Multi-scale Stroke-based Rendering by Evolutionary Algorithm

Hyung W. Kang, Uday K. Chakraborty, Charles K. Chui, Wenjie He

Department of Mathematics and Computer Science
University of Missouri, One University Blvd., St. Louis, MO 63121, USA
{kang,uday,chui,he}@arch.umsl.edu

Abstract

This paper presents an efficient method based on evolutionary algorithm for optimizing the rendering quality in multi-scale stroke-based non-photorealistic rendering. The proposed method produces better results than previously published methods such as random descent and the single-level genetic algorithm-based approach.

Keywords: Non-photorealistic rendering, Stroke-based rendering, Computer graphics, Evolutionary algorithm

1 Introduction

This paper extends our previous work [1, 2] on developing an efficient strategy for non-photorealistic (painterly) rendering using evolutionary algorithms. In painterly rendering, the basic elements for constructing pictures are brush strokes. In general, a stroke is characterized as a rectangular object with a number of parameters, including position (x, y), color (R, G, B), orientation (θ), width (w) and height (h). Based on the parameter values, each stroke is placed on the canvas (white image) one by one to simulate the painting process. One notable property in placing strokes is that there can be many overlapping strokes as in real paintings (see Fig. 1). Thus, the ordering of stroke is significant as it affects the quality of the result.

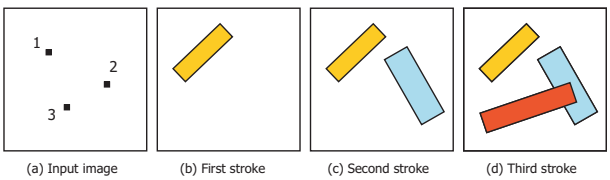


Figure 1. Strokes for painterly rendering

Haeberli [4] introduced the first stroke-based rendering system, which was based on the user's interactive stroke placements on a given input image. Litwinowicz [8] and Meier [9] used particles to uniformly place strokes on a canvas and to follow their movements in a video. Hertzmann [5] presented a layered painting method using multi-sized brushes, and later proposed an algorithm for optimizing the rendering results [6]. Some researchers also developed stroke distribution algorithms based on computer vision techniques such as image moment [10] and image segmentation [3]. Although each of these approaches successfully produces its own distinct rendering style, none of these aims particularly at optimizing the rendering quality, except for [6] which was based on a random descent process. In our previous work [1, 2], we showed that our approach based on evolutionary algorithm performs better than random descent algorithm in terms of minimizing the fitness value. In the present paper, we extend our previous algorithm to incorporate multi-level stroke distribution, achieving even better results.

The rendering problem. The problem to be solved can be stated as follows: Given an input image, place the strokes on a white canvas (image) so that the fitness function (equation 1) is minimized. To simplify the problem, we optimize the position variables (x, y) and the color variables (R, G, B) in our current implementation, compute the orientation and length from the input image, and let width be fixed for a given level (scale).

Multi-level stroke distribution. In our approach, the strokes are regarded as geometrical entities in a continuous 2D domain. More specifically, each stroke is associated with a 2D point and a stroke distribution is modeled as a set of arbitrarily scattered data points on a 2D space. Note that this model is consistent with the actual painting process of the artists, where they can apply an arbitrary number of brush strokes at arbitrary positions on the canvas. To deal with the stroke distribution in multiple levels, the strokes at each level i are represented as a point set $P^i = \{\mathbf{p}_0^i, \mathbf{p}_1^i, ..., \mathbf{p}_{max(i)}^i\}$ where each point $\mathbf{p}_j^i$ in P^i is associated with its 2D position (x_j^i, y_j^i). The point set P^i starts with a relatively small number of points at the coars-

est level ($i = 0$), and it grows larger as i increases, or as we move to the the finer levels. Again, this is similar to the artists' painting process where new strokes are incrementally added on top of the old strokes on the canvas. Note that the successive point sets in this multiresolution setting maintain a nested structure, where the higher-level point set contains all the lower-level point sets.

2 The algorithm

An evolutionary algorithm with a variable-length chromosome and several domain-specific mutation operators is used.

Chromosome representation. Multiple layers (levels) of strokes are employed. The strokes at the beginning levels are larger (i.e., wider), covering more of the canvas, while the later levels have progressively smaller stroke widths. The bigger strokes are placed first, followed by the smaller ones. The multi-scale resolution concept is implemented in the evolutionary algorithm by having a single chromosome encode strokes at different levels. In this paper, we report results for five levels; it is easy to generalize the approach to any number of levels. A chromosome comprises five sections, each section having a dynamically varying number of strokes. A section corresponds to a level (layer). The strokes of a particular section (level) have the same size, while the strokes of the next section (upper level) have a lower size. Each section has a pre-determined stroke size. This simple chromosome representation permits parallel stroke placement in multiple resolutions. Clearly, the number of strokes is not fixed but varies across sections (in a given chromosome) and also across chromosomes. Further, the number of strokes in a given section of a given chromosome varies over time (generations). The minimum and maximum number of strokes that a chromosome may have are predetermined constants. In the present implementation the number of strokes in a chromosome varies dynamically during a single run.

A single stroke is represented by its two-dimensional coordinates (x and y) and the three color components (R, G and B). The x and y values are floating-point numbers in the interval [0,1], with a particular x (or y) value being mapped to the abscissa (or ordinate) of a pixel of the image. This allows the present encoding method to handle any input image size. The R, G and B values are integers between 0 and 255 inclusive. The strokes of a section are implemented as elements of a dynamically growing / shrinking array. The ordering of the elements of the array determines the sequence of stroke placement.

Population initialization. The initial population is created by giving each chromosome a random number of strokes between two limits, $MIN\_STROKES$ and $MAX\_STROKES$. This is done by initializing each section of each chromosome with a random number of strokes between $MIN\_STROKES/5$ and $MAX\_STROKES/5$. The (x, y) coordinates of each stroke of each section are initialized uniformly randomly in the interval [0,1]. Each of the R, G, B components of each stroke of each chromosome is set to the corresponding value in the input image and then perturbed following color-perturbation-mutation (explained later).

Fitness function. The goal is to generate a rendering as close to the input picture as possible while using the minimum number of strokes. That is, the strokes are to be placed so that the total number of strokes is minimized to maximize the abstractness, and the difference from the input image is minimized to maximize the rendering quality. The following fitness function is used (I and I' represent the input and the output images, respectively):

$$f(I') = (1 - 1/\#strokes) \sum_{\mathbf{x}} \|I'(\mathbf{x}) - I(\mathbf{x})\|^2 \qquad (1)$$

The multiplying factor is used to reward chromosomes with a fewer number of strokes. The problem is one of minimization.

Selection. Binary tournament selection with replacement is used where the winner of the tournament is selected with probability 1. The generational, elitist model of the genetic algorithm is used, where the best individual of the previous generation replaces the worst individual of the current generation.

Crossover. Two children are created by crossover between two parents. Crossover is implemented by uniformly randomly choosing a section (between 1 and 5), and then choosing, again uniformly randomly, two cut-points in that section – one cut-point for each parent. For example, suppose the section chosen for a particular crossover operation is 3 (picked between 1 and 5). Let the current length (i.e., number of strokes) of section 3 in the first parent be 10, and let that length in the second parent be 7. Then the first cut-point is between 1 and 10, while the second is between 1 and 7. Once the cut-points are determined, strokes of the chosen section are swapped between the parents. Thus a single crossover operation affects exactly one (the same) section in each of the two parent chromosomes so that strokes of the same size are swapped.

Mutation. Four different problem-specific mutations are used. In each of these mutations, better mutants are unconditionally accepted while worse mutants are accepted with a low, predetermined probability.

Delete strokes: Both the total number of strokes to be deleted from a given chromosome and the individual strokes (to be deleted) are chosen randomly. The total number of strokes to be deleted is determined as a random integer between zero and the minimum of the following two: a (predetermined) fraction of the current length of the chromosome,

and current length – $MIN\_STROKES$. While performing stroke deletion, no attention is paid to section boundaries. The individual deletions in a single mutate-delete operation are applied one after another, following the above-mentioned accept/reject policy at each deletion.

Add strokes: A random number of new strokes are added at the end of a randomly chosen section of the chromosome. The sizes of the new strokes correspond to the section that is being augmented. One mutate-add operation adds to exactly one section. (In general, different mutate-add operations work on different sections.) The color components of the new strokes are first obtained from the input image and then altered by applying color-perturb-mutation (explained later). Note that the new strokes are added at the end of the section, that is, they are placed on top of other strokes at that level, since in general placing the strokes 'beneath' the already placed strokes on the canvas does not have much impact on changing the rendering result. The total number of strokes to be inserted in a single operation is determined as a random integer between zero and the minimum of the following two: a (predetermined) fraction of the current length of the section, and $MAX\_STROKES/5$ – current length of the section.

Reverse segment: One section is randomly chosen for reversing. Two randomly chosen strokes are used to define a segment of strokes in that section, and the strokes in the segment are reversed. Like mutate-add and crossover, this mutation affects exactly one section (level) of the chromosome.

Perturb strokes: Two types of perturbations are applied to a stroke — position perturbation and color perturbation, the two perturbations being independent of each other. Each stroke in a chromosome is perturbed, independently of any other stroke, with a (predetermined) position-perturb probability. For the x of each stroke, a random perturbation amount is obtained by multiplying a predetermined fraction by a uniform random number in [0,1]. The perturbation amount is then added to (or subtracted from) the original x value. The add / subtract decision is made with a 50% probability. The y component of a stroke is similarly perturbed. The color components of a stroke are also perturbed, probabilistically, and independently of the x- (or y-) perturbations. With a predetermined color-perturb probability, a stroke is chosen for color perturbation and each of its three color components is given a random amount of positive or negative perturbation. The perturbations are guaranteed not to result in out-of-bound values.

Orienting strokes. The orientation of each stroke is derived from the input image $I(x, y)$. First, we generate a gradient image ∇I whose pixel contains three components (dx, dy, mag), where (dx, dy) denote the gradient direction and mag represents the gradient magnitude. We then blur ∇I several times to reduce noise. While the gradient image contains the correct direction information near the strong edges, the information in the homogeneous regions is usually incorrect as their gradient magnitudes are nearly zero. To create a smooth gradient vector field, we diffuse the information from the strong edges to nearby pixels by iteratively applying weighted sum in a spatial mask (with mag values as weights), while keeping the information around the strong edges (using a certain threshold on mag). Finally, we turn the gradient directions 90° (by resetting them as $(-dy, dx, mag)$), to make them follow the shapes of the object boundaries. The resulting gradient image contains the desired gradient vector field to guide the stroke orientation at each pixel.

Termination condition. The quality of the rendered image is to a great extent a matter of human perception. Equation 1 shows that the best possible solution has $f = 0$, corresponding to the case when the input and the output images are the same. Clearly, that would never be an acceptable solution. It is also to be noted that it is possible for two (or more) different chromosomes to have the same numerical fitness. We chose to terminate a run when either the fitness value of the best individual in the current population drops below a predefined threshold or the maximum number of generations is reached.

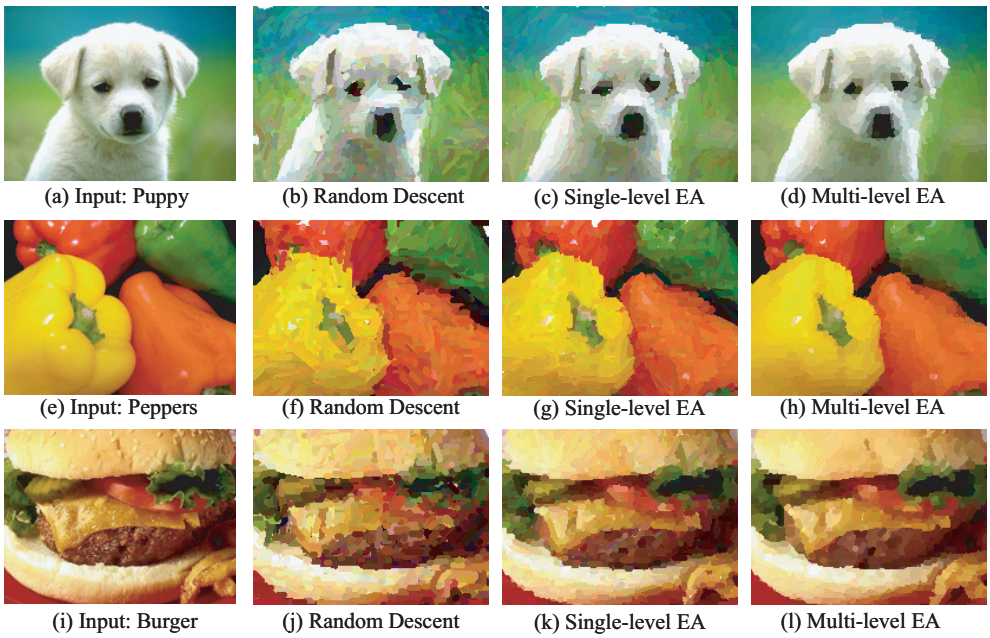
3 Experimental results

We conducted experiments on a number of input images of various sizes to compare the rendering results of our method against (1) a random descent method (which is the core idea in [6]), and (2) our single-level genetic algorithm-based method [1, 2]. For the random descent method, an iterative process of random stroke insertion, move, delete operations is applied, where each of these trial operations is first tested and then accepted only when it is confirmed to lower the fitness value. As shown in Figure 2 and Table 1, our method outperforms both the above-mentioned approaches on two counts: (a) fitness value, and (b) the visual quality of the rendered images. For the same number of strokes, the multi-scale method produces smaller (better) fitnesses for all the input images. Because of space restrictions, we show only three images in Figure 2.

The stroke length is dynamically determined by applying a stroke clipping algorithm [8], where each stroke is clipped at nearby edges with strong image gradient values. We used a fixed stroke widths for a given section (level). This stroke rendering method was also applied for the random descent method.

4 Conclusion

This paper described an efficient method for optimizing the rendering quality for multi-scale stroke-based rendering,



(a) Input: Puppy (b) Random Descent (c) Single-level EA (d) Multi-level EA

(e) Input: Peppers (f) Random Descent (g) Single-level EA (h) Multi-level EA

(i) Input: Burger (j) Random Descent (k) Single-level EA (l) Multi-level EA

Figure 2. A comparison of rendering results for various images

Image	# of strokes	Random Descent	Single-level EA	Multi-level EA
Puppy	600	925603.3	669895.2	479909.0
Peppers	500	894295.7	659004.4	519893.8
Burger	750	1191108.8	892121.0	689824.4

Table 1. Comparison of the three methods

based on an evolutionary algorithm that uses new, problem-specific mutation operators and a variable-dimensional chromosome. Initial results showed that the present method outperformed both random descent-based method and our previously published evolutionary algorithm-based approach.

Acknowledgments: HWK's research is supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Assessment). UKC is supported by a 2004 UMSL Research Award.

References

[1] U. K. Chakraborty and H.W. Kang, Stroke-based rendering by evolutionary algorithm, Proceedings of IEEE INDICON 2004, pp. 52-57.

[2] U.K. Chakraborty and H.W. Kang, Painting by evolutionary algorithm, Proc. 5th International Conference on Recent Advances in Soft Computing (RASC2004), December 2004, pp. 249-254.

[3] B. Gooch, G. Coombe, and P. Shirley (2002). Artistic Vision: Painterly Rendering Using Computer Vision Techniques. In *Proc. Non-Photorealistic Animation and Rendering*, 2002.

[4] P. Haeberli (1990). Paint by Numbers: Abstract Image Representations. In *Proc. ACM SIGGRAPH*, pages 207-214, 1990.

[5] A. Hertzmann (1998). Painterly Rendering with Curved Brush Strokes of Multiple Sizes. In *Proc. ACM SIGGRAPH*, pages 453-460, 1998.

[6] A. Hertzmann (2001). Paint by Relaxation. In *Computer Graphics International*, pages 47-54, 2001.

[7] A. Hertzmann (2003). A Survey of Stroke-Based Rendering. In *IEEE Computer Graphics and Applications*, vol. 23, no. 4, pages 70-81, 2003.

[8] P. Litwinowicz (1997). Processing Images and Video for an Impressionist Effect. In *Proc. ACM SIGGRAPH*, pages 407-414, 1996.

[9] B. Meier (1996). Painterly Rendering for Animation. In *Proc. ACM SIGGRAPH*, pages 477-484.

[10] M. Shiraishi and Y. Yamaguchi (2000). An algorithm for automatic painterly rendering based on local source image approximation. In *Proc. NPAR*, pages 53-58, 2000.