

Animated Drawings Rendered by Genetic Programming

Perry Barile
School of Computer Science
and Information Technology,
RMIT University
Melbourne, Australia
bpasqual@cs.rmit.edu.au

Marsha Berry
School of Creative Media,
RMIT University
Melbourne, Australia
marsha.berry@rmit.edu.au

Vic Ciesielski
School of Computer Science
and Information Technology,
RMIT University
Melbourne, Australia
vc@cs.rmit.edu.au

Karen Trist
School of Creative Media,
RMIT University
Melbourne, Australia
karen.trist@rmit.edu.au

ABSTRACT

We describe an approach to generating animations of drawings that start as a random collection of strokes and gradually resolve into a recognizable subject. The strokes are represented as tree based genetic programs. An animation is generated by rendering the best individual in a generation as a frame of a movie. The resulting animations have an engaging characteristic in which the target slowly emerges from a random set of strokes. We have generated two qualitatively different kinds of animations, ones that use grey level straight line strokes and ones that use binary Bezier curve strokes. Around 100,000 generations are needed to generate engaging animations. Population sizes of 2 and 4 give the best convergence behaviour. Convergence can be accelerated by using information from the target in drawing a stroke. Our approach provides a large range of creative opportunities for artists. Artists have control over choice of target and the various stroke parameters.

Categories and Subject Descriptors

I.3.6 [Computer Graphics]: Methodology and Techniques; I.2.2 [Artificial Intelligence]: Automatic Programming

General Terms

Algorithms

Keywords

Evolutionary Search, Non Photorealistic Rendering, Evolved Art, Genetic Art

Track: Genetic Programming

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.
Copyright 2009 ACM 978-1-60558-325-9/09/07 ...\$5.00.

1. INTRODUCTION

Genetic programming has proved to be a reliable and robust method for solving many difficult scientific, engineering and industrial problems. In these kinds of problems there is a clear, objective measure of fitness. The evolutionary process is generally required to deliver a solution that is optimal by some criteria. In the art world the focus is on creativity rather than optimization. Evolution, through crossover and mutation, has the potential to combine components in ways that humans normally wouldn't think of. The potential in art exists for evolving new styles. Also, the evolutionary paradigm has great potential for producing interesting animations. Generally fitness improves as the number of generations increases. Visualizing the improvement can provide the underlying dynamic of an animation.

In this paper we describe an approach to generating drawings of a kind that have not been done by human artists. Our drawings are produced by a sequence of strokes in a somewhat haphazard manner not typically employed by human artists. We have implemented pencil-like strokes – grey level straight lines – and ink-like strokes – black curves on white. We require the evolutionary process to deliver a program that executes a set of strokes that will give a likeness of a target image. We produce animations by using the best individual in a generation as a frame of a movie. The movies have the engaging characteristic that the subject gradually emerges from a random collection of strokes.

The challenge to artists, graphic designers and photographers is to produce imagery that engages a user. Good artistic design considers interaction, communication and temporal relationships between a viewer and a subject. While various mature software products, for example Photoshop and GIMP, allow a user to apply artistic effects to still images, we seek to provide a designer with the ability to explore the search space of the rendered output. An artist can draw on the potential of repetition and subtle variation to strengthen the artistic effects and power of their artwork. The artist user can start with a photographic image or with a drawing and experiment to get the effects desired.

Historically a large volume of artwork has been produced by artists using various kinds of brush strokes on some kind of canvas. These include paint brush strokes with oil or water based paint, pencil lines and charcoal sketches. A

large number of different styles has emerged. In the area of black and white drawings, for example, some drawings are rendered with a large number of very fine pencil strokes, while some artists can render powerful images with just a few charcoal strokes [5].

Our overall goal is not to emulate artists, but to determine whether we can produce interesting images and animations by using stroke based rendering in the domain of black and white drawings. In particular our research questions are:

1. How can a stroke based rendering of a target image be produced with genetic programming?
2. What kinds of strokes and fitness measures will give interesting animations?
3. What is a suitable configuration of genetic programming?
4. How does the use of information from the target image influence the rendering?

2. RELATED WORK

There has been considerable interest in evolved art in recent years. Some of this work has been compiled into a new book and DVD [25]. The work in evolutionary art can be divided into two main approaches. In the first approach the task is to generate a visually appealing image from scratch. The user guides the evolution by interactively providing fitness judgments. Images are generated from mathematical functions [18, 21, 28], or from other structures such as grammars or L-Systems [22]. In the second approach the task is to generate a visually appealing image by some kind of non photorealistic rendering technique. The fitness function for the evolution is a measure of similarity to a target image. The overall aesthetic quality is still judged by the user, but not directly. If a satisfactory result is not obtained the user changes some of the parameters, either interactively or by performing another run. Our work falls into this category.

2.1 Stroke Based Rendering

A stroke based algorithm generates a set of strokes and applies them in a linear sequence. [15] achieved a step forward in computer graphics. Supplying a user with a source image and a canvas image the same size as the source image, a user was able to pick points on the source image. The selected point on the source image was sampled and colour information extracted. This information was transferred to the corresponding point on the canvas image and one of several choices of artistic effects would be applied on the canvas. This process of software-based image point sampling combined with user-driven stroke placement proved to be effective and enabled images of aesthetic quality to be created. There have been many subsequent efforts to produce artistic effects by utilising stroke-based rendering algorithms. Hertzmann developed a technique that drew curved spline strokes along edges discovered in a source image and established that target images can be faithfully reproduced by drawing a large number of small strokes[17]. [27] describes an approach to generating curved strokes.

2.2 Non-Photorealistic Rendering

2.2.1 Classical Methods for NPR

Non-photorealistic rendering is a major sub area of computer graphics and has been the subject of the NPAR series of conferences held since 2000. There has been considerable work on developing classical computer vision based algorithms for generating non-photorealistic renderings of a target photograph. In [3], Baxter et al. describe a tool which models natural media and simulates the flow and drying of paint. In [8], Chu et al. simulate East Asian ink and water-colour painting techniques. Kasao and Miyata [1] describe algorithms for a number of different styles of painting, including oil and water colour. Pencil renderings are described by [26] while algorithms for pen and ink renderings are described in [29] and [13].

2.2.2 Evolutionary Methods for NPR

A number of researchers have used evolutionary techniques to generate non-photorealistic renderings. Some of this work is reviewed in [20]. In early work by Baker and Seltzer [2] a genetic algorithm representation and interactive fitness evaluation are used to evolve line drawings of faces. Chakraborty [6] describes an approach to rendering a target image using a genetic algorithm representation and a very simple set of flat rectangular brush strokes. Colomosse [10] describes an investigation into the generation of painterly renderings in which a genetic algorithm representation is used for representing brush strokes and associated parameters values. In [30] Semet et al. generate painterly and pencil sketch renderings using an ant colony model. Edge maps and different models of the depositing of pheromones are used to achieve the different artistic effects. The user changes the parameter settings during the course of the run to guide the evolution according to their desired aesthetic criteria. Neufeld et al. [24] describe an approach to evolving artistic filters. An empirical model of aesthetics is used in conjunction with genetic programming and multi-objective optimization. This work uses automated fitness assessment rather than human judgment.

2.3 Animated Evolutionary Art

Most of the work in evolutionary art has tended to focus on evolving static images, the installation of McCormack [23] being a notable exception. Evolutionary techniques have been used to generate animated sequences of the development of locomotion in artificial creatures [7]. Hart [16] has developed animations in which an evolved image, represented by mathematical expression tree, is morphed into another evolved image by a genetic cross dissolve process. The cross dissolve process is computed as a sequence of crossover and mutation operations. [9] and [12] describe animations of photomosaic images. The target image is rendered as a photomosaic and the animation is constructed from the evolutionary search.¹

3. REPRESENTATION

In this section we describe the way we have configured a genetic programming system to generate grey level and black

¹Our animated renderings can be found at <http://evol-art.cs.rmit.edu.au>

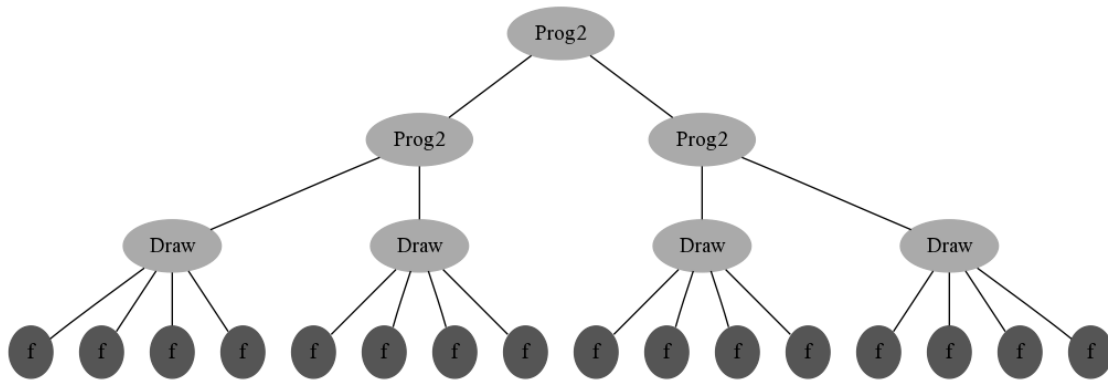


Figure 1: A Typical Program Tree. Uses only Prog nodes of arity 2.

and white drawings. We then describe the associated drawing functions and present visual results of those functions applied to two target images.

A pencil drawing can be viewed as sequence of grey strokes on a white canvas. A stroke could be a straight line or some kind of curve. Each stroke will have a number of parameters, such as grey level, length, thickness and orientation. Similarly an ink drawing can be viewed as a sequence of black strokes on a white canvas. A good choice of strokes will give a good rendering of an image. Finding such a sequence is a significant search problem which could be a candidate for evolutionary search. Since a variable number of strokes needs to be represented, one approach would be to use a messy genetic algorithm [14]. In this approach an individual is represented as a variable length string of codons and the standard mutation and crossover operators are modified to deal with variable lengths. For a pencil drawing, an individual could represent a sequence of pencil strokes and the associated parameter values. Alternatively, an individual could be a program which executes a sequence of parameterized draw functions. We have chosen the second representation and used tree based genetic programming [19]. The primary reason for our choice is flexibility of development and experimentation. In the GP representation experimenting with a new stroke with a number of new parameters requires only re-coding of the draw function, the chromosome remains the same. In the messy GA representation the chromosome needs to be changed to accommodate the new parameters, as well as the associated draw function.

3.1 Functions and Terminals

In our representation there are two kinds of functions, *Draw* functions and *ProgN* functions. The *Draw* functions take a number of floating point parameters. Execution of a *Draw* function results in the setting or changing of pixel values on the canvas. The *ProgN* functions are used for sequential execution of their arguments as in [19]. *ProgN* functions take N input arguments. *ProgN* functions accept both *ProgN* and *Draw* functions as input arguments, while *Draw* functions accept only terminals. As an example, programs which contain only *ProgN* nodes of arity 2 would be generated according to the following grammar:

```

PROG2  -> DRAW | Prog2(PROG2,PROG2)
DRAW   -> Draw(T,T,T,T)
T      -> float
  
```

A program from this grammar is shown in Figure 1. Programs are evaluated using pre-order traversal. The effect of this is that a tree can be viewed as a linear sequence of pencil strokes. Generally *ProgN* nodes will be towards the root of the tree. *Draw* nodes will always be one level up from the leaves. The only terminals are the random floating point numbers that are arguments to the *Draw* functions. They are restricted to the range $[0, 1]$

We have experimented with 4 different drawing functions. These are described below.

3.1.1 Drawing Function #1: Pencil, Blind Search

A pencil stroke is rendered in the form of a straight gray-scale line at an angle, with a finite length. When a pencil stroke is to be painted, the parameters it receives from the GP representation are used to represent the following characteristics of a line: Position, Length, Angle and Colour. Whenever a pixel within a line is drawn on a pixel that has been previously drawn, the two colours to be combined are added together and then averaged. The algorithm is characterised as a blind search because pencil strokes are drawn without reference to the target image. The fitness function employed is a simple pixel to pixel comparison between a rendered image and the target image. The fitness function seeks to minimise the difference. Figure 2 shows a target and a number of the evolved images from this technique. The technique generates interesting animations but exhibits very slow convergence towards the target.

3.1.2 Drawing Function #2: Pencil, Guided Search

A modification of function #1 is to introduce some knowledge of the target image to the pencil stroke function. When a pixel is about to be drawn, the function is given the value of the corresponding pixel on the target image. The pixel is changed if and only if the colour about to be drawn is closer to the target pixel than the colour that has already been drawn on the canvas. This guarantees that pixels are drawn only if they promote convergence towards the target. An example target and output are shown in Figure 3. By using localised information about the target image, the function is able to more faithfully reproduce the target. Even at the initial generation the rendered image displays some texture information of the target. An interesting feature is that this function draws roughly the same number of pencil strokes per rendering as does draw function #1, yet is able to convey much more meaningful information.

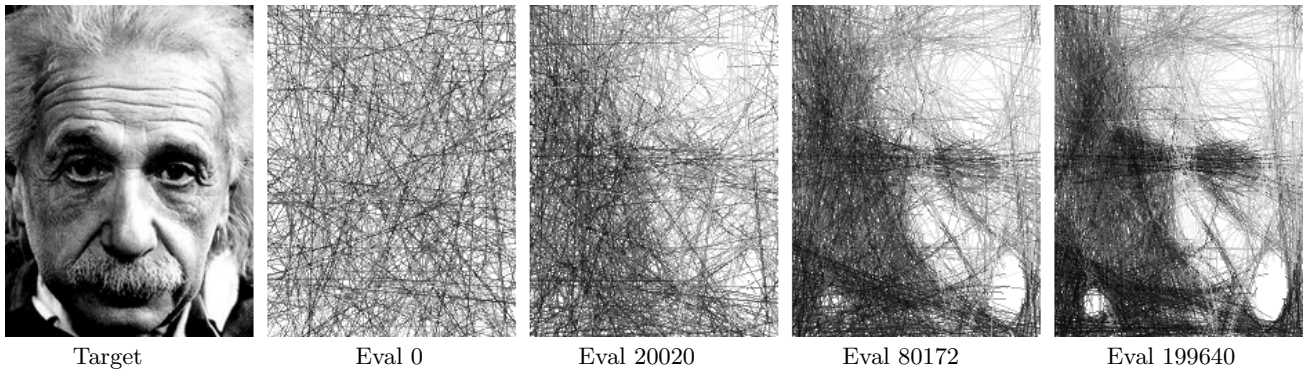


Figure 2: Output of Draw Function #1: Pencil Drawing, Blind Search.



Figure 3: Output of Draw Function #2: Pencil Drawing, Guided Search.

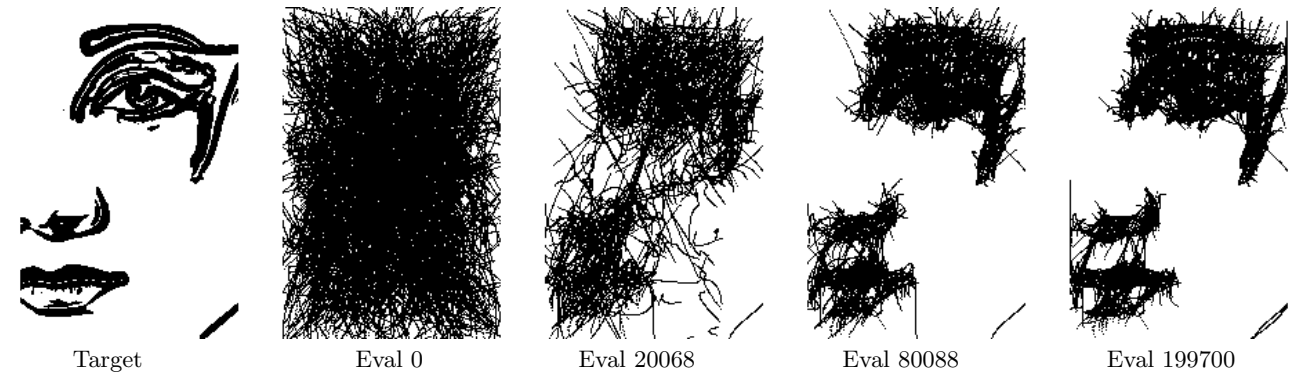


Figure 4: Output of Draw Function #3: Line Art, Monochrome.

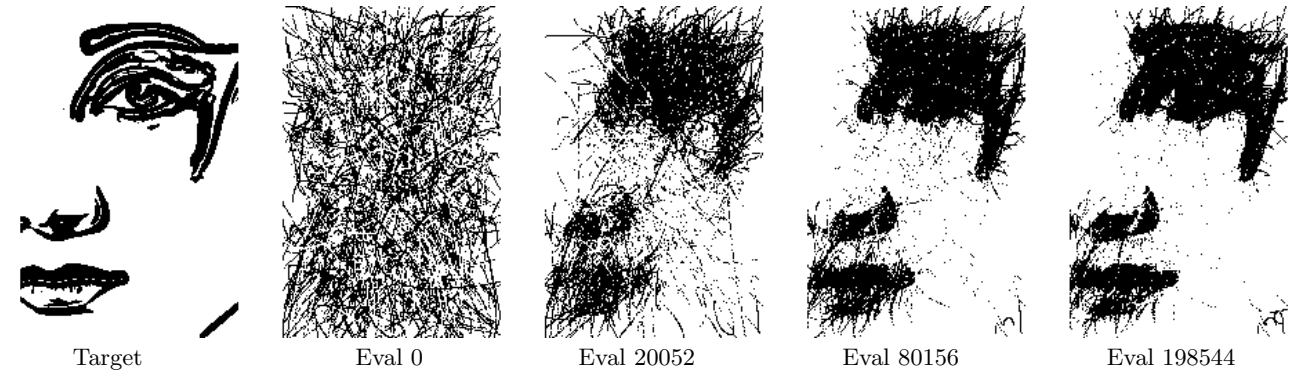


Figure 5: Output of Algorithm #4: Line Art, Two Colours.

3.1.3 Drawing Algorithm #3: Ink, Monochrome

Whereas draw functions #1 and #2 are intended to emulate a pencil drawing effect, this function is intended to simulate an ink style. Instead of drawing a line, a cubic Bezier path is drawn [4]. A Bezier path is a parameterised representation of a curved line. The shape of a Bezier path is controlled by the position of four control points. To plot the points along the path, the control points are substituted into the following formula, where t is a real number $\in \{0, 0.000001, \dots, 1\}$.

$$(1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t) P_2 + t^3 P_3 \quad (1)$$

The four inputs from the *Draw* function are now interpreted differently and represent each of the four control points required to draw a cubic Bezier path. Every pixel is drawn in black onto a white canvas. To aid the algorithm in placing lines, a weighted fitness function is used. The rationale behind the formula is as follows. There are only two types of pixels: black pixels and white pixels. When measuring the fitness of a rendering, there can be only two bad outcomes: drawing a black pixel where there should be white, and vice versa. A characteristic of a black and white illustration is that one colour predominates over the other. A mechanism must be incorporated to ensure that the less dominant desired colour does not dominate the rendering. The target image we have used is predominately white. We have used a weighted fitness function to limit the number of black pixels that are drawn. The weighted difference is calculated at canvas initialisation. It is a real valued number representing the ratio of white pixels in the target image to black pixels. The weighted difference is applied as a punishment factor for black pixels drawn where white is desired.

$$W = \frac{\text{white pixels}}{\text{black pixels}} \quad (2)$$

Figure 4 shows a black and white target and a sequence of evolved images using this draw function. The original grey level image is shown in Figure 6. The ink draw function begins with a messy placement of strokes and iteratively refines the placement of those strokes. The still images do not capture the dynamics of the animation which starts as a mass of curves wriggling all over the canvas. The amount of movement slowly reduces as the animation resolves into a relatively stationary target.

3.1.4 Drawing Function #4: Ink, Two Colours

Draw function #3 is modified to allow strokes to be rendered in either black or white. To facilitate this the GP representation has to be changed to allow Draw functions with 5 terminal inputs instead of the 4 used by the other drawing algorithms. Some evolved individuals are shown in Figure 5. By allowing the drawing of both black and white colours onto a changing canvas, we are emulating the effect of an eraser. Thus the algorithm revolves not only the placement of pencil strokes, but also the shapes of those strokes. The visual result, the dynamics of which are not evident from the sequence of still images, is much more compelling than that in figure 4.



Figure 6: Grayscale target image for Figures 4 and 5.

4. DETERMINING THE GP CONFIGURATION

This section describes the various experiments we performed in order to determine a suitable GP configuration. For each experiment we table the averaged results of 5 runs. Across all experiments the results of individual runs displayed a high degree of similarity. For that reason we limited the number of runs per experiment to 5. Table 1 shows settings that were common to all runs.

4.1 Population Size

Given the nature of our representation, we have some expectations about the behaviour of crossover and mutation. Each sub tree can be regarded as a building block that will render part of the image. The vast majority of nodes in our trees occur at the level of the *Draw* functions – either the functions themselves or their terminal inputs. Consequently the majority of nodes chosen for crossover or mutation will be at this level. Crossover will most often involve the swapping of building blocks. Either a pair of building blocks will swap one input parameter or the swapping of building blocks will result in a slight reordering of pencil strokes. At the lower levels of trees, mutation will have little affect. Either a new pencil stroke will be generated or one input to a pencil stroke will be changed. At higher levels in trees there is more potential for change. In the case of crossover

Table 1: GP Parameters common to all experiments

Parameter	Value
Tree Generation	Ramped Half-and-half
Selection Strategy	Roulette Wheel
Replacement Strategy	Generational Replacement
Termination	Fitness target reached or Max. evaluations reached
Functions	ProgN Draw
Terminals	$frand, \mathbb{R} \in \{0.000, 1.000\}$
Image size	180 × 240
Maximum tree depth	10
Runs per experiment	5

Table 2: GP parameters for population based experiments

Pop Size	Elitism	Cross	Mut	Fitness at 100,000 evals	Fitness at 200,000 evals
10	1	4	5	0.112	0.093
4	1	2	1	0.096	0.081
4	1	0	3	0.098	0.084
2	1	0	1	0.095	0.082

the expected result is that sequences of pencil strokes will be radically different – but only the sequences. The same strokes will be rendered, but in a different order. Mutation at the higher levels, however, has the potential to drastically change the output of a program by generating an entirely new sequence of strokes.

We experimented with a range of population sizes and noticed that smaller populations tended to give faster convergence. This is consistent with the findings in [11]. We therefore decided to conduct experiments using population sizes of 10, 4 and 2. Table 2 shows the elitism, crossover and mutation rates for these experiments, as well as the fitness after 100,000 and 200,000 evaluations. Since the populations are so small, some care needs to be taken in determining these rates. Given that we expect the fittest individual to dominate, we set the elitism rate to a low value. We envisaged populations that would mutate on a regular basis, but still display some slight diversity through crossover. Thus after setting the elitism value we set the mutation rate to 50% and adjusted the crossover rate accordingly. Setting a mutation rate of 50% for the population of 4 would result in a crossover value of 1, which is not feasible. So we conducted experiments on two populations of size 4, one with 25% mutation and one with 75% mutation.

The results of these population sizes with drawing function #1 are shown in Figure 7 and in Table 2. The population sizes of 4 and 2 clearly outperform the population of 10. We now suggest why a smaller population would outperform a larger population in our problem domain. Due to the na-

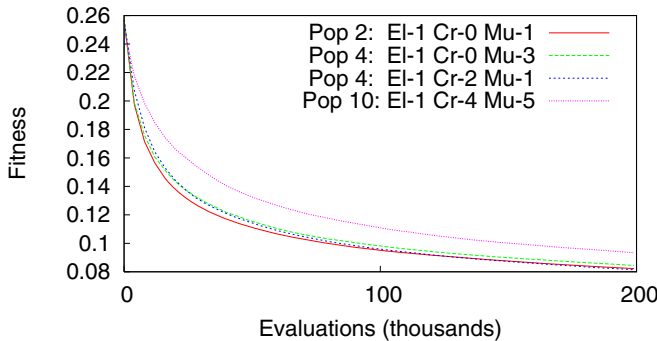


Figure 7: Population Size

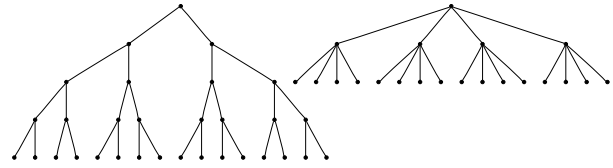


Figure 8: Same drawing sequence represented by trees of arity 2 and 4

ture of our drawing algorithms and the GP representation, in any generation, any improvement in fitness is likely to be small and is mainly independent of population size. The reason for this is that the probability of successfully mutating a superior program is significantly low. A population of 10 is 5 times the size of a population of 2. Thus the population of 2 requires 5 times as many generations in order to provide a fair basis of comparison. Because generational increase in fitness is going to be small regardless of population size, the smaller population has more opportunities to increase in fitness.

4.2 Prog Function Arity

In our representation there is a many-one genotype-phenotype mapping, that is, there are many trees for any particular sequence of *Draw* functions. The left hand tree of Figure 8 shows one representation of a sequence of draw functions. However, there are many other binary trees for the same sequence of terminals. Furthermore, this same sequence of terminals could be represented by the tree of arity 4 shown on the right hand side of Figure 8. While the result of executing both program trees will be the same, there will be significant differences in the evolutionary process. The arity 4 programs will contain larger building blocks. The evolution will proceed well if good building blocks of size 4 strokes can be found. If such building blocks cannot be found, evolutionary progress will be inhibited.

We have experimented with different combinations of *ProgN* function arities. Our experiments consisted of testing 3 sets of *ProgN* types: populations consisting of a) *Prog2*, *Prog3* and *Prog4* functions; b) *Prog5*, *Prog6* and *Prog7* functions; c) *Prog8*, *Prog9* and *Prog10* functions. As *ProgN* function arities increase, the size of the generated programs increases exponentially and so we have decreased the maximum tree depth for the experiments involving higher arity functions to provide a fair basis of comparison with those employing lower arity functions.

None of our experiments with *ProgN* functions with arity higher than 4 were successful – not one run ever increased in fitness during any generation. We propose the following reasons for this. As stated previously the greatest potential for improvement in fitness lies in the possibility of mutation occurring above the level of *Draw* functions. With trees of small depth, the chances of this occurring become remote. Moreover, one of the challenges faced by all *ProgN* functions is that all of their inputs have to perform well at the same time. Higher arity functions incorporate several more inputs than lower arity functions and therefore have a much lower probability of all their inputs performing well simultaneously.

We found that programs consisting of *ProgN* functions of arities 3 and 4 performed best. The problem with *Prog2*

functions was that programs do not generate enough *Draw* functions to render a sufficient number of pixels on a canvas. *Prog3* and *Prog4* functions, however, provide enough *Draw* functions at a tree depth of 10 to produce good drawings.

4.3 Alternate Tree Representations

As noted earlier, a factor in the use of our GP representation is the fact that *ProgN* functions serve only to provide a structural form, but contribute nothing directly to the drawing of an image. We have investigated two alternative tree structures to determine whether higher level nodes in the tree can contribute to achieving good fitness. From the implementation point of view this involves replacing the current *ProgN* and *Draw* with a single new *Draw* function.

```

NDRAW  -> NewDraw1(NDRAW,NDRAW,NDRAW,NDRAW)
NDRAW  -> float

```

NewDraw1 is the same as the previous *Draw* except that it now returns a value and can be an argument in another *NewDraw1* function. This value will then be used by the higher level *NewDraw1* function. However, it is not at all clear what this value should be. We experimented with a number of possibilities including *max*, *min* and *average* of the arguments for drawing function #1. The performance was very poor so we halted experiments on this representation. We conclude that the reason for the poor performance is that *NewDraw1* functions in upper levels of the tree require meaningful data to be passed up to them. We could not find a mechanism to incorporate meaningful passing of data.

NewDraw2 is similar the previous draw in that no value is returned. However, each *NewDraw2* function has, in addition to its floating point arguments, an additional argument which could be another *NewDraw2*:

```

NDRAW  -> NewDraw2(T,T,T,T,NDRAW)
NDRAW  -> NULL
T      -> float

```

The evolved programs consist of trees in which each node in the tree renders a stroke, not just the leaves. This means that the same number of strokes can be represented in trees that are considerably smaller than those described in Section 3. Furthermore, the likelihood of search problems due to the many-one genotype-phenotype mappings should be considerably reduced.

Table 3 shows a comparison of performance between our original draw function and *NewDraw2*. There are no strong trends evident. In terms of fitness, *NewDraw2* is worse for populations of size 10 and 2, and about the same for populations of size 4. In terms of size, *NewDraw2* is better for populations of size 10 and 2, and about the same for populations of size 4. Figure 9 shows fitness graphs of *NewDraw2* for different population sizes. The same trends as those in Figure 7 are evident. There are no compelling reasons to move from our original representation to *NewDraw2*.

5. CONCLUSIONS

We have shown how to use genetic programming to produce engaging animations of black and white drawings. The animations begin as a random collection of strokes and gradually resolve into a recognizable target image. The individual frames of the animation are renderings of the best individual of each generation. The sequence of strokes to

Table 3: Comparison of original *Draw* and *NewDraw2*.

Pop Size	Fitness at 200,000 Draw #1	Program size Draw #1	Fitness at 200,000 <i>NewDraw2</i>	Program size <i>NewDraw2</i>
10	0.093	8500	0.101	4880
4	0.081	7922	0.081	7310
4	0.084	7729	0.080	5650
2	0.082	5613	0.095	3275

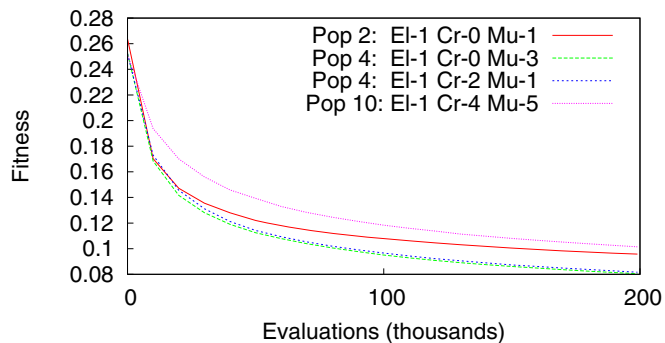


Figure 9: Population size for *NewDraw2*

render the image can be represented as program trees with two kinds of nodes – *Draw* and *ProgN*. Execution of *Draw* nodes places a stroke on the canvas. The *ProgN* nodes passively provide the necessary tree structure. Fitness is a measure of the difference between evolved individual and a target. We have produced interesting animations with two kinds of strokes – grey level straight-line pencil strokes and black ink Bezier curves. A fitness function that summed pixel differences was adequate for the pencil strokes, however the ink strokes required a weighted fitness that strongly penalized placing a black pixel where there should be a white. Surprisingly, small population sizes were best for this domain. A population size of 4 with one new individual being generated by elitism, two by crossover and one by mutation gave the best overall performance. The rate of convergence and the time for the subject to emerge from the initial random sequence of strokes can be controlled by use of information from the target image in drawing a stroke. Changing a pixel only when the updated value is closer to the target value accelerates convergence and gives a better resolution of the final image.

Artists have used our programs to create a wide range of animated artwork. Having control over many different variables means the artist can have creative control over the effects desired for each particular piece. The effects are more varied and interesting than is possible with commercial filters.

In further work we plan to experiment with a wider range of brush strokes, with colour strokes and with different brush shapes.

6. REFERENCES

- [1] Atsushi Kasao and Kazunori Miyata. Algorithmic Painter - a NPR method to generate various styles of painting. *The Visual Computer*, 2005.
- [2] E. Baker and M. Seltzer. Evolving Line Drawings. In *Graphics Interface*, pages 91–91. Canadian Information Processing Society, 1994.
- [3] W. Baxter, J. Wendt, and M. C. Lin. Impasto: A realistic, interactive model for paint. In *NPAR '04: Proceedings Of The 3rd International Symposium On Non-Photorealistic Animation And Rendering*, pages 45–148. ACM, 2004.
- [4] S. R. Buss. *3-D Computer Graphics: A Mathematical Introduction with OpenGL*, pages 163–168. Cambridge University Press, 2003.
- [5] S. Camhy. *Art of the Pencil: A Revolutionary Look at Drawing, Painting, and the Pencil*. Watson-Guptill, 1997.
- [6] U. Chakraborty and H. Kang. Stroke-based rendering by evolutionary algorithm. In *India Annual Conference, 2004. Proceedings of the IEEE INDICON 2004. First*, pages 52–57, 2004.
- [7] N. Chaumont, R. Egli, and C. Adami. Evolving Virtual Creatures and Catapults. *Artificial Life*, 13(2):139–157, 2007.
- [8] N. S.-H. Chu and C.-L. Tai. Moxi: Real-time ink dispersion in absorbent paper. In *SIGGRAPH '05: ACM SIGGRAPH 2005*, 2005.
- [9] V. Ciesielski, M. Berry, K. Trist, and D. D'Souza. Evolution of animated photomosaics. In M. Giacobini, editor, *Applications of Evolutionary Computing: Proceedings of the 5th European Workshop on Evolutionary Music and Art (EVOMUSART07)*, volume 4448 of *LNCS*, pages 498–507. Springer, Apr. 2007.
- [10] J. Collomosse and P. Hall. Genetic paint: A search for salient paintings. In *Applications of Evolutionary Computing, EvoWorkshops*, pages 437–447. Springer, 2005.
- [11] C. Gathercole and P. Ross. Small populations over many generations can beat large populations over few generations in genetic programming. In J. R. K. et al., editor, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 111–118, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.
- [12] Gayan Wijesinghe and Shahrul Badariah Mat Sah and Vic Ciesielski. Grid vs. Arbitrary Placement of Tiles for Generating Animated Photomosaics. In *2008 World Congress on Computational Intelligence*, 2008.
- [13] Gershon Elber and Elaine Cohen. Probabilistic silhouette based importance toward line-art non-photorealistic rendering. *The Visual Computer*, 2006.
- [14] D. E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(5):493–530, 1989.
- [15] P. Haeberli. Paint by numbers: Abstract image representations. *Computer Graphics*, 24(4):207–214, 1990.
- [16] D. Hart. Toward Greater Artistic Control for Interactive Evolution of Images and Animation. *Lecture Notes In Computer Science*, 4448:527, 2007.
- [17] A. Hertzmann. A survey of stroke-based rendering. *Computer Graphics and Applications*, pages 70–81, 2003.
- [18] Karl Sims. Artificial evolution for computer graphics. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, 1991. <http://www.karlsims.com/papers/siggraph91.html>.
- [19] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, Mass., 1992.
- [20] M. Lewis. Evolutionary visual art and design. In Romero and Machado [25], pages 3–37.
- [21] P. Machado and A. Cardoso. All the Truth about NEvaR. *Applied Intelligence*, 16(2):101–118, 2002.
- [22] J. McCormack. Interactive evolution of L-system grammars for computer graphics modelling. In D. G. Green and T. Bossomaier, editors, *Complex Systems: from Biology to Computation*, pages 118–130. IOS Press, 1993.
- [23] J. McCormack. Turbulence: An interactive installation exploring artificial life. In *Visual Proceedings: The Art and Interdisciplinary Programs of SIGGRAPH 94*, In Computer Graphics Annual Conference Series, ACM SIGGRAPH, pages 182–183. <http://www.csse.monash.edu.au/~jonmc/projects/turbulence.html>, 1994.
- [24] C. Neufeld, B. Ross, and W. Ralph. The evolution of artistic filters. In Romero and Machado [25], pages 335–356.
- [25] J. Romero and P. Machado, editors. *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*. Natural Computing Series. Springer, 2008.
- [26] M. Sousa and J. Buchanan. Computer-Generated Pencil Drawing. In *Western Computer Graphics Symposium*, volume 1, 1999.
- [27] Suguru Saito and Akane Kani and Youngha Chang and Masayuki Nakajima. Curvature-based stroke rendering. *The Visual Computer*, 2008.
- [28] T. Unemi. SBART 2.4: breeding 2D CG images and movies and creating a type of collage. In *Knowledge-Based Intelligent Information Engineering Systems, 1999. Third International Conference*, pages 288–291, 1999.
- [29] G. Winkenbach and D. Salesin. Computer-generated pen-and-ink illustration. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 91–100. ACM New York, NY, USA, 1994.
- [30] Yann Semet and Una-May O'Reilly and Fredo Durand. An Interactive Artificial Ant Approach To Non-photorealistic Rendering. *Lecture Notes In Computer Science: GECCO 2004*, 2004.