subject to crossover.

4.2 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 1 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
```

Macro level mutation We have implemented the following macro level operators;

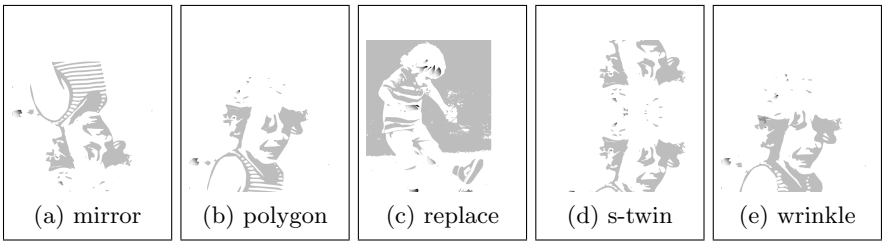
- `thicken` - this operator samples another image from the image collection and adds it at a random point in the composition
- `thin` - opposite of `thicken`; this operator removes a random chosen image from the composition (unless there is only one image on the canvas left; in that case the `thin` operator 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)

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 in some cases 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.
- `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. The effect on portrait images is sometimes funny, and sometimes unpleasant (the images sometimes resemble the work of Francis Bacon).

Figure 3 shows five different mutations on the image of Figure 2c[3]

[3] On `http://oursite.org/` we show more images of all mutations

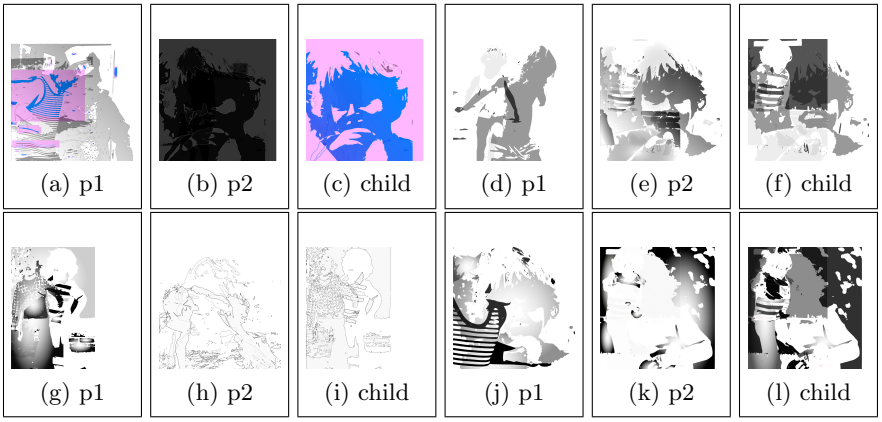


(a) mirror (b) polygon (c) replace (d) s-twin (e) wrinkle

Fig. 3: Examples of five different mutations of the original from Figure 2c

4.3 Crossover

We implemented a uniform crossover operator to create 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 4.



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

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

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

5 Experiments and Results

We performed experiments to evaluate the applicability of SVG. In our series of experiments we did a number of runs with unsupervised evolution (no human in

the loop). The aesthetic evaluation was performed by an aesthetic measure that we designed for the purpose of evolving aesthetically pleasing images in pop-art style. In the next subsections we will describe our aesthetic measure, present the parameters of our evolutionary art system, and present the resulting images.

A simple aesthetic measure for pop art In previous work we have applied a number of aesthetic measures in evolutionary art, 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 [5] and Ralph Bell Curve [21]) assigned low scores to the evolved images, also to images that we liked ourselves. We decided to create a simple aesthetic measure that favors contrast in hue, as is often seen in screen printing and pop art [18]. Our aesthetic measure is a combination of two ideas; the first idea comes from the Global Contrast Factor or GCF [13]. 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 [3]; 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 (like a colour combines well with tints of itself, etc.), 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 (R_1 and R_2), calculate the average hue for both regions, and calculate the difference between the average hues. Repeat this step for all regions of the image, for a number of different resolutions, and calculate the average hue difference.

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 uniform crossover
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 1: Parameters of our evolutionary art system used in our experiments

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 1. In Figure 5 we show a portfolio of 40 images that we gathered from the 20 runs of our experiment.

Fig. 5: Portfolio of images gathered from twenty runs with SVG and Colour Contrast (hue) aesthetic measure

Given the limited input image collection, 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).

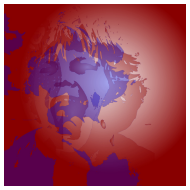
6 Conclusions and Discussion

In this paper we have described our extended investigations in using SVG as a genotype in evolutionary art. Our first research question was to investigate

(a) image 1

(b) image 2

(c) image 3



(d) image 4

Fig. 6: Close-up of four images from Figure 5; notice the difference in 'abstractness' between image 1 and image 2; image 1 has a large number of overlapping images, whereas the second image is a very simple image, almost like a poster or album cover. The third image is good example of the 'siamesetwin' mutation operator, producing Rorschach like images. The fourth image is relatively simple in composition, but can be considered a 'lucky shot', since two different images of faces in contrasting colours overlap on almost the same position in the image.

whether it was possible 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 with a narrow subject (all photographs were portraits of the 4 year old son of the first author), it will be trivial to repeat the experiments with bigger image collections with more varied subjects. Clearly, having recognisable content in the final images was not a goal in itself. We achieved recognisability by using existing images as starting points. As for the second research question regarding the evolution of surprising new images, our findings are positive as well. Many combinations of images and alterations of images are 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).

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. Faber Birren. *Principles of color: a review of past traditions and modern theories of color harmony*. Schiffer Publishing, 1987.
4. John Collomosse. Evolutionary search for the artistic rendering of photographs. In Romero and Machado [19], pages 39–62.
5. Esteve del Acebo and Mateu Sbert. Benford's law for natural and synthetic images. In Neumann et al. [15], pages 169–176.

6. 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.
7. 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.
8. E. den Heijer and A.E. Eiben. Evolving art with scalable vector graphics. In *Proceedings of the 13th Annual conference on Genetic and evolutionary computation*, GECCO '11, pages 427–434. ACM, 2011.
9. J. Gips G. Stiny. Shape grammars and the generative specification of painting and sculpture. In *Information Processing*, pages 1460–1465, 1972.
10. B. Gooch and A. Gooch. *Non-photorealistic Rendering*. A.K. Peters, 2001.
11. 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.
12. Penousal Machado and Amílcar Cardoso. All the truth about nevar. *Applied Intelligence*, 16(2):101–118, 2002.
13. Kresimir Matkovic, László Neumann, Attila Neumann, Thomas Psik, and Werner Purgathofer. Global contrast factor-a new approach to image contrast. In Neumann et al. [15], pages 159–168.
14. Craig Neufeld, Brian Ross, and William Ralph. The evolution of artistic filters. In Romero and Machado [19], pages 335–356.
15. László Neumann, Mateu Sbert, Bruce Gooch, and Werner Purgathofer, editors. *Computational Aesthetics 2005: Eurographics Workshop on Computational Aesthetics in Graphics, Visualization and Imaging 2005, Girona, Spain, May 18-20, 2005*. Eurographics Association, 2005.
16. Michael O'Neill, John Mark Swafford, James McDermott, Jonathan Byrne, Anthony Brabazon, Elizabeth Shotton, Ciaran McNally, and Martin Hemberg. GECCO '09, pages 1035–1042. ACM, 2009.
17. 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.
18. Michael Perry. *Pulled: A Catalog of Screen Printing*. Princeton Architectural Press, 2011.
19. 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.
20. Steven Rooke. Eons of genetically evolved algorithmic images. In Bentley and Corne [2], pages 339–365.
21. 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.
22. Thorsten Schnier and John S. Gero. Learning genetic representations as alternative to handcoded shape grammars. In *Artificial Intelligence in Design '96*.
23. Peter Selinger. Potrace: a polygon-based tracing algorithm. http://potrace.sourceforge.net/potrace.pdf, 2003.
24. 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.
25. World Wide Web Consortium (W3C). Scalable vector graphics (svg). http://www.w3.org/Graphics/SVG/.