Brock University

Department of Computer Science

Automatic and Interactive Evolution of Vector Graphics Images with Genetic Algorithms

Steven Bergen and Brian J. Ross
Technical Report # CS-11-01
January 2011

Brock University
Department of Computer Science
St. Catharines, Ontario
Canada L2S 3A1
www.cosc.brocku.ca

Automatic and Interactive Evolution of Vector Graphics Images with Genetic Algorithms

Steven Bergen · Brian J. Ross

Abstract Vector graphics images are composed of lists of discrete geometric shapes, such as circles, squares, and lines. Vector graphics is popular in illustration and graphic design. The generation of vector images by evolutionary computation techniques, however, has been given little attention. This paper uses genetic algorithms to evolve vector images. By restricting the numbers of primitives and colour schemes used, stylized interpretations of target images are produced. Automatic evolution involves measuring the pixel-by-pixel colour distance between a candidate and target image. The JNetic evolutionary vector graphics system is described. JNetic supports automatic and user-guided evolution, chromosome editing, and high-detail masks. The user can paint masks over areas of the target image, which will be used to reproduce the high-detail features within those areas. The system has been successfully used by the authors as a creative tool.

Supported by NSERC USRA and NSERC Operating Grant 138467.

S. Bergen
Department of Computer Science
Brock University
500 Glenridge Ave.
St. Catharines, ON L2S 3A1
Canada
Tel.: 905-688-5550
E-mail: sb04qv@brocku.ca

B.J. Ross
Department of Computer Science
Brock University
500 Glenridge Ave.
St. Catharines, ON L2S 3A1
Canada
Tel.: 905-688-5550
E-mail: bross@brocku.ca

Keywords Genetic algorithm · Vector graphics · Evolutionary art

1 Introduction

Genetic algorithms are widely used for finding good quality solutions for a diverse variety of difficult problems in science, engineering and mathematics [16,20]. They have also been used in artistic applications such as music [7], and visual art and design [3,27]. Their application in image generation is well recognized, and it has resulted in a new genre of art [9,36].

Richard Dawkins is a pioneer in using genetic algorithms for image generation [8]. His original motivation was to show the great variety of graphical structures that can arise from evolution. A computer program, Biomorphs, is described, which lets the user evaluate a population of 2-dimensional images, and thereby act as a force of natural selection for breeding new generations of images from those selected. Although not intended as an artistic application, it showed the potential power of artificial evolution as an exploration tool for graphic design.

A procedural texture is an image generated by program code and mathematical formulae [11]. Sims used a genetic algorithm to manipulate the texture generating equations [30]. The result was a number of intriguing and complex images. These images were mathematical and abstract in style, due to their mathematical origins. Sims work was very influential, and others have extended his approach by considering new forms of mathematical expressions and evolutionary processes. A sample of examples of other approaches includes [17, 28,10,18,1]. All the images in these works abound with complex highly detailed patterns, noise, and colour gradients (Figure 1). Sometimes they simulate textures

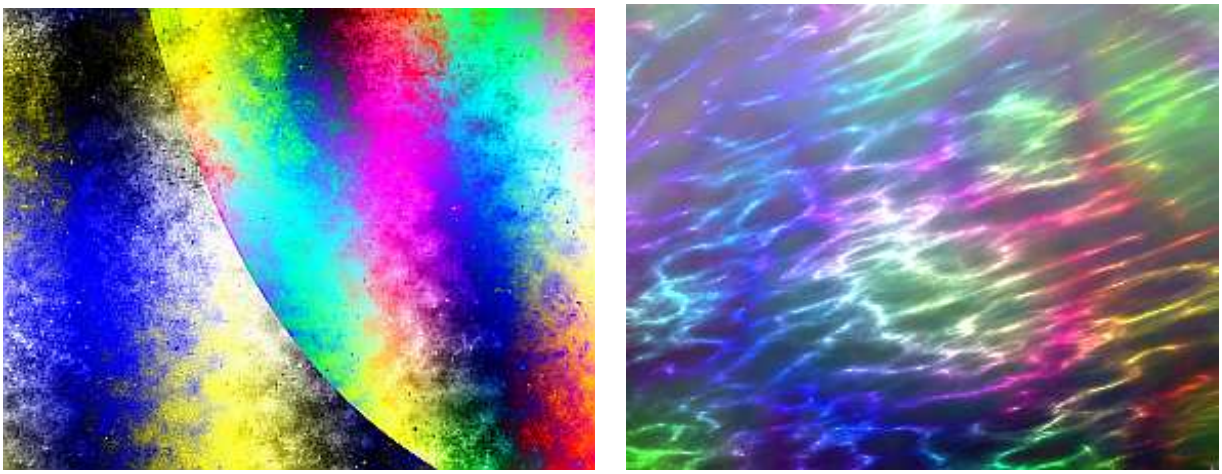


Fig. 1 Images made from procedural texture expressions.

found in nature, such as minerals, clouds, and wood grains.

Like the Dawkins system, Sims and many others utilize interactive genetic algorithms, in which a human user inspects all candidate images in the population, and manually scores them during the course of evolution. The main reason for user interactivity is that there is often not an automated means for the genetic algorithm to evaluate images, based on aesthetic or other artistic criteria. Hence, such systems can be considered as semi-automated exploration tools with which a user can use to navigate a complex universe of computer-generated images [9].

More recent research has started to consider the automatic evolution of images. The challenge for automatic image generation is to find appropriate fitness criteria. In some applications, the goal is to have a genetic algorithm reproduce or closely match some supplied target image. In such cases, the fitness function can be a distance measurement in RGB colour space between candidate images and the target image. Ibrahim[21] and Wiens and Ross[37] evolve procedurally generated images that match a target image. Matches are determined using a set of image analyses tests. Research is also exploring the use of mathematical models of aesthetics in genetic algorithms [19,25,27,32,29,26]. Aesthetic modeling is still in its formative stages in this discipline.

Genetic algorithms have been applied to more representational styles of art and design, that are not based on mathematical textures. For example, Todd and Latham use evolutionary computation to evolve images of geometric structures with organic characteristics [34]. Lewis uses an interactive GA for evolving different styles of cartoon faces [24]. Realistic faces are reconstructed with genetic algorithms, which might have use in crime investigations [14]. Mondrian and Escher-style art is generated with a genetic algorithm [12]. Architectural structures are evolved with genetic algorithms [22].

Another important class of computer imagery is vector graphics. Vector graphics are fundamentally different from procedural textures in style and form. Vector images are geometric in nature, since they are composed of discrete geometric shapes, such as lines, circles, and polygons. Vector images are commonly used in the world of graphic design and illustration. Not only are vector graphics desireable from an aesthetic perspective, but they are also popular from a pragmatic point of view. Vector graphics permits straight-forward editing of images, by manipulating collections of composite shapes. Examples of vector graphic art can be viewed at [40].

Unlike procedural textures, vector graphics images have not been extensively studied in the evolutionary art field, and few examples of research exists on the topic. Weller introduced the idea of automatic evolution of vector graphics images [35]. His goal was to compress a target image by finding a set of geometric primitives which, when rendered, result in an approximation of the original image. The resulting images lose the high-resolution detail of the originals, and are too coarse of reproductions to be practical for image compression. On the other hand, the images have a certain stylistic appeal.

Wilkens extended Weller's work, by performing multiple phases of evolution, in an attempt to capture high-resolution details from the target image [39]. Each phase involves reducing the size of geometric primitives, in order to fine tune the colour distance between the rendered image and the target. Although more detail is

retained in Wilkens' results, most high-resolution detail is nevertheless discarded.

This paper re-examines the use of genetic algorithms to generate vector graphics images. As done by Weller [35] and Wilkens [39], we use a genetic algorithm to render an image from geometric primitives. The primitives may have different sizes, shapes, and colours. The goal of evolution is to find a set of primitives whose final rendering closely matches a target image, as determined by pixel-by-pixel colour matching. The intention is not to reproduce the target image precisely, but rather, use it as a visual inspiration or metaphor for a new, geometric approximation of the image. We are interested in this final, approximate image from an aesthetic, artistic point of view. Therefore, this research differs in intent and design from most evolutionary computation research in the literature. We are not attempting to find optimized solutions, since they would simply be exact reproductions of the target image. Rather, errors and inexactness are desirable qualities for our solutions, as they make the resulting images creative and artistically interesting.

We have a number of motivations and goals for our research. A problem seen in earlier approaches was the loss of high-resolution detail in evolved vector images [35,39]. We address this issue by introducing user-defined image masks. With masks, areas of an image can be identified that require more precise details in the final vector result. For example, features in a face can be masked for high-resolution processing. Additional evolutionary effort generate more refined vector renderings for these identified areas.

Another goal is to implement a system which supports both automatic evolution and user-guided interactive evolution. Our system, *JNetic*, is a user-friendly, standalone tool for evolutionary vector-based art. JNetic lets the user freely move between interactive and automatic evolution as desired. This overcomes the weaknesses of each. For example, with fully automatic evolution, after the user sets initial parameters, he or she is afforded no influence during evolution, and is relegated to a passive role only. The user must either accept or reject the final result. On the other hand, strictly interactive evo-art systems suffer from user fatigue. The user must micro-manage every aspect of evolution, by rating each individual in the population. This does not usually allow highly refined results.

Our system also presents a complete environment for vecto graphics evolution. A rich variety of options are available to choose from, such as a number of primitive shapes, the mixing of different primitives together, and various colour rendering options. The user may even edit chromosomes, to correct images as desired. Final images may be saved in the common "SVG" vector graphic format, to be edited in other vector graphics applications.

The paper is organized as follows. Genetic algorithms are briefly reviewed in Section 2. Section 3 discusses the technical features of JNetic. Some of the interfaces of JNetic are presented in Section 4. A selection of evolved images is presented in Section 5. The use of JNetic in a creative context is discussed in Section 6. Section 7 gives concluding remarks and comparisons with related work.[1]

2 Genetic Algorithms

```
A. Initialize random population of N chromosomes.
B. Loop from 1 to final generation:
   a. Calculate fitnesses of all chromosomes.
   b. Loop until N child chromosomes:
      1. Select 2 parent chromosomes from population.
      2. Apply crossover, generating two offspring.
      3. Apply mutation to both children.
      4. Add children to new population.
   c. Replace old population with new population.
   d. Apply elitism (if enabled).
```

Fig. 2 Genetic Algorithm

A genetic algorithm is a search procedure inspired by Darwinian evolution [16,20]. The idea is that a population of individuals (chromosomes, candidate solutions) is continually processed and refined, until eventually an acceptable solution is obtained. The refinement process is motivated by ideas from natural evolution.

Figure 2 outlines a basic genetic algorithm. The process begins in step (A) by creating a population of size N of random individuals or chromosomes. Each chromosome represents a complete candidate solution for the problem at hand. For example, in the case of JNetic, a chromosome represents a complete image. For most non-trivial problems, this random population will not contain anything resembling an acceptable solution. The genetic algorithm will proceed to refine this population in the loop at step (B). Each iteration through loop (B) results in a new population or generation of individuals. First, each chromosome in the population is evaluated and scored, either via an algorithmic formula, or via a human user assigning a score interactively. Next, a new population is creating in loop (b). Step (i) uses a fitness-based scheme to select two chromosomes from the population. Fitness-based selection

[1] The JNetic system, and a gallery of images, are available online at *http://www.cosc.brocku.ca/ bross/JNetic/*.

implies that the previously assigned fitness scores in step (a) are used to determine strong individual individuals to use for subsequent reproduction. This parallels Darwinian survival and reproduction of the fittest in nature. Once two parent chromosomes are selected, a crossover or recombination operation is applied to the chromosomes (described below). This results in two offspring with features that are inherited from each parent. A mutation operator is then applied, which involves changing a chromosome value at random. The offspring are then inserted into the next population, which will eventually replace the current population in step (c). Finally, an optional elitism. step can be performed (step d), which takes the strongest chromosome(s) in the current population, and adds it to the new population. This step ensures that the strongest individual will not be lost between generations. This entire process continues until the final generation specified by the user is reached in the B loop. The fittest chromosomes in the population are considered solutions as found by the genetic algorithm during this session.

Fitness-based selection involves using a procedure for choosing chromosomes for reproduction based on their strength or quality, as reflected by their fitness scores. As in natural evolution, fitter individuals are more likely to survive and reproduce. However, it is beneficial for genetic diversity - and overall solution quality in the long term - to not simply select the absolute strongest chromosomes in the population, but to also give weaker individuals a chance at reproduction. One means of implementing fitness-based selection is *tournament selection*. A tournament size K is specified, with K typically ranging between 2 and 7. Then K individuals are randomly chosen from the population. The individual with the strongest fitness from this set is designated the tournament winner, and is selected for reproduction. For crossover, a separate tournament is performed for each parent. When K is lower (for example,2), then there is a higher probability of selecting weaker individuals. Conversely, a high K value will mean that stronger individuals are usually selected, and we say that there is a higher selection pressure.

Crossover is the most important reproduction operator used by the genetic algorithm. First, two parent chromosomes are selected for crossover. For one-point crossover, a single random splice point is determined, and each parent contributes one portion of its chromosome during reproduction. The spliced chromosomes are designated the offspring, and inherit genes (characteristics) from each parent.

The other reproduction operator is mutation. It involves randomly altering a gene value. If the chromosome is comprised of a list of numbers, then a mutation operation may involve selecting one gene and replacing it with a new random value. In this way, mutation permits new gene values to arise in chromosomes that might be lost if only crossover were used.

Note that there is a high degree of randomness involved in the genetic algorithm. The initial population is generated randomly. Parent chromosomes are selected for reproduction using randomized selection procedures. Crossover points and mutations occur at random positions. The effect is that different executions of a genetic algorithm usually result in quite varied results. Therefore, it is worth running a genetic algorithm multiple times on a problem, and examining the results obtained from each, to determine the most appropriate solution.

3 JNetic

3.1 Image Representation

A chromosome is a list of numbers which represent geometric primitives comprising a vectorized image. A rendering procedure will traverse the chromosome, and translate fields of genes on the chromosome into corresponding geometric shapes on a canvas. Therefore, each chromosome contains the information required to render a complete image, and a population of 100 chromosomes represents 100 individual images.

Figure 3 shows how a chromosome is rendered to a graphics image. Three circles are represented in the chromosome. Each circle has a 2D coordinate for its center, as well as a radius dimension. If RGB colour mode is used, then the colour (red, green, blue) is encoded. Each circle is then rendered in the order given in the chromosome, from left to right. This ordering will affect how circles obstruct each other, as later objects are drawn on top of previously rendered ones.

JNetic primitives include circles, rectangles, N-point polygons, lines, and spline (curved) paint strokes. Each geometric shape can be constrained with respect to its size. For example, circles can have minimum and maximum radii specified, rectangles can have height and width restrictions, lines have width ranges, and N-point polygons and splines can have a maximum N (number of vertices). The maximum number of shapes allowed in the chromosome is specified by the user. Combinations of different shapes are also possible.

3.2 Colour Representation

JNetic supports two kinds of colour schemes: RGB mode and colour palette mode. RGB mode is shown in Fig-

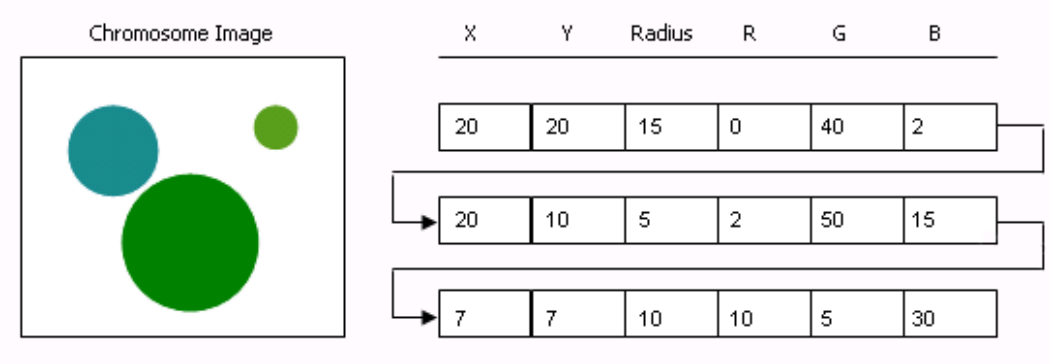


Fig. 3 Chromosome representation

ure 3. Each primitive has its RGB colour directly encoded within it. An advantage of this scheme is that it permits the maximum range of colours to be selectable within an image. A disadvantage is that it takes additional effort for the genetic algorithm to discover the RGB colours that match with the target image. Colour palette mode uses a limited number of colours, predefined by the user. For example, primary, tertiary colours, and grey scale shades might be chosen. Note that limiting the colour palette is one way to generate colour-constrained abstractions of the target image.

A feature possible in colour palette mode is colour quantization [6]. This automatically generates palette colours, based on the colours resident in the target image. This helps the genetic algorithm, as the search for matching colours becomes greatly simplified. JNetic uses octree quantization [15]. This algorithm generates K approximations to the most frequent colours, and colours that are not used are merged with other less-frequent colours close to them in colour space.

Alpha transparency is also supported. When used, colours will mix with previously rendered colours on the canvas. This extends the range of colours used in the colour palette, by permitting additional shades to arise during rendering. It also results in a transparency effect, in which objects can appear translucent.

3.3 Image evaluation

During automatic evaluation, JNetic compares each rendered vector image to the target image. The idealized goal is to render an exact pixel-by-pixel replica of the target image. Given that the vector image is composed of a limited number of geometric shapes, possibly rendered in a finite colour palette, this will be impossible to realize. Of course, exactly reproducing the target image is not actually desirable when stylized images are wanted.

Let (R_p, G_p, B_p) be the red, green, and blue channels of a pixel p. The formula for measuring similarity between vector images and the target image is:

$$Distance = \sum_i \sqrt{(R_{v_i} - R_{t_i})^2 + (G_{v_i} - G_{t_i})^2 + (B_{v_i} - B_{t_i})^2}$$

where i ranges over all pixels in the vector image (v_i) and target image (t_i). An exact match yields a distance of zero, and higher values represent images further away from the target. This is used by the genetic algorithm to score images in the population.

The colour distance calculation is the most time consuming computation occurring in the genetic algorithm. Larger images incur a heavy time penalty during this calculation. Consequently, the user can specify an optional gap or number of pixels to skip when computing colour distances. For example, a gap of 2 means that every other pixel is used, which reduces the total calculation by a half. Small gaps have a negligible effect on final results. Larger gaps may result in less accurate results.

3.4 High-detail Masks

The distance formula in Section 3.3 evaluates all portions of an image equally. This results in an area-based evaluation, in which larger areas contribute proportionally more to fitness than smaller areas. We may want to see details in images, however, such as facial features in portraits. Unfortunately, if the facial features represent a small area of the image, their contribution to the distance calculation will be inconsequential. Such features will be lost in final results.

To promote the rendering of high-detailed areas of an image, the user can paint a *mask* over high-detail areas (Figure 4). The entire image is then called the *base image*, and the masked portion is the *mask image*. Likewise, the chromosome is split into base and mask

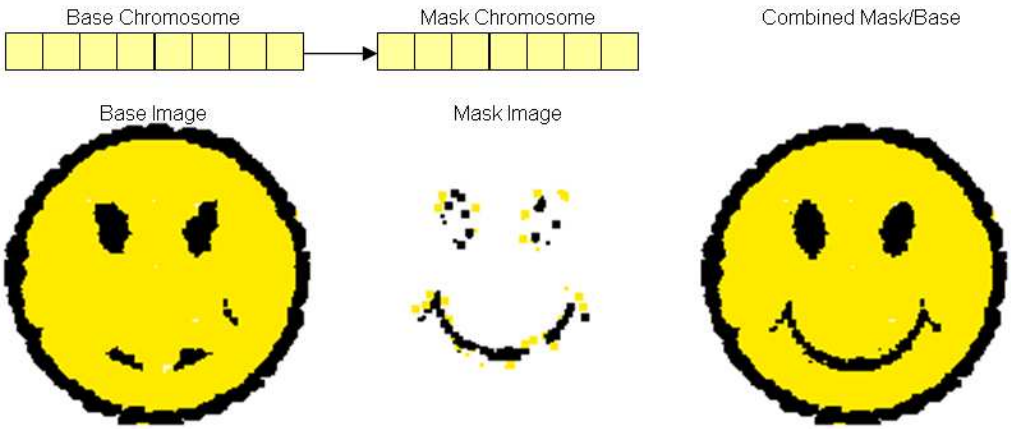


Fig. 4 Mask processing

portions. Both chromosome sections use the same representation as described in Section 3.1. They can be set to use different primitives and colour models, as they are never combined with each other during reproduction. Before fitness evaluation, the base image is first rendered, and then the mask image is rendered on top of it. The mask primitives are rendered within the mask area only; portions of primitives outside the mask area are erased. The entire image is then evaluated for colour distance as usual. Therefore, the mask chromosome ensures that its encoded primitives are dedicated to rendering the mask portion of an image. The mask is effective for capturing high-detail portions of an image if the mask area is not too large, and a sufficient number of primitives are allocated to the mask chromosome.

3.5 Genetic Algorithm

As is usually the case with all applications of evolutionary computation, setting JNetic's genetic algorithm parameters is more of an art than a science. The user can set the usual GA parameters, such as population size, maximum generation limit, reproduction operation probabilities, and tournament size, among others. Either one-point, two-point, or N-point crossover operators can be used. Mutation is also included, and can be controlled with a mutation range value, to prevent extreme changes in gene values. Elitism – the preservation of the best performer in each generation – is also possible.

It is likely that trial runs will be desired, to see how well GA parameters work with the chosen chromosome representation, colour palette, and target image. Decisions are ultimately based upon the artistic sensibilities of the user, and what she or he desires as an outcome of the evolutionary process. Since JNetic is intended to be used as an exploratory tool, the user is recommended to experiment freely with parameters, and to be unconcerned about finding a mythical "ideal" combination of settings. Unlike the kinds of optimization problems in which GA's are typically used, it is desirable for the generated solution to be approximate, to have omissions, and to show errors. This makes images interesting. Spending too much effort fine-tuning GA parameters is ultimately counterproductive to the creative process.

Of course, a run can be interrupted at any time. Many parameters can be altered mid-course in a run, and then the run can be resumed. The user might decide to use the system interactively, and manually evaluate members of the population. The ability to move back and forth between automatic and interactive evolution makes JNetic a practical tool for artistic expression.

3.6 Implementation

JNetic is implemented in Java. The name "JNetic" pays homage to Java-based libraries and systems. Java is selected due to its transportability among different computers and operating systems, its programmer-friendly environment, its relatively good performance, and the availability of high-level libraries. JNetic renders images with the Java2D rendering library [23]. Most of the geometric primitives, as well as colour models, are implemented with this library. Elements of the user interface (Section 4), such as windows, text, graphics, and other requirements, are implemented in Java Swing [13]. Finally, Netbeans is used to implement graphical user interface elements [5].

Actual performance statistics depend upon the complexity and size of an image being processed, the size of the population, and the power of the computer hard-

ware. We find that completely automatic runs typically take anywhere between one to six hours, and sometimes longer, depending on the level of precision desired in the final result.

4 User Interface

An important motivation behind the design of JNetic is that it should be a tool that encourages the arbitrary switching between total automated search by the genetic algorithm and interactive evolution with the user in control. Figure 5 shows JNetic's primary interface. Basic genetic algorithm parameters such as total number of generations, population size, reproduction rates, and tournament size, are set in this window. The genetic algorithm can be started, paused, resumed, and halted. Other controls create dialogues that allow the setting of primitives, colour models, mask images, and other parameters. Thumbnails of the top scoring individuals are displayed. The entire population can be inspected if desired.

When in interactive evolution mode, the user overrides the automated fitness assignments by manually selecting images for further reproduction. When doing so, selected images are assigned a perfect score, which makes it likely they will be used by the tournament selection. If automatic evolution is resumed, the automated fitness assignments will be used.

By pausing the genetic algorithm, and selecting a thumbnail image, the chromosome editing dialog in Figure 6 appears. This interface lets the user select any primitive in the base or mask chromosome. A selected primitive is highlighted, and any of its characteristics (size, location, and colour) can be modified. In this way, undesirable portions of an image can be manually corrected as desired. The image under revision can also be saved, for external editing.

A useful feature of the chromosome editor is the *hill climber*. This does a local search optimization on a chromosome, by performing repeated mutations on it. Every time a mutation results in an improved fitness, that mutation is retained. Otherwise, the mutation is ignored. This repeats until the user stops the hill climber. This utility can be useful for refining an image at the end of a run. Alternatively, if evolution is having problems creating a good result during some point in a session, hill climbing can be used to boost the fitness of selected individuals.

5 Examples

Figure 7 shows the effect of using masks and colour quantization. The target image is shown in (a). The result in (c) is from a run using circles, and with direct RGB colour encoding with alpha transparency. In comparison, the result in (d) arises when a fixed-size colour palette of quantized colours (with transparency) is used. Note how this run was able to more accurately render the image. Next, a mask in (b) was painted on the facial features. The result in (e) shows that the facial features are more visible than without the mask. The last image in (f) illustrates a creative use of masking. The background behind Mona Lisa has a mask defined over it. The base chromosome uses lines, while the mask uses circles. The result is the use of two distinct primitives for Mona Lisa and the background.

Figure 8 compares how high-resolution details are retained in both Wilkens [39] and JNetic. The Wilkens and JNetic runs share many of the same parameters, for example, population size of 200, 90% crossover rate, 2.5% mutation rate, tournament size 3, and RGB-mode colour. Wilkens attempts to capture high-resolution detail by using 3 separate phases of evolution. The solutions from phase 1 and 3 are shown. Phase 1 uses 30 size-constrained triangles. The resulting image from phase 1 becomes the background image in phase 2, upon which 30 new half-size triangles are rendered. Phase 3 repeats this with 30 even smaller triangles. The JNetic run uses 60 triangles for the base (non-masked) image, and 60 for the masked portion on Mona Lisa's face. In a single GA run, JNetic renders more high-resolution facial details than Wilkens' final result.

The fitness performance of the JNetic run in Figure 8 is shown in Figure 9. Most progress occurs during the first 200 generations, and the majority of image refinement has occurred by generation 500. The close proximity of the best, worst, and average fitness curves shows that the population has tightly converged throughout the run. It is possible that an increase of the mutation rate could help diversify the population.

Figure 10 illustrates the progress of evolution during a single run. The images show how improvements are more pronounced in earlier generations of a run, while later generations see more incremental changes that hone into the target image more precisely.

Figure 11 shows some of the stylistic variations possible with different chromosome specifications. Lines are used in (b), circles and rectangles in (e), and dispersed small circles in (f). The images in (c) and (d) are the results of applying a Gaussian blur (radius 6 pixels) to the images in (a) and (b) respectively. The blurring approximates the information "seen" by the genetic al-

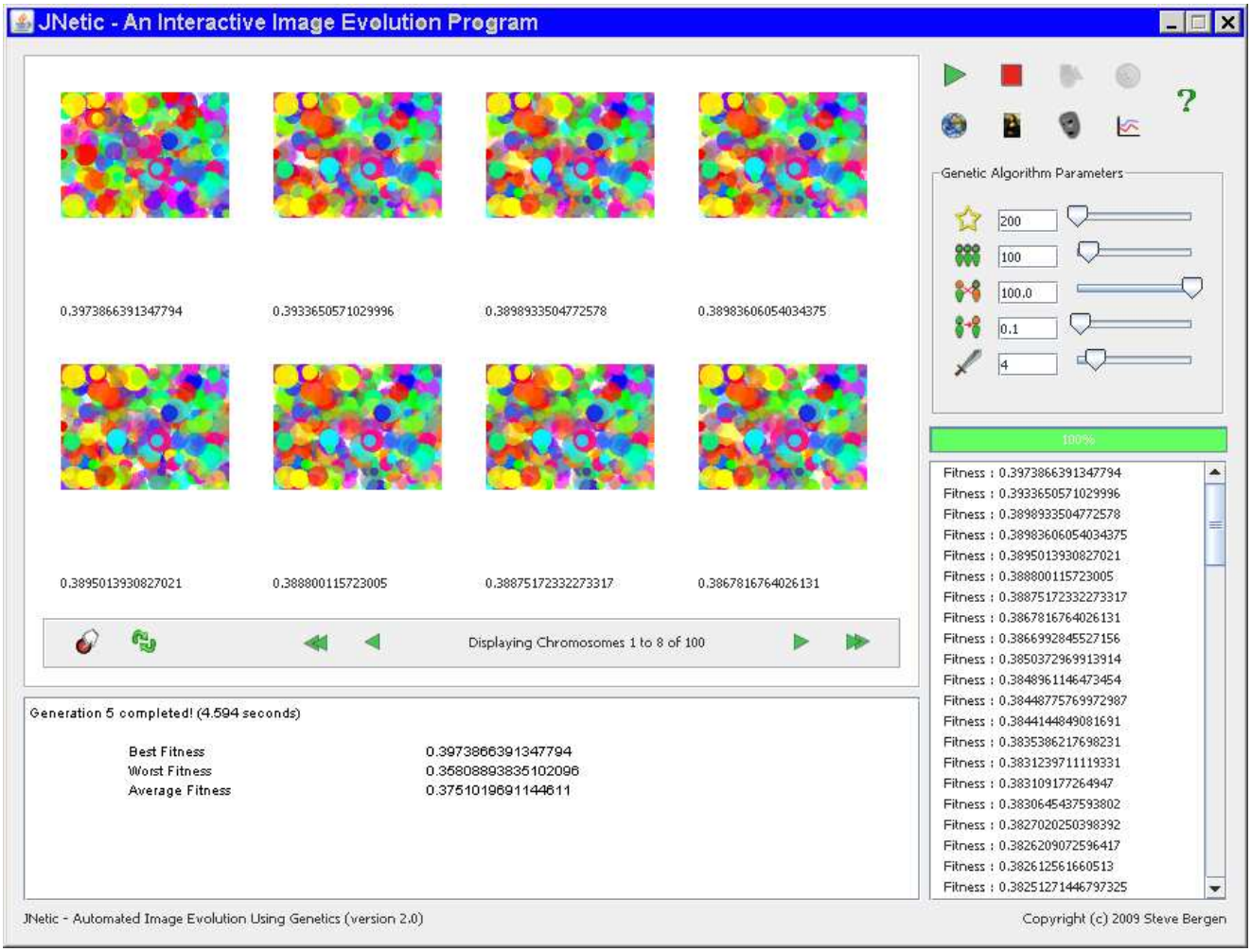


Fig. 5 Main JNetic interface

gorithm, as denoted by the fitness scores. These represent the main colour similarities of the images, when high-resolution detail is lost.

The two images in Figure 12 show the diverse styles of results possible with JNetic, depending on the primitives and colour palette used. *Painting* uses curved (spline) paint strokes and opaque colours with alpha-blending turned off. This results in an image that appears as if painted in thick acrylic paint. This is still a vector-graphics image, however. *Portrait* is made with a set of rectangles and circles, with alpha-blending turned on. A selected option on the final output is "primitive stroking", which outlines the primitives in black. This gives the final image a futuristic, cybernetic feel.

6 JNetic as a Tool for Creativity and Expression

6.1 General Observations

Whitelaw addresses the relationship between evolutionary art and artistic creativity [36]. He makes the point that "evolved artwork itself functions in an unconventional way". The evolutionary process is not highly manageable by an artist, as it is profoundly directed by forces outside the artist's direct influence. As is apparent in the review of genetic algorithms in Section 2, evolution relies on probability and randomness, and controlling such forces of nature is difficult. More importantly, evolution breeds solutions based on criteria external to an artist's normal thought processes. This is the main strength of evolutionary art: the ability to generate unexpected and innovative images selected from a virtually infinite number of possibilities, using criteria outside the artist's mindset. This being said, the

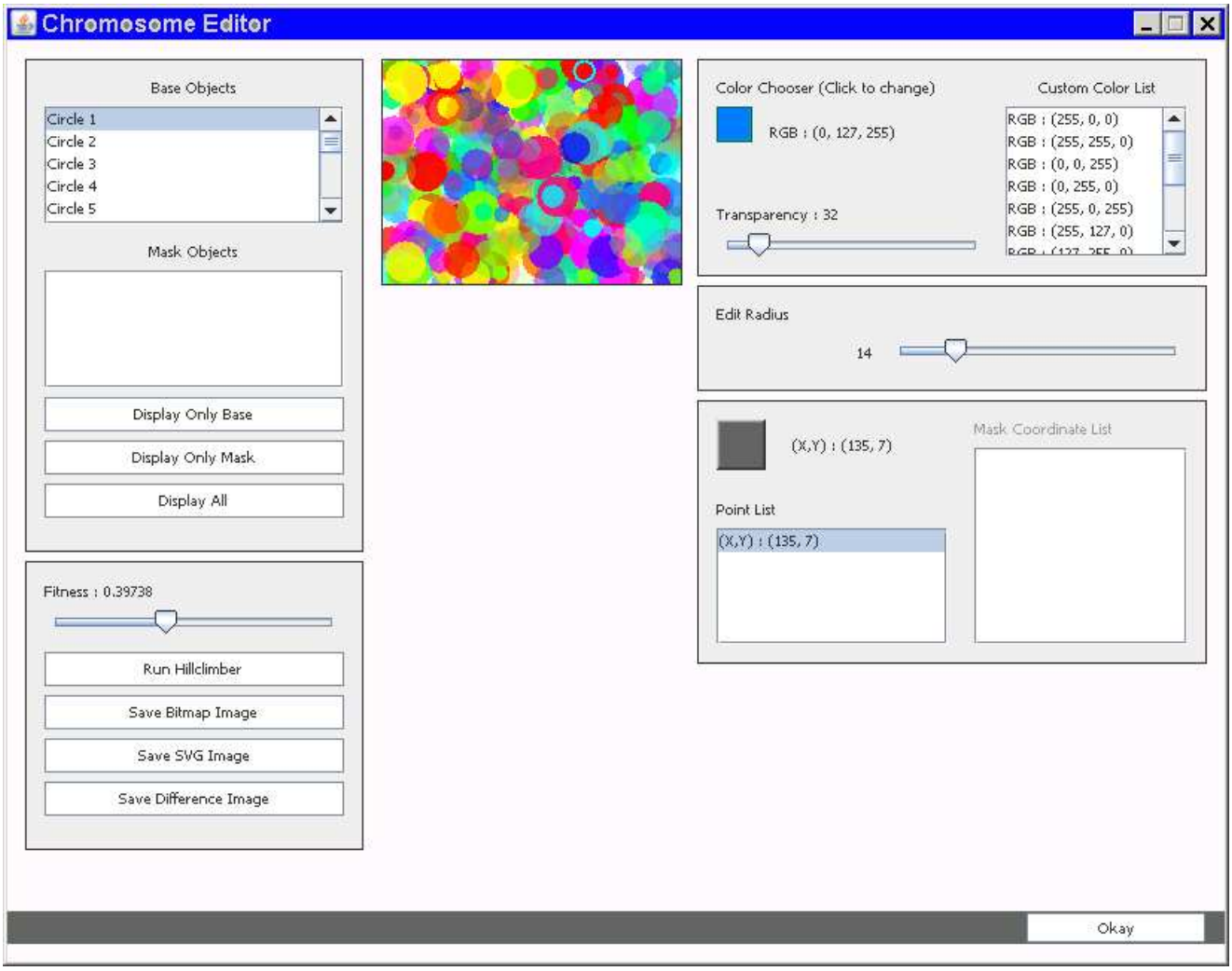
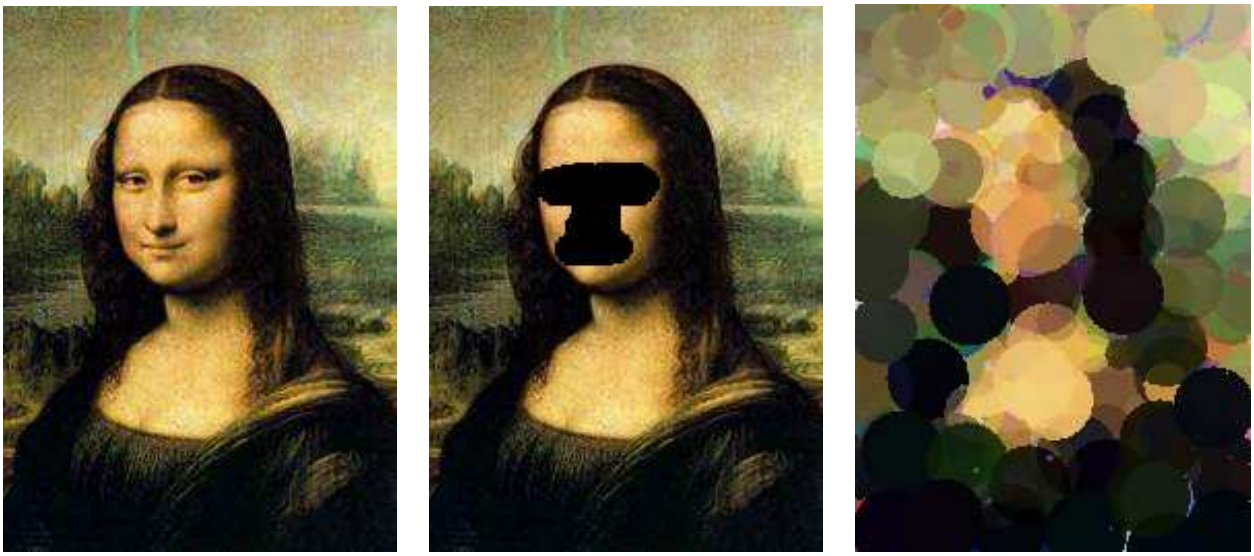


Fig. 6 Chromosome editor

artist should not surrender to evolution, and relinquish complete creative control. Rather, one must consider evolution to be a force that is to be tamed, directed and partnered with.

The motivation behind the design of JNetic is to nurture and promote the relationship between the artist and evolutionary discovery. Our experience is that creativity is well-served in evolutionary art by exploiting the free interplay of automated and interactive evolution. Consider the extreme cases. Purely interactive genetic algorithms depend upon the user at all stages of evolution, which usually results in user exhaustion or boredom after manually inspecting and rating hundreds of images. On the other hand, as we found in earlier research relying solely on automated evolution can sometimes be discouraging, as one must either accept or reject the end result, possibly after many hours or days of processing [29,26]. By permitting the user to interrupt evolution at any time, and interactively influence evolution, the user is afforded the ability to engage in with the evolutionary process. JNetic even permits the user to edit images (chromosomes) during evolution, and in a sense, genetically engineer evolved art at its molecular level.

When a user decides to engage in interactive evolution with JNetic, the problem of user-fatigue can arise, as with other interactive evo-art applications. Should the user decide to manually rate the entire population (which might be in the hundreds or thousands), user fatigue will be guaranteed. Our experience is that automatic evolution is used as the main evolutionary process during the vast majority of time, while interactive evolution may be taken one or two times at most during a run. When interactivity occurs, it usually involves making some fast and simple changes to the run: assigning a few chromosomes very high or very low scores; editing a particular geometric shape, by resizing or moving it, or even deleting it; or setting a new GA parameter,



(a) Source (b) Mask (c) RGB

(a) Quantized (b) Quantized mask (c) Quantized mask v.2

Fig. 7 Examples of source, mask, and results.

such as increasing the mutation rate. The local-search "hill climber" is also useful during the latter-stages of a run, to help refine a particular chromosome.

The recommended approach to take with JNetic is one of experimentation. An artist should try variations of genetic algorithm parameters, colour palettes, and chromosome definitions, to see what might arise. It is important to realize that separate runs using the identical parameters, can produce quite different outcomes, due to the randomness inherent in genetic algorithms. We treat JNetic as a tool akin to a complex paint brush. Just as an artist cannot control where every single bristle of a paint brush applies paint on a canvas, we do not presume that we should control all aspects of the results of evolution. Features of an image will arise from forces outside our purview. By exploiting the interactive features of JNetic, the artist can work with evolution to discover a creation that is more likely to fulfill his or her artistic goals. This reinforces the point of view that there is not a single best solution one is trying to find, but rather, innumerable possibilities from which to choose [9].

Creative opportunities may arise if the target image is considered an inspirational prototype or *archetype* for evolution, as opposed to being treated literally as an absolute goal. For example, a photograph can be retouched, edited, and painted before being given to JNetic, in an attempt to coerce and tease new colours

(a) Wilkens phase 1 (b) Wilkens phase 3 (c) JNetic

Fig. 8 Comparison of high resolution detail retention between Wilkens and JNetic. JNetic uses a mask over the face.

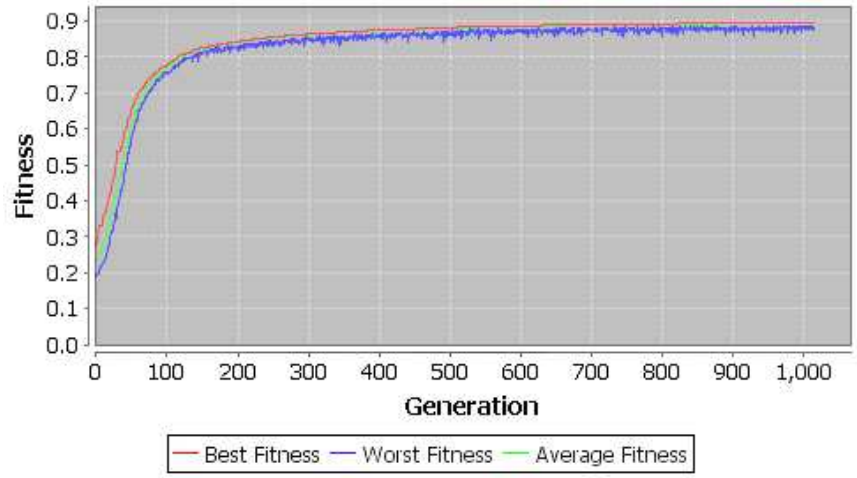


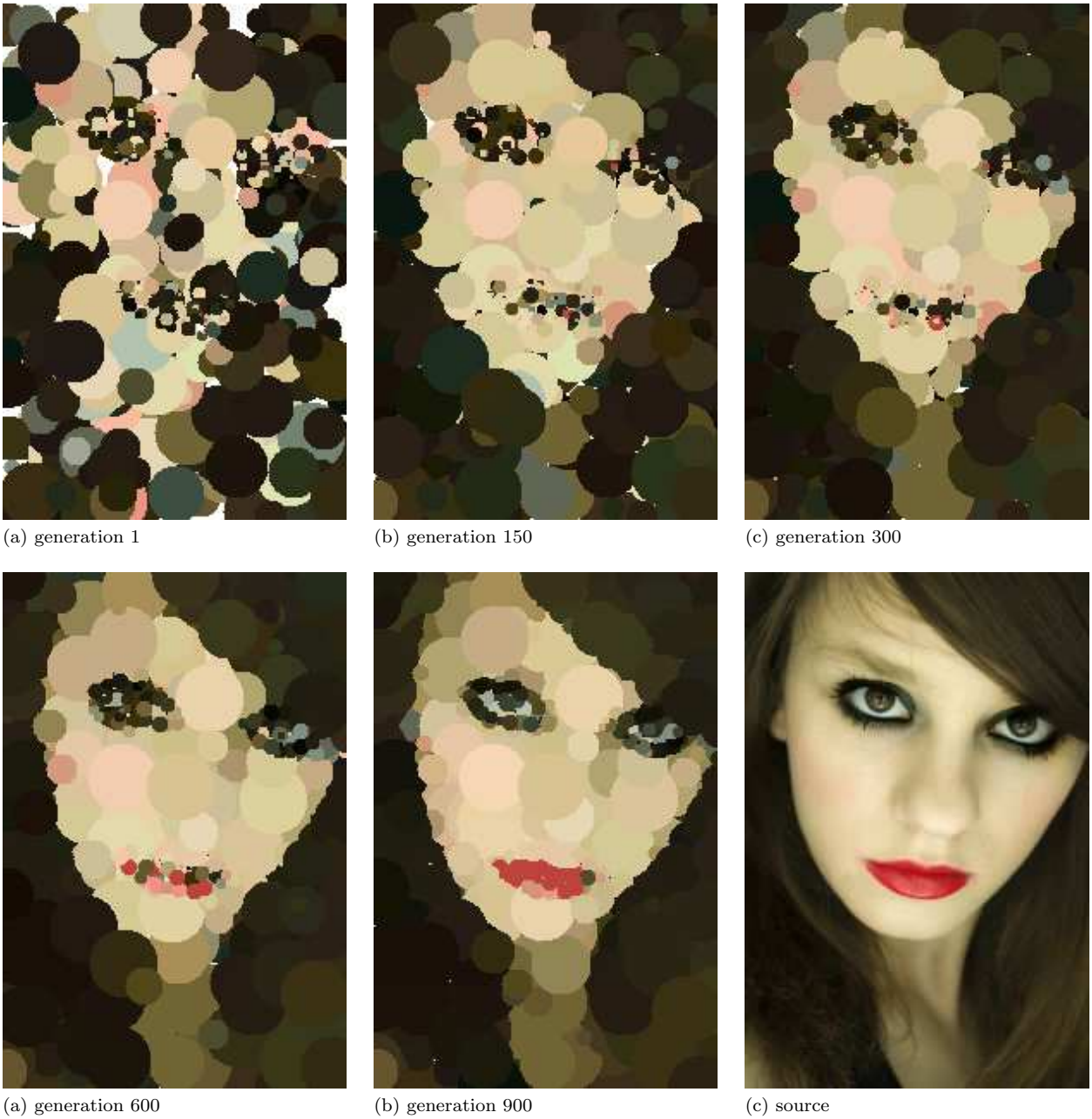
Fig. 9 Fitness performance of JNetic run in Fig. 8 (c).

and structures in the evolved images. Since generated results are geometric approximations of the target, this approach can result in innovative outcomes.

The products of JNetic may be the final results. Alternatively, they may be intermediate phases of larger projects. Images can be exported in the *scalable vector graphics* (SVG) format, which can be edited in graphics applications such as Adobe Illustrator. This lets an artist completely edit the geometry and appearance of a JNetic image. The SVG format allows images to be scaled to an arbitrary resolution with no loss of quality, perhaps for high-quality printing. This is a benefit of vector-based graphics that is not possible with evolutionary art applications that evolve bitmaps from texture expressions.

7 Conclusions

Vector graphics has not been studied in detail in the evolutionary art world, and so this paper fills a need for addressing this important class of image. Our work is motivated by the work of Weller [35] and Wilkens [39], and contributes a number of technical advancements. A significant contribution is the use of image masks for evolving high-detail areas of an image, should they be



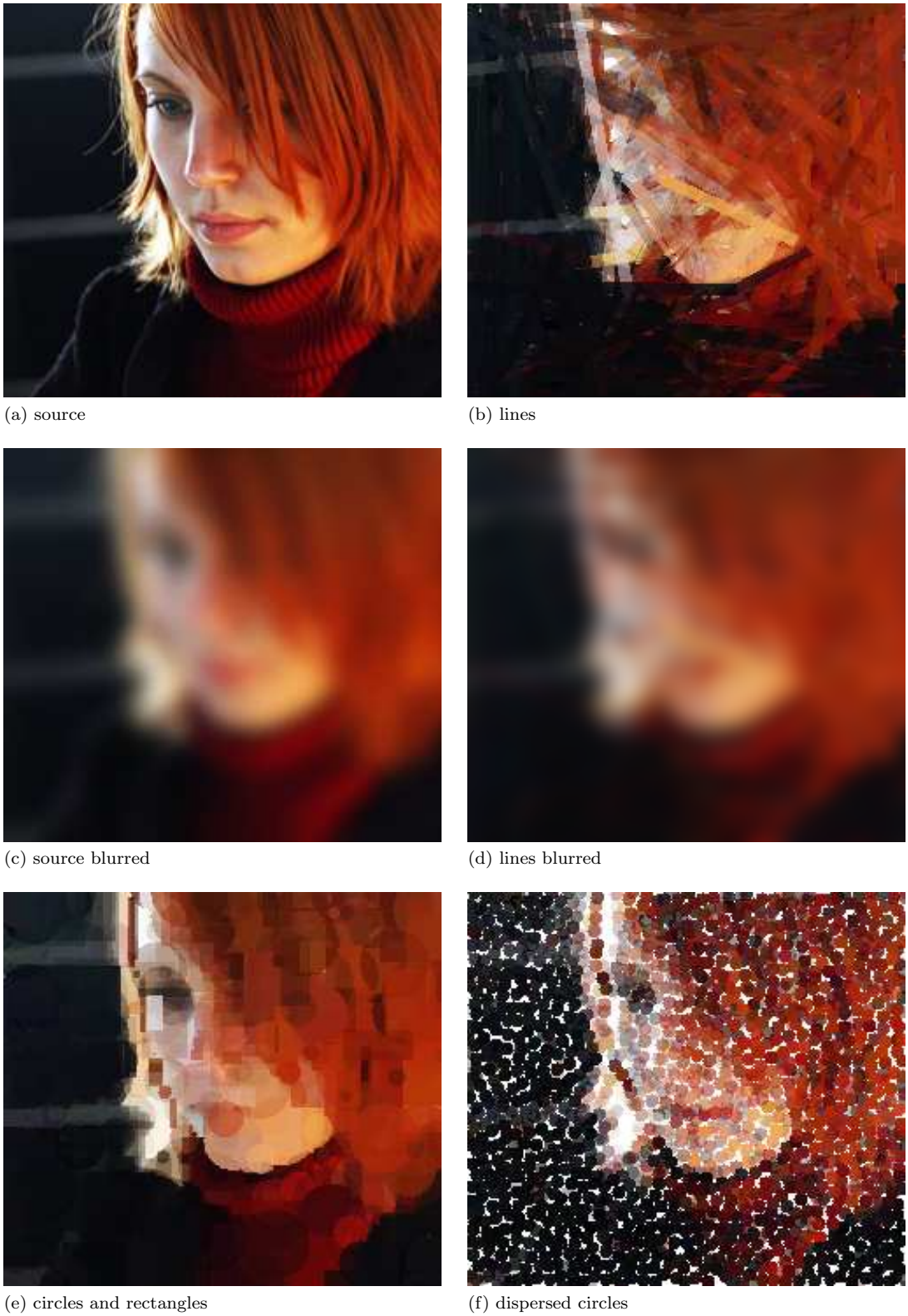
(a) generation 1 (b) generation 150 (c) generation 300

(a) generation 600 (b) generation 900 (c) source

Fig. 10 Progress during evolution.

desired. The Weller and Wilkens systems will usually result in images that have lost high-resolution details. Our system, JNetic, is distinguished by an interface that promotes user interaction with the evolutionary process, which elevates the role of an artist from passive viewer to active participant. JNetic is a self-contained artistic tool, that supports a variety of geometric primitives, colour models, and chromosome editing.

JNetic results share similarities with photo-mosaic tiling [38] and painting [2] applications that use genetic algorithms. As with JNetic, both applications generate images whose colour distance is in proximity with a target image. By animating the best image from each generation, an animated photo-mosaic or non-photorealistic painting effect is produced. JNetic permits the best image from each generation to be saved, which can be animated to give a similar effect. Our evolved images may have superficial similarities to some filter effects used in commercial applications, for example, Adobe Photoshop's mosaic filter. However, JNetic images are always composed of solid-coloured opaque or translucent geometric objects, while the mosaic tile effect can



(a) source

(b) lines

(c) source blurred

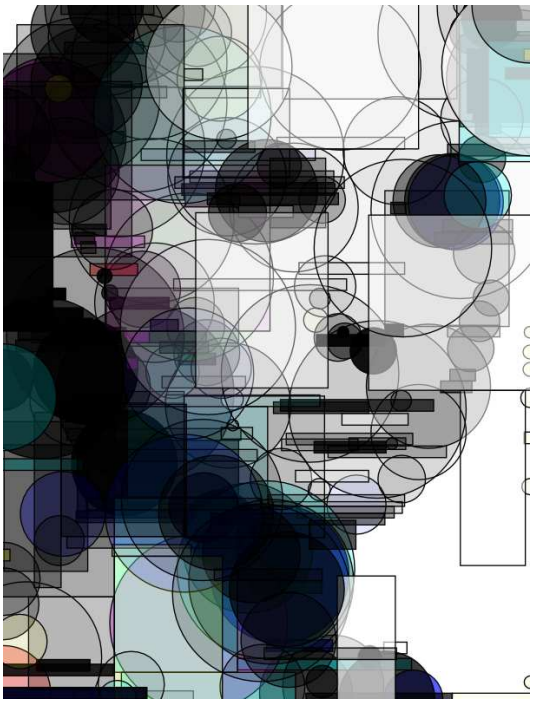
(d) lines blurred

(e) circles and rectangles

(f) dispersed circles

Fig. 11 More examples. Images (c) and (d) are the results of blurring images (a) and (b) respectively.

(a) *Painting* by S. Bergen.



(b) *Portrait* by B.J. Ross.

Fig. 12 Two final examples.

render bitmap textures onto the tiles. Furthermore, unlike most determinstic filter effects, JNetic results can be more variable and unpredictable in final appearance.

Research in computer graphics has explored the automatic vectorization of images. Our evolved vector images, however, are stylistically distinctive from most of that work. For example, Swaminarayan and Prasad [33] generates vector images that are somewhat graphic design oriented in appearance. Image analysis is first done to determine the construction of polygons, and their colours are assigned using colour sampling techniques. The results are akin to the "trace" functions found in some commercial vector graphics applications. Their ability to define complex polygons that directly map to internal details within images means that their results are more faithful renditions of the source image than is possible with our simpler geometries. A genetic algorithm might produce comparable results if enhanced spline-based polygons were included as geometric primitive. (Although we implement paint brush strokes residing on spline paths, they are intended to be non-photorealistic in appearance.) In addition, most vectorization algorithm such as theirs are deterministic, and multiple executions produce identical results. This is in contrast to genetic algorithms, whose stochastic nature means that separate executions may result in very different outcomes – some more preferable to others. The user is also able to influence evolved results with our system, by engaging in interactive evolution. Other automatic vectorization work such as [31,41] is motivated in creating exact reproductions of the source image. Therefore, their results are not comparable in either intent nor appearance to our stylized output.

Other enhancements to JNetic are being considered. Strategies for promoting population diversity would improve performance. Convergence is a universal issue with genetic algorithms, and runs often converge quickly to a suboptimal solution. Recently, we have developed an evo-art application called *JNetic Textures*, which combines vector graphics and procedural textures together, in an attempt to combine the strengths of both [4]. We see exciting prospects when mathematical models of aesthetics are considered [26,27]. We have also been investigating the idea of vector-based animations, and initial results are promising.

References

1. G. Bachelier. Embedding of Pixel-Based Evolutionary Algorithms in My Global Art Process. In J. Romero and P. Machado, editors, *The Art of Artificial Evolution*. Springer, 2008.
2. P. Barile, V. Ciesielski, K. Trist, and M. Berry. Animated drawings rendered by genetic programming. In *Proc. GECCO 2009*. ACM Press, 2009.
3. P. Bentley and D.W. Corne. *Creative Evolutionary Systems*. Morgan Kaufmann, 2002.
4. S. Bergen and B.J. Ross. Evolutionary Art Using Summed Multi-objective Ranks. In *Genetic Programming - Theory and Practice*, May 2010.
5. T. Boudreau, G. Jesse, S. Greene, J. Woehr, and V. Spurlin. *NetBeans: The Definitive Guide*. O'Reilly, 2002.
6. R. Buckley. Parallelohedra and uniform colour quanitzation. In A.W. Paeth, editor, *Graphics Gems*, pages 65–71. Academic Press, 1995.
7. A.R. Burton and T. Vladimirova. Generation of musical sequences with genetic techniques. *Computer Music Journal*, 23(4):59–73, Winter 1999.
8. R. Dawkins. *The Blind Watchmaker*. W.W Norton, 1996.
9. A. Dorin. Aesthetic Fitness and Artificial Evolution for the Selection of Imagery from the Mythical Infinite Library. In *Advances in Artificial Life – Proc. 6th European Conference on Artificial Life*, pages 659–668. Springer-Verlag, 2001.
10. A. Dorin. Artificial life, death, and epidemics in evolutionary, generative, electronic art. In *Applications of Evolutionary Computing: EvoWorkShops 2005*, pages 448–457. Springer-Verlag, 2005.
11. D.S. Ebert, F.K. Musgrave, D. Peachey, K. Perlin, and S. Worley. *Texturing and Modeling: a Procedural Approach*. Academic Press, 1994.
12. A.E. Eiben. Evolutionary Reproduction of Dutch Masters: The Mondriaan and Escher Evolvers. In J. Romero and P. Machado, editors, *The Art of Artificial Evolution*. Springer, 2008.
13. J. Elliot, R. Eckstein, M. Loy, D. Wood, and B. Cole. *Java Swing (2e)*. O'Reilly, 2002.
14. C.D. Frowd and P.J.B. Hancock. Evolving Human Faces. In J. Romero and P. Machado, editors, *The Art of Artificial Evolution*. Springer, 2008.
15. M. Gervautz and W. Purgathofer. A simple method for colour quantization: octree quantization. In A.S. Glassner, editor, *Graphics Gems*, pages 287–293. Academic Press, 1990.
16. D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, 1989.
17. J. Graf and W. Banzhaf. Interactive Evolution of Images. In *Proc. Intl. Conf. on Evolutionary Programming*, pages 53–65, 1995.
18. G. Greenfield. Evolving expressions and art by choice. *Leonardo*, 33(2):93–99, 2000.
19. G. Greenfield. Evolving aesthetic images using multiobjective optimization. In *Proc. CEC 2003*, pages 1903–1909, 2003.
20. J.H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1992.
21. A.E.M. Ibrahim. *GenShade: an Evolutionary Approach to Automatic and Interactive Procedural Texture Generation*. PhD thesis, Texas A&M University, December 1998.
22. H. Jackson. Toward a symbiotic coevolutionary approach to architecture. In P.J. Bentley and D.W. Corne, editors, *Creative Evolutionary Systems*, pages 299–313. Morgan Kaufmann, 2002.
23. F. Klawonn. *Introduction to computer graphics: using Java 2D and 3D*. Springer, 2008.
24. M. Lewis. Aesthetic Evolutionary Design with Data Flow Networks. In *Proc. Generative Art 2000*, 2000.
25. P. Machado and A. Cardoso. Computing Aesthetics. In *Proc. XIVth Brazilian Symposium on AI*, pages 239–249. Springer-Verlag, 1998.
26. C. Neufeld, B. Ross, and W. Ralph. The Evolution of Artistic Filters. In J. Romero and P. Machado, editors, *The Art of Artificial Evolution*. Springer, 2008.
27. J. Romero and P. Machado. *The Art of Artificial Evolution*. Springer, 2008.
28. S. Rooke. Eons of Genetically Evolved Algorithmic Images. In P.J. Bentley and D.W. Corne, editors, *Creative Evolutionary Systems*, pages 330–365. Morgan Kaufmann, 2002.
29. B.J. Ross, W. Ralph, and H. Zong. Evolutionary Image Synthesis Using a Model of Aesthetics. In *CEC 2006*, July 2006.
30. K. Sims. Interactive evolution of equations for procedural models. *The Visual Computer*, 9:466–476, 1993.
31. H. Sun, L. Liang, F. Wen, and H.-Y. Shum. Image Vectorization using Optimized Gradient Meshes. *ACM Transactions on Graphics*, 26(3), July 2007.
32. N. Svangard and P. Nordin. Automated Aesthetic Selection of Evolutionary Art by Distance Based Classification of Genomes and Phenomes using the Universal Similarity Metric. In *Applications of Evolutionary Computing: EvoWorkshops 2004*, pages 447–456. Springer, 2004. LNCS 3005.
33. S. Swaminarayan and L. Prasad. Rapid Automated Polygonal Image Decomposition. In *Proc. 35th Applied Imagery and Pattern Recognition Workshop (AIPR'06)*, pages 28–33, 2006.
34. S. Todd and W. Latham. *Evolutionary Art and Computers*. Academic Press, 1992.
35. C. Weller. Generation of vector-based graphics from existing bitmap images by means of the genetic algorithm, 2002. Last accessed April 28 2009.
36. M. Whitelaw. Breeding Aesthetic Objects: Art and Artificial Evolution. In P. Bentley and D.W. Corne, editors, *Creative Evolutionary Systems*, pages 129–145. Morgan Kaufmann, 2002.
37. A.L. Wiens and B.J. Ross. Gentropy: Evolutionary 2D Texture Generation. *Computers and Graphics Journal*, 26(1):75–88, February 2002.
38. G. Wijesinghe, S. Sah, and V. Ciesielski. Grid vs arbitrary placement for generating animated photomosaics. In *CEC 2008*, pages 2739–2745, 2008.
39. S. Wilkens. Rendering non-photorealistic images by means of a genetic algorithm, 2005. Unpublished student project.
40. WWW. Illustratorworld, 2010. Last accessed August 12, 2010.
41. T. Xia, B. Liao, and Y. Yu. Patch-Based Image Vectorization with Automatic Curvilinear Feature Alignment. *ACM Transactions on Graphics*, 28(5), December 2009.