International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:4, No:7, 2010

Evolved Strokes in Non Photo–Realistic Rendering

Ashkan Izadi,Vic Ciesielski
School of Computer Science and Information Technology
RMIT University, Melbourne, 3000, VIC, Australia
{ashkan.izadi,vic.ciesielski}@rmit.edu.au

Abstract—We describe a work with an evolutionary computing algorithm for non photo–realistic rendering of a target image. The renderings are produced by genetic programming. We have used two different types of strokes: "*empty triangle*" and "*filled triangle*" in color level. We compare both *empty* and *filled* triangular strokes to find which one generates more aesthetic pleasing images. We found the *filled* triangular strokes have better fitness and generate more aesthetic images than *empty* triangular strokes.

Keywords—Artificial intelligence, Evolutionary programming, Genetic programming, Non photo–realistic rendering.

I. Introduction

THE combination of Darwin's theory and computer graphics provide some new ideas of creating artistic images. By using evolutionary techniques, researchers and artists have created aesthetically pleasing images and videos. Producing art works can be done in different ways. One way is photo realistic rendering and many computer graphics researchers [3] have concentrated on this. Other researchers concentrated on *non photo–realistic rendering* or NPR. The researchers on NPR gave more attention to some aspects of rendering such as painting [6], pen-ink hatching [11], pencil sketches and drawings [3].
A large range of methodologies have been used for non photo-realistic rendering. We use *genetic programming* (GP). GP is tree based and has been successfully used for many problems, for example optimization, computer visions, symbolic regression.

We apply genetic operations (mutation, crossover, elitism). We use the "roulette wheel" selection for our algorithm. Many evolutionary approaches to NPR have used genetic algorithms [13], [15] but only a few works have been done with genetic programming. From the artistic aspect, the focus of the evolutionary algorithms is not just optimization but also is aesthetically pleasing renderings. By using genetic programming, we can produce artistic images. Barile and et. al. [1] used simple grey line strokes to draw on the canvas. Overall, our aim is to explore drawing different types of triangular strokes by using genetic programming to generate artistic animations.

The artistic perspective of NPR is to engage a viewer rather than just finding the quickest path to the target. To make more artistic animations, we give the opportunity to artist users to control renderings. For instance, a user can define the maximum length line of a triangle.

We select the best image of each generation as a frame of a movie and combine all of these frames to make an animation. The animation starts with a random collection of triangles and the target image is gradually revealed. The experiments and results detailed in this paper will answer the following research questions:

1) What is a suitable configuration of genetic programming for non photo–realistic rendering?
2) How different types of triangular strokes can affect the aesthetic qualities of the rendered images?

II. Related Work

Researchers and artists have modified and extended the computer graphics and evolutionary techniques to get aesthetic design [13], [16]. In the early 1990s, Haeberli designed a new method for rendering images. The Haeberli system gave the opportunity to to have interaction with the drawing process [5].

A new algorithm presented by Litwinowicz [9] used rectangular brush strokes to paint on the canvas. This algorithm was more successful than Haeberli system because this system did not lose details [3]. This system was more concerned with the complexities of brush placement than the easier ones. In late 1990s, Hetzmann employed a technique which used curved brush strokes [7]. Gooch and et al. employed a new "color segmentation" algorithms for stroke placement [4].

A. Evolutionary Approaches

There are several types of evolutionary algorithms, such as genetic algorithms (GA) [18], genetic programming (GP) [8]and differential evolution (DE) [14]. All forms of evaluation computing (EC) use Darwinian principle.

Genetic programming is a form of evolutionary computing which represents genetic material in the form of computer programs. The programs are typically represented as parse trees. Other forms of GP are "linear" GP, which produces assembly language representations, and "Cartesian" GP, which produces graph representations.

Two methods are usually applied the chromosomes, crossover and mutation. Crossover is a combination of two chromosomes which have been selected by selection process. Filter chromosomes have a higher probability of being selected.

Chakraborty [2] describes an approach for rendering target images by using a genetic algorithm representation. The system uses 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 parameter values.

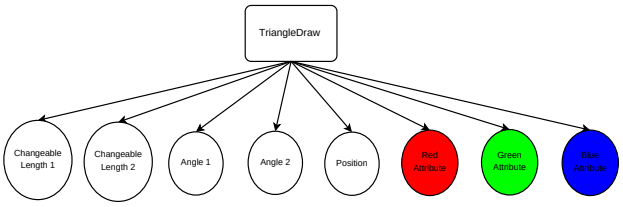

Fig. 1. The draw–triangle function with eight terminals.

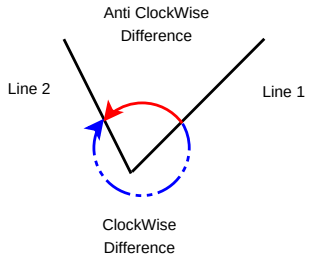

Fig. 2. The difference angle calculation of two lines.

Semet et al. [12] generate painterly and pencil sketch renderings using an ant colony model.

Barile and et al. [1]used simple grey line strokes in which they employed a draw function with four terminals. All research efforts in NPR share a common characteristic. This is to take an existing image and produced a new rendered image which possesses some likeness to the original image (target image). Like many other research efforts, our rendering technique also uses a reference image.

III. Representation

In this section, we describe the algorithms that we have implemented for our work. This section begins with descriptions of the draw-triangle and fitness functions. We then describe different types of strokes which we have employed. Last, we describe *genetic programming* approach and our methodology.

A. Draw–Triangle function

To draw a triangle on the canvas, we employ a "draw-triangle" function that has eight terminals. These terminals are first and second line lengths of the triangle, the angle of the first line on the canvas, the angle between the two lines, the position of the first line and three channel colors (red, green and blue). Fig. 1 shows the "draw–triangle" function.

The triangle–draw function specifies a position on the canvas. From this position the system draws the second line of triangle. This function uses two angles. When a pixel is being drawn the brush value combined with the corresponding existing pixel value on the canvas according to the following formula (we employ ϕ as 0.5).

$$Red_{new} = Red_{stroke} * \phi + Red_{canvas} * (1 - \phi)$$

$$Green_{new} = Green_{stroke} * \phi + Green_{canvas} * (1 - \phi) \quad (1)$$

$$Blue_{new} = Blue_{stroke} * \phi + Blue_{canvas} * (1 - \phi)$$

$$NewPixel = Red_{new} + Green_{new} + Blue_{new}$$

Triangular strokes are painted on the canvas without any reference to the target image and we employ a pixel-by-pixel fitness function. To compute fitness, we add pixel differences between the three channels on the target image and the corresponding channel on the rendered image. Then we normalized the result of each channel(Δ_n) and get the average.

$$\Delta Red = \sum_{i=1}^{M}\sum_{j=1}^{N} |targetRed(i,j) - canvasRed(i,j)| \quad (2)$$

$$\Delta Green = \sum_{i=1}^{M}\sum_{j=1}^{N} |targetGreen(i,j) - canvasGreen(i,j)|$$

$$\Delta Blue = \sum_{i=1}^{M}\sum_{j=1}^{N} |targetBlue(i,j) - canvasBlue(i,j)|$$

$$f(x) = \frac{\Delta_n Red + \Delta_n Green + \Delta_n Blue}{3}$$

B. Triangular Strokes

We give an opportunity to artist users to select the maximum length size of two lines in a triangle. If these lines are specified as zero, the system provides a default value. The system will choose the angle for each line. We use the following calculation to find the third line length:

$$\Delta\theta = \theta 1 - \theta 2 \quad (3)$$

$$L3 = \sqrt{L1^2 + L2^2 - 2L1L2\cos\Delta\theta} \quad (4)$$

Where $\theta 1$ is angle of line one with a canvas, $\theta 2$ is the angle of second line with a canvas, $\Delta\theta$ is the angle difference between two lines, L1 is the first line, L2 is the second line and L3 is the third line.

We check whether the two lines and angles are capable to create a triangle or not. Sometimes two lines have the same angle with the canvas or their angle difference is π. This means that we can not have a third angle. To calculate $\Delta\theta$, we should calculate it in both clockwise and anti-clockwise. Then we choose the smaller value as $\Delta\theta$ Fig. 2 shows this concept.

We always assume that line three is drawn from the last pixel of line one. To calculate $\theta 3$, we shift $\theta 1$ to the new position (θ_{new}) by adding 180 degree to $\theta 1$. The following equation shows this.

$$\theta new = \theta 1 + \pi \quad (5)$$

The angle of the third line with the canvas depends on two factors:

- Which one of the angles ($\theta 1$ or $\theta 2$) is greater?

TABLE I
PARAMETERS OF GENETIC PROGRAMMING AND DRAWING PROCESSES.

Functions	program3,4 and draw-triangle
Terminals number	8
Crossover	50%
Mutation	25%
Elitism	25%
Selection	Roulette Wheel
Termination	Max. generation reached or 0.005 difference
Technique	Blending

- Is $\Delta\theta$ greater than π degree or not?

By computing these factors, we select one of the following equations to get $\theta 3$ ($\theta 3$ must be between 0–2π). If $\theta 3$ is out of the range 0–2π, we have to add or subtract 2π to get the final and correct $\theta 3$.

$$\theta 3 = \theta new + \phi 2 \tag{6}$$

$$\theta 3 = \theta new - \phi 2 \tag{7}$$

To draw the empty triangle, pixels can be written on the canvas by the aforementioned calculations. But to draw a filled triangle we have to fill up an empty triangle. To fill up an empty triangle we employ "barycentric coordinates" (Vince [17]). We create an invisible rectangle around of each triangle to reduce the cost of searching for an inner pixel.

C. Genetic Programming Configuration

We employ three functions, program3, program4 and draw–triangle, in our GP approach. Program3 and 4 have the same action, they do affect drawing directly because they accept just draw-triangle functions or themselves as arguments. Their purpose is to extend the trees and enable more brushstrokes to be drawn on the canvas. We employ the program3 and program4 because we want an ordered list sequence of triangle strokes. The draw-triangle function just accepts terminals and draws strokes on the canvas directly. The terminals of the draw-triangle function which we described in the previous section (III.A) are real numbers in the range 0.0 to 1.0. Fig. 3 shows the relation between the Program 3 and program4 and the draw-triangle nodes.

We use random mutation and crossover. If the mutation node happens to be at the bottom of a tree, it causes changes only on the terminal of a draw-triangle function. So, it will probably have minimal effect on the canvas. But if this operation happens near the root of the tree, it could be a significant alteration of the canvas. For crossover we have the same situation. The parameters which we employ in our genetic programming configuration are shown in Table I.

IV. RESULTS AND EXPERIMENTS

In this section, we describe the various experiments that we have done in order to investigate the kinds of renderings that can be achieved with the two types of brushstrokes. Artist users have a facility to define some parameters of the program such as line length or different type of triangles. From a computer science perspective, in genetic programming we desire to find an optimal result quickly, that is fast convergence, and employ the fewest resources. However, from the artistic aspect, we desire aesthetically pleasing renderings rather than just fast convergence and reach to target image.

TABLE II
CONFIGURATION OF OUR GENETIC PROGRAMMING.

Functions	program3,4 and draw-triangle
Terminals number	8
Generation	100,000
Max Tree size	8
Min Tree size	3
Population	4
Crossover	2
Mutation	1
Elitism	1
Selection	Roulette Wheel
Termination	Max. generation reached or 0.005 difference
Technique	Blending

We have experimented with different images and many runs for each image. We used the GP parameters shown in Table II. We tested different tree sizes for each image. The tree depth must be enough large to generate enough strokes to give a recognizable target. We found that if the maximum tree size is bigger than eight, we reach the target very early. This is not desirable from the artistic[1] perspective. If we provide a minimum tree size bigger than four, we paint on the canvas with a lot of strokes in very first generations. This is not also desirable from the artistic aspect. Thus we try to produce the animation which starts with just a few triangles and then gradually increase number of triangles until the target is reached. We found that the best configuration for maximum tree size is eight or seven and for minimum tree size is three.

We investigated populations of four and a hundred. Usually in GP a larger population has better convergence and fitness rather than a smaller population because there are more individuals in the large population and this provides more chance to find a better individuals in the search space. Our results are not content with this explanation.

The experiments show that a population four has better fitness and output than a population of a hundred. The reason for this needs more investigation, but it might be because the improvement of fitness is very small and population size cannot affect our fitness function that much. Another reason could be the blending technique: triangles cross and if there have bad fitness, it can be improved. When we compare both populations with each other we conclude that the population of four is also less costly than population a hundred and has better convergence. So we have selected the population four for our configuration. To produce artistic animations, number of generations should be around one hundred thousand otherwise we will not close to the target. We use table II as the main configuration for our rendering images.

We provide two different images in this paper out of

[1]Dr. M. Berry and K. Trist, from school of creative media, RMIT

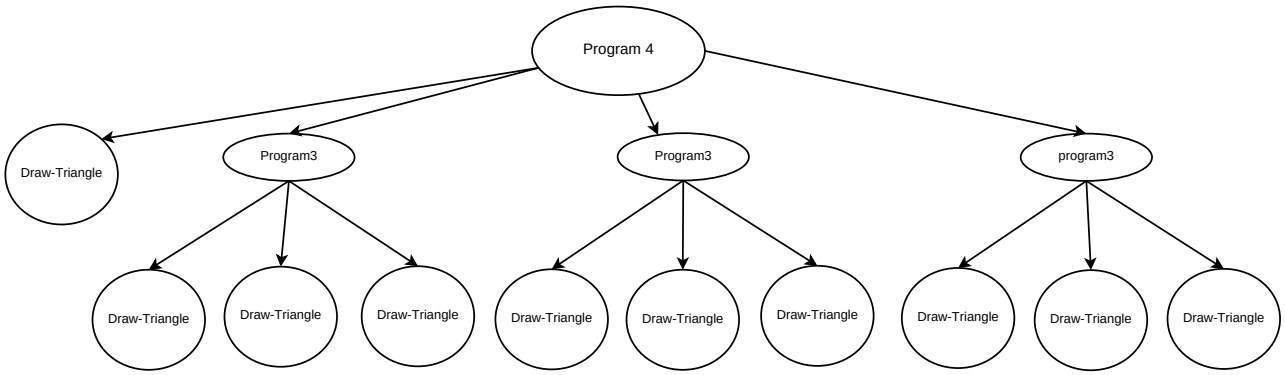

Fig. 3. Genetic programming tree with three different functions. The draw–triangle function gets just terminals.

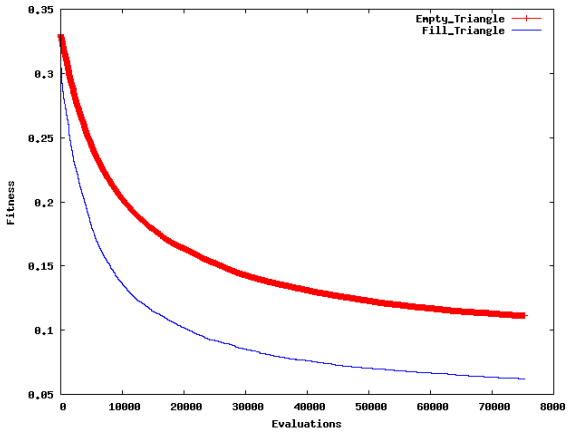

Fig. 4. The evaluation of fill triangle and empty triangle.

several runs with different images. Fig. 5 and Fig. 6 show the result of our triangular strokes. We have compared the empty triangle and the fill triangle with each other by numeric measurement (their fitness) to find which one can produce the better convergence. We show one of these comparisons in Fig. 4.

V. Conclusion

In this work we have implemented two new type of strokes that are the empty and filled triangle. Relating to research question #1, we produce the rendering by employing genetic programming. We have shown that our genetic programming configuration which combines three different node types (program3, program4 and draw–triangle) produces trees that generate linear sequences of triangular strokes. The fitness function is the sum of pixel differences between the rendered individual and the target image. We found that the maximum tree size for our goals has to be less than eight and also the minimum tree size is less than four. We conclude that a small population size outperforms a large population size, however, this needs more investigation to understand why it happens. Relating to research question #2, we implemented different types of triangular strokes . Fig. 4 shows that filled triangle can achieve 60% better fitness than empty triangle.

Artists[2] believe that the filled triangle is more engaging than the empty triangle. They explain that the rendering of empty triangle is similar to the lines after the first thousand generations. Thus there is not very big difference between a simple line and the empty triangle rendering. Overall, the filled triangle achieves the goals of an engaging rendering with good convergence to the target.

In future work, we tend to investigate other methods of stroke drawings to create more interesting renderings. Moreover, we will investigate alternate evolutionary forms such as “Cartesian genetic programming”.

Acknowledgment

We thank our evolutionary art group Dr. D. D’Souza, Dr. J. Riley, Dr. M. Berry and K. Trist for suggestions relating to this paper.

References

[1] P. Barile, V. Ciesielski, and K. Trist. Non-photorealistic rendering using genetic programming. In *SEAL ’08: Proceedings of the 7th International Conference on Simulated Evolution and Learning*, pages 299–308, Berlin, Heidelberg, 2008. Springer-Verlag.

[2] U. Chakraborty and H. Kang. Stroke-based rendering by evolutionary algorithm. pages 52 – 57, Dec. 2004.

[3] J. P. Collomosse. Evolutionary search for the artistic rendering of photographs. In J. Romero and P. Machado, editors, *The Art of Artificial Evolution*, Natural Computing Series.

[4] B. Gooch, G. Coombe, and P. Shirley. Artistic vision: painterly rendering using computer vision techniques. In *NPAR ’02: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, pages 83–ff, New York, NY, USA, 2002. ACM.

[5] P. Haeberli. Paint by numbers: abstract image representations. In *SIGGRAPH ’90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 207–214, New York, NY, USA, 1990. ACM.

[6] A. Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *SIGGRAPH ’98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 453–460, New York, NY, USA, 1998. ACM.

[7] A. Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *SIGGRAPH ’98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 453–460, New York, NY, USA, 1998. ACM.

[8] J. R. Koza. *Fundamental Algorithms*, volume 1 of *On the Programming of Computers by Means of Natural*. MIT press, Cambridge, Mass., four edition, December 1992.

[2]Dr. M. Berry and evolutionary art group of RMIT

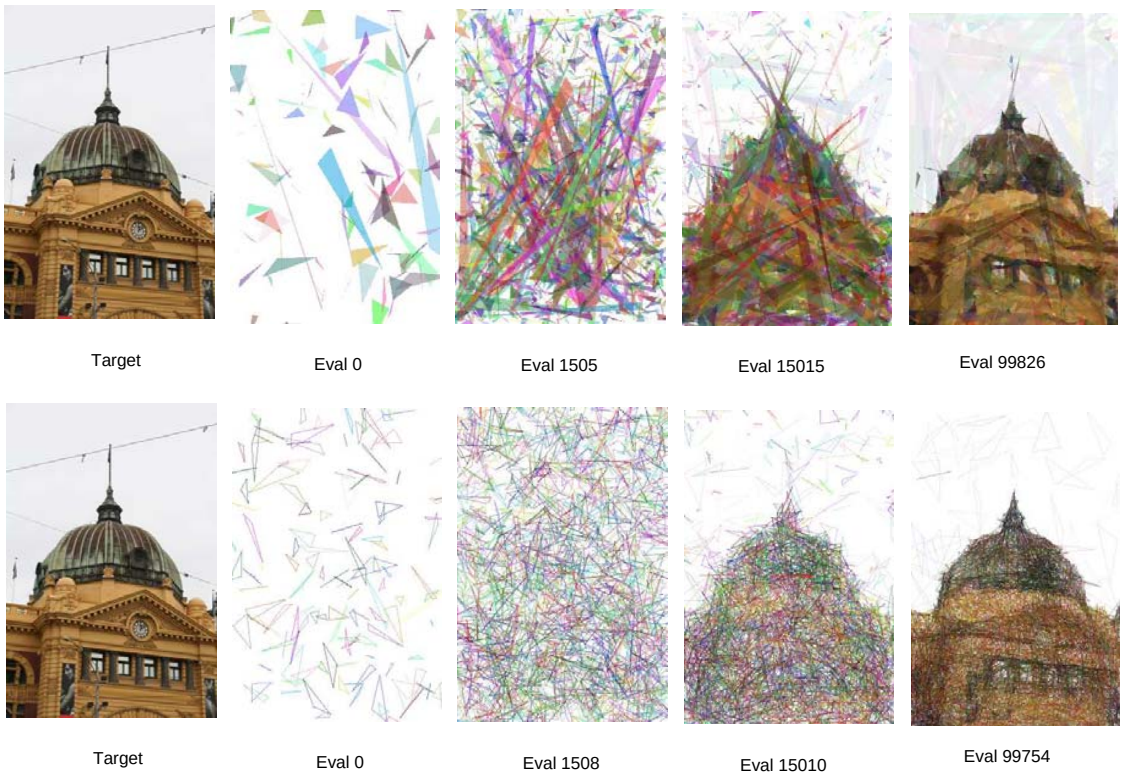

Fig. 5. Sequence from a run of the evolutionary algorithm by using triangle strokes.

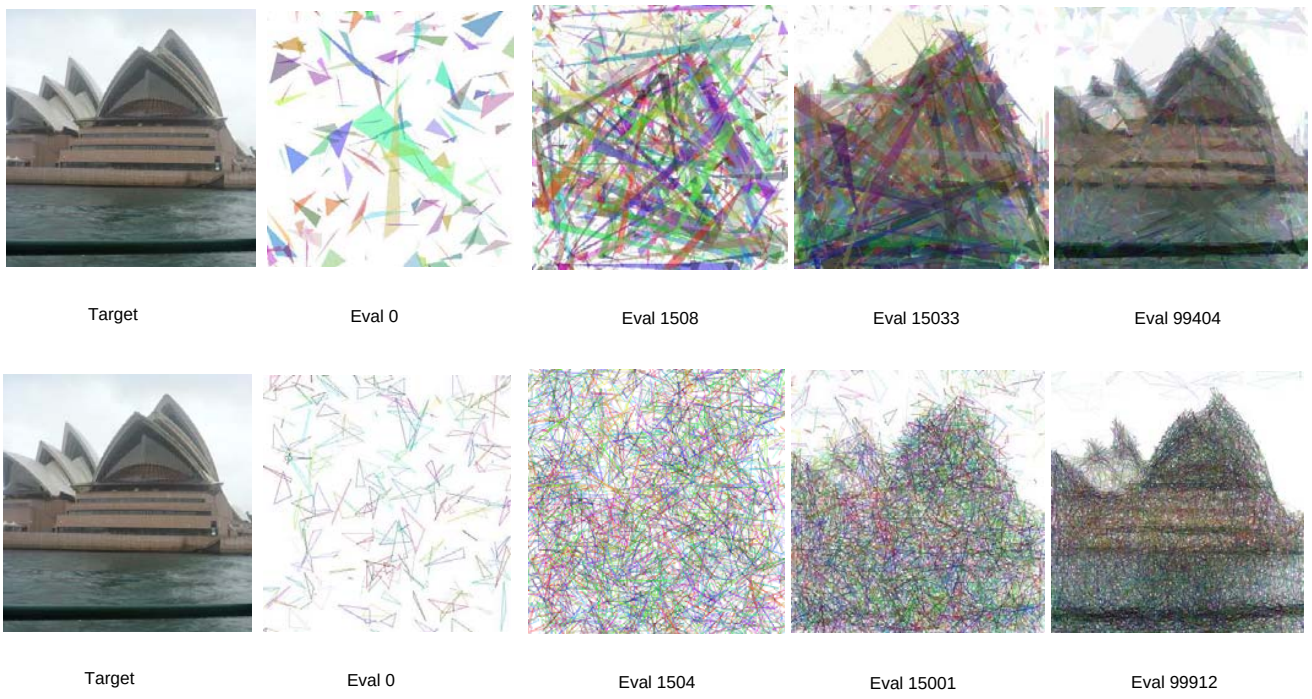

Fig. 6. Sequence from a run of the evolutionary algorithm by using triangle strokes.

[9] P. Litwinowicz. Processing images and video for an impressionist effect. page 407414. ACM SIGGRAPH, 1997.
[10] P. H. P. Collomosse. Genetic paint: A search for salient paintings. In *Applications on Evolutionary Computing*, pages 437–447, New York, NY, USA, 2005. Springer Berlin / Heidelberg.
[11] M. P. Salisbury, M. T. Wong, J. F. Hughes, and D. H. Salesin. Orientable textures for image-based pen-and-ink illustration. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 401–406, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.
[12] U.-M. D. F. Semet, Y. O Reilly. An interactive artificial ant approach to non-photorealistic rendering. In *Genetic and Evolutionary Computation GECCO 2004*, pages 188–200. SpringerLink, 2004.
[13] K. Sims. Artificial evolution for computer graphics. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 319–328, New York, NY, USA, 1991. ACM.
[14] R. Storn and K. Price. Differential evolution a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11, December.
[15] D. Terzopoulos. Artificial life for computer graphics. *Commun. ACM*, 42(8):32–42, 1999.
[16] S. Todd and W. Latham. *Evolutionary Art and Computers*. Academic Press, Inc., Orlando, FL, USA, 1994.
[17] J. A. Vince. *Mathematics for Computer Graphics*, chapter 12, pages 193–221. Undergraduate Topics in Computer Science. Springer London, 2006.
[18] D. Whitley. A genetic algorithm tutorial. In *Statistics and Computing*, pages 65–85, Netherland, October 2004. Springer.