This paper presents an effective method based on genetic algorithm for optimizing the rendering quality in image-based painterly rendering. Based on a multi-level evolutionary approach, the proposed method produces, for a variety of input images, results that are better in a statistically significant way than previous methods.

Keywords: Non-photorealistic rendering; painterly rendering; computer graphics; optimization; evolutionary algorithm.

1. Introduction

This paper describes an effective technique, based on the genetic algorithm, for generating non-photorealistic rendering of images. Genetic and evolutionary algorithms\textsuperscript{1,2} are a class of population-based stochastic heuristics that have been applied to numerous search and optimization problems in science, engineering and other fields\textsuperscript{3,4}. These algorithms are popular for their ability to outperform traditional optimization techniques on difficult problems.

Non-photorealistic rendering (NPR) is an active field of research in modern computer graphics and visualization. While traditional computer graphics mainly focused on synthesizing photorealistic images, NPR has recently emerged as a better alternative in many applications with its notable advantages over sheer photorealism. NPR basically allows for various ways and styles in interpreting or synthesizing
a scene, giving room for flexibility, creativity, and expressiveness in scene rendering. NPR is also capable of quickly directing the viewer’s attention to important features or objects in the scene through various levels of simplification or exaggeration, which makes it useful for effective visual communication, feature-based visualization/simplification, and data compression. NPR is now widely used for generating special effects in movies and games, or for creating feature-based illustrations of live characters, natural scenes, medical structures, commercial products, artificial ornaments, etc. In many applications, NPR problems use one or more reference images to create non-photorealistically rendered output images. Most of the image-based NPR techniques are also termed stroke-based, where a brush (or pen) stroke is used as the basic element for constructing a picture. In computerized stroke-based rendering, in general the goal is to direct a computer to automatically distribute a set of brush strokes to create an abstracted yet visually pleasing output image that is sufficiently close to the input image. In particular, our focus in this paper is on painterly rendering, such as is found in oil paintings or watercolor paintings, which may be regarded as one of the oldest and the most natural forms of human art. Not surprisingly, this has been one of the most widely studied subjects in the area of NPR research. Compared to other NPR styles, painterly rendering uses a large number of relatively rough and thick brush strokes to compose an artistic picture, and thus inherently involves a high degree of fuzziness in the decision making process, which makes it particularly challenging. An example of computer-generated painterly rendering is given in Figure 1.

For more than a decade or so, a variety of techniques have been proposed for generating painterly renderings from digital images. Haeberli introduced the first painterly rendering system, which was based on the user’s interactive stroke placements on a given input image. Litwinowicz proposed an automatic approach to
distribute strokes uniformly on a canvas by successively filling in the holes detected by Delaunay triangulations. Hertzmann presented a multi-level painting algorithm using curved strokes of multiple sizes. Shiraishi and Yamaguchi developed a novel stroke distribution algorithm based on image moment function and space-filling curves. Gooch et al. employed computer vision techniques such as image segmentation and medial axis algorithm to place strokes along salient image features. Hays and Essa proposed an effective stroke-orienting technique based on radial basis functions. While each of these approaches successfully produces its own distinct rendering style, none of them aims particularly at optimizing the rendering quality. In a way, these algorithms can be classified as greedy methods, where each stroke is placed on the presumably ideal location using some criteria, but a stroke’s placement, once determined, cannot be changed later in the algorithm.

On the other hand, Hertzmann proposed a painting algorithm specifically for optimizing the quality of the painting. In his formulation, the rendering problem is equivalent to minimizing an energy function defined (in a simplified form) as follows:

\[ E(I') = w_d \cdot E_d(I') + w_s \cdot E_s(I') \]  

where \( E_d = \sum_x \| I'(x) - I(x) \| \) represents the sum of differences between the input image \( I \) and the painting \( I' \) measured at each pixel \( x = (x, y) \), and \( E_s \) denotes the number of strokes used in the painting, controlling the level of abstraction (fewer strokes generally result in a more abstract and artistic painting). Their combination weights are denoted \( w_d \) and \( w_s \), respectively. Thus, the main goal consists of two conflicting subgoals, that is, to generate a rendering as close to the input picture as possible while using a minimum number of strokes. In Hertzmann’s optimization algorithm, the strokes are placed at random pixel locations and their positions are subsequently adjusted over a number of iterations, in an attempt to minimize the value of the objective function. The adjustment operators include random stroke addition, deletion, move, resizing, and recoloring, and they are conducted in a trial-and-error fashion by monitoring their effects on the objective function. While this algorithm indeed succeeds in producing paintings of high quality, such a random-descent process could easily result in a local minimum solution.

In our preliminary work, we suggested that the rendering quality can be further optimized by employing an evolutionary algorithm. It is motivated by the fact that an image-based painterly rendering problem generally involves a high dimensional space with many local minima, where it is hard to find an analytic solution. Collomosse and Hall presented an effective painting algorithm based on genetic search, which is guided by the pre-extracted feature image called salience map. Their algorithm aims at incorporating the ‘subjective’ feature interpretation of an individual user, and thus the salience map is trained from the interactive user input.

In this paper, we present a completely automatic, multi-level painting algorithm based on evolutionary computing. More specifically, the proposed method develops a multi-level evolutionary approach to manipulating variable-dimensional chromo-
U. K. Chakraborty, H. W. Kang, and P. P. Wang

somes representing dynamically growing/shrinking sequences of brush strokes. As will be shown in our experimental results, the quality of painting can be improved in a statistically meaningful way by conducting the optimization in multiple levels (or layers). That is, our method produces lower objective function values than the previous methods, given the same number of strokes. It also allows the user to choose the best solution out of multiple high-quality solutions (rendered strokes) provided by the evolutionary process.

The remainder of this paper is organized as follows. Section 2 defines the problem and Section 3 presents our multi-level evolutionary algorithm for painterly rendering. In Section 4, we show some experimental results and provide comparisons with two previously published techniques. We conclude this paper in Section 5.

2. The problem

The problem under study can be defined as follows: Given an input image, obtain a painterly rendering by using brush strokes. Specifically, given the reference image, place the strokes on a white canvas so that the fitness function (Eq. (3.2), see later in Section 3.3) is minimized. A stroke is characterized as a curved object with properties (parameters) such as position, color, path, width and length (see Figure 2). In our current implementation, we optimize the position variables \((x, y)\), the color variables \((\text{red, green, blue})\), and the stroke size \((\text{width, length})\), and we compute the stroke path from the input data (image). In addition, we allow the use of stroke texture and opacity, which may be provided by the user, to enhance the expressiveness of the painting. Strokes are placed on the canvas (white image) one by one to simulate the painting process.

In our approach, the strokes are classified into multiple levels (layers), where each level contains a set of strokes, with the stroke size being inversely proportional to the level number. That is, large strokes are placed at the low levels, and small strokes are at the high levels. The complete painterly rendering is constructed by combining all of these layers, placing the lowest layer in the back and successively placing the next higher layers in front (see Figure 3). Note that this model is

\(^a\)In the figure, all strokes are drawn as rectangles for simplicity.
consistent with the conventional painting process of real artists, where they work in multiple levels of details, starting from the coarsest level (background layer) using a large brush and progressively touch up the foreground details with smaller brushes\(^9\). Also note that there can be many overlapping strokes as in real paintings. Thus, the ordering of strokes does matter as it affects the quality of the painting.

3. The proposed method

A multi-level evolutionary algorithm with a variable-length chromosome and several problem-specific mutation operators is used to obtain a near-optimal painting. The algorithm is outlined below (\(t\) denotes the generation number):

\[
\begin{align*}
t &= 0; \\
\text{initialize population}(t); \\
\text{while (acceptable solution not found and } t < \text{ maximum-gen) } \{ \\
\text{ \quad repeat } \{ \\
\text{ \quad \quad select one individual from population}(t); \\
\text{ \quad \quad select another individual from population}(t); \\
\text{ \quad \quad for each level}(i) \\
\text{ \quad \quad \quad cross them in section}(i) \text{ to produce two individuals of population}(t+1); \\
\text{ \quad \quad } \} \text{ until population}(t+1) \text{ is filled}; \\
\text{for each individual of population}(t+1) \{ \\
\text{ \quad for each level}(i) \{ \\
\text{ \quad \quad apply mutate-delete on section}(i); \\
\text{ \quad \quad apply mutate-reverse on section}(i); \\
\text{ \quad \quad apply mutate-perturb-position on section}(i); \\
\text{ \quad \quad apply mutate-perturb-size on section}(i); \\
\text{ \quad \quad apply mutate-perturb-color on section}(i); \\
\text{ \quad \quad apply mutate-add on section}(i); \\
\text{ \quad } \} \\
\text{ } \} \\
\text{ } \} \\
\text{ } t = t + 1; \\
\} 
\end{align*}
\]
3.1. Chromosome representation

Genetic algorithms require an encoding (representation) of the candidate (trial) solutions. The encoding affects the design of the crossover and mutation operators. A painting is encoded by a chromosome that has a number of strokes arranged in a certain order. Especially, multiple levels of strokes are employed. The strokes at the beginning levels are larger (i.e., wider and longer), covering more of the canvas, while the later levels have progressively smaller stroke sizes. 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 all the levels (see Figure 4). 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. The strokes of a particular section have similar sizes, while the strokes of the next section (upper level) have lower sizes. Each section has a pre-determined range of stroke size, which is inversely proportional to the level number. 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.

A single stroke is represented by its 2-dimensional coordinates \((x, y)\), the color components \((r, g, b)\), and the size components \((w, l)\). 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 \(w\) and \(l\) are integers in the range of \([w_{\text{min}}(\text{level}), w_{\text{max}}(\text{level})]\) and \([l_{\text{min}}(\text{level}), l_{\text{max}}(\text{level})]\), respectively (\text{level} denotes the level number). In most of our experiments we set \(w_{\text{min}}(x) = 2^{(5-x)} + 2\), \(w_{\text{max}}(x) = 2^{(6-x)} + 2\), \(l_{\text{min}}(x) = 20 \cdot (5 - x) + 5\), and \(l_{\text{max}}(x) = 20 \cdot (6 - x) + 5\). The strokes of a chromosome are implemented as elements of a dynamically grow-

![Fig. 4. Chromosome representation](image-url)
ing/shrinking array. The ordering of the elements of the array determines the sequence of strokes (which stroke is “below” or “above” which other stroke).

3.2. Population initialization
The initial population is created by giving each chromosome a random number of strokes between two limits, MIN_STROKES/3 and MAX_STROKES/3. This is done by initializing each section of each chromosome with a random number of strokes between MIN_STROKES/(3*5) and MAX_STROKES/(3*5). The (x, y) coordinates of each stroke of each section are initialized uniformly randomly in the interval [0,1]. Alternatively, one may choose to use some pre-computed probability at each pixel to scatter the strokes non-uniformly, by exploiting the configuration of the input image (see Section 3.6.2 for details), which could contribute to a significant speed-up in the subsequent optimization. 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 (Section 3.6.4). Each of the w and l components of each stroke is randomly set in the predefined range of the corresponding level.

3.3. Fitness
The following fitness function is used:

\[ f(I') = (1 - 1/N_s) \sum_x ||I'(x) - I(x)|| \]  

where I and I’ represent, respectively, the input and the output images, and N_s denotes the number of strokes. The multiplying factor is used to reward chromosomes with a fewer number of strokes. The problem is one of minimization. Here we do not distinguish between the fitness function and the objective function. No fitness scaling is used.

3.4. Selection
Binary tournament selection with replacement is used. In each selection, two uniformly randomly chosen individuals are compared between themselves (in terms of their fitness values), and the one with a better (lower) fitness is the winner. Binary tournament selection is used as it is the simplest to implement. We did not investigate if proportional or rank-based selections would yield better results.

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.

3.5. Crossover
For each section, we use single-point crossover (head-tail swap at a single cut-point) to produce two offspring sections from two parent sections. The cut-point is chosen
Suppose two parents $P_1$ (of length $L_1$) and $P_2$ (of length $L_2$) are crossed at $R$ = random $(1, \min(L_1, L_2))$ to produce two children $C_1$ and $C_2$ (see Figure 5). Then $C_1$ is of length $L_2$, with the first $R$ strokes copied from the first $R$ strokes of $P_1$, and the remaining $L_2 - R$ strokes copied from the last $L_2 - R$ strokes of $P_2$. Similarly, $C_2$’s first $R$ strokes come from $P_2$ and the last $L_1 - R$ strokes from the corresponding strokes of $C_1$. Note that the crossover occurs only between the same sections of the two parents, and crossover at one section does not affect any other section.

3.6. Mutation

Four different problem-specific mutations (Sections 3.6.1—3.6.4) are used. In each of these mutations, worse mutants are accepted with a low probability. The following pseudocode explains how to decide whether the result of the application of mutation is to be accepted:

\[
\text{if the mutated individual has a better fitness than the current one} \\
\text{then accept the mutated individual as the current individual} \\
\text{else accept the mutated individual as the current individual with} \\
\text{a small, predetermined probability.}
\]

3.6.1. Delete strokes

For each section, both the total number of strokes to be deleted from a given section and the individual strokes (to be deleted) are chosen randomly. The total number of strokes to be deleted is determined as a random integer in $[0, x]$ where $x = \max(\rho_d \cdot (\text{current section length} - \text{MIN_STROKES}/5), 0)$, $\rho_d$ being a predetermined fraction. The individual deletions in a single mutate-delete operation are applied one after another, following the above-mentioned accept/reject policy at each deletion.
3.6.2. Add strokes

A random number of new strokes are added at the end of a 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 (Section 3.6.4). 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 in $[0,x]$ where $x = \max[\rho_a \cdot (\text{MAX}\_\text{STROKES}/5 - \text{current}\_\text{section}\_\text{length}), 0]$, using a predetermined fraction $\rho_a$.

To expedite the optimization process, we also propose an alternative data-driven stroke insertion policy. In general, artists place brush strokes more densely (and carefully) in an important area. The ‘importance’ of an area is often measured by the strength of the feature outlines in the area. Large brushes normally take care of rough object interiors, and thus they are applied all over the canvas, ignoring the feature outlines. On the other hand, smaller brushes handle the fine details of an object, and hence should be placed near the strong outlines. Based on this idea, we associate each pixel with a probability of accepting a stroke placement on it. First, we obtain an edge map using the standard gradient-based method, and then create a distance field where each pixel records the minimum distance to the nearest edge. Let $d(x)$ denote the distance at $x = (x, y)$, then the probability $p_s(x,k)$ of placing a stroke at $x$ in level $k$ is defined as follows:

$$p_s(x,k) = \tanh[\nu(k) \cdot (1 - d(x)/\max y d(y))]$$ (3.3)

where $\nu(k) = 5.0/k$ in our experiment. That is, in low levels, the probabilities are high in most of the pixels, whereas in upper levels, the probabilities are high only near the strong outlines. This strategy allows for a spatially non-uniform stroke distribution in each level and expedites the optimization process by disallowing a large number of unnecessary stroke placements.

3.6.3. Reverse segment

Two randomly chosen strokes in a given section are used to define a segment of strokes in a chromosome, and the strokes in the segment are reversed (see Figure 6).

3.6.4. Perturb strokes

For each section, three types of perturbations are applied to a stroke — position perturbation, size perturbation, and color perturbation, the three perturbations being independent of each other. Each stroke in a section 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 size components (width, length) of a strokes are perturbed, probabilistically, and independently of the $x$- (or $y$-) perturbations. With a predetermined size-perturb probability, a stroke is chosen for size perturbation and each of its two components is given a random amount of positive or negative perturbation. The three color components of a stroke are also perturbed similarly, using a predetermined color-perturb probability. The perturbations are guaranteed not to result in out-of-bound values.

3.7. Orienting strokes

The orientation of each stroke is derived from the input image $I(x, y)$. The governing factor of the stroke direction is the feature outlines nearby. The outlines normally follow the image tangent vector $\nabla I^\perp$ (perpendicular to the image gradient $\nabla I = (\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})$), and its influence is proportional to the vector magnitude, denoted $|\nabla I^\perp|$. However, many image pixels often form small textures or a homogeneous region and thus their tangent vectors may be unrelated to those of the nearby ‘dominant’ outlines, or their magnitudes may be so small that their directions are meaningless. Thus, to force the direction vectors to follow the flow of the dominant outlines, we employ a variational formulation.

The stroke direction field, denoted $\mathbf{v}(x, y) = [u(x, y), v(x, y)]$, is obtained by minimizing the following energy functional:

$$E(\mathbf{v}) = \iint |\nabla I^\perp|^2 |\mathbf{v} - \nabla I^\perp|^2 + \mu (u_x^2 + u_y^2 + v_x^2 + v_y^2) dxdy$$  \hspace{1cm} (3.4)$$

where $\mu$ is a regularization parameter. For a pixel where $|\nabla I^\perp|$ is large, its direction vector $\mathbf{v}$ is more affected by the first term and thus tends to resemble the original tangent vector. That is, the dominant outline directions are preserved. When $|\nabla I^\perp|$ is small, its $\mathbf{v}$ is more affected by the second term and adjusts its direction according to the neighboring vectors to create a smoothly varying field around it. Based on calculus of variations, the above energy functional is minimized by solving the
following Euler equations:

\[ \mu \nabla^2 u - (u + \frac{\partial I}{\partial y})(\frac{\partial I}{\partial x})^2 + (\frac{\partial I}{\partial y})^2 = 0 \]  \hspace{1cm} (3.5)

\[ \mu \nabla^2 v - (v - \frac{\partial I}{\partial x})(\frac{\partial I}{\partial x})^2 + (\frac{\partial I}{\partial y})^2 = 0 \]  \hspace{1cm} (3.6)

where \( \nabla^2 \) is a Laplacian operator. These equations are solved numerically, starting from the initial \( v \) (set as \( \nabla I^\perp \)), and terminating when a steady state is reached. The resulting vector field is used to guide the curved stroke directions in all levels. Fig. 7 illustrates stroke direction fields obtained from some sample images.

Fig. 7. Example stroke direction fields

### 3.8. Parameter settings

Table 1 shows the parameter settings used for the results presented in Section 4. These values were found to produce reasonably good first results; no systematic parameter tuning was attempted.

### 3.9. Termination condition

In this problem it is hard to have a well-defined termination criterion. The global optimum is not known in advance, nor do we need to find it to arrive at an acceptable result. The quality of the rendered image is to a great extent a matter of human perception. Eq. (3.2) 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.
4. Experimental results

We conducted experiments on a number of input images of various sizes to compare the rendering results of our method against a random descent method and a single-level evolutionary algorithm-based method. For the random descent method, an iterative process of random stroke insertion, deletion, move, resizing, and recoloring 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. For the single-level evolutionary algorithm, we applied the same set of operations as our multi-level version, except that only a single layer is used in this case. We show three representative images in Figure 8, in which only the best result out of five independent runs, for each of the three algorithms, is displayed. Table 2 shows, for each of the three images (Flowers, Venus, and Mountain), the mean and standard deviation of the fitness values for five independent runs of each of the three methods. As shown in Figure 8 and Table 2, our method outperforms the other two approaches on three counts: (a) fitness value (the smaller the fitness value, the better), (b) fitness standard deviation (a smaller standard deviation indicates a more consistent method) and (c) the (arguably subjective) visual quality of the rendered images. For the same number of strokes, our method produces smaller (better) fitnesses for all the input images. The consistently lower standard deviations of our method show that this is clearly a better strategy.

To investigate whether the differences in performance can be attributed to chance, unpaired $t$-tests are performed. Results of unpaired $t$-tests are presented in Tables 3 (random descent versus the proposed method) and 4 (single-level EA
versus the proposed method). We test the null hypothesis $H_0 : \mu_1 - \mu_2 = 0$ against the one-sided alternative $H_1 : \mu_1 - \mu_2 > 0$ where $\mu_2$ is the mean fitness of the
proposed method and $\mu_1$ that of the competitor. We choose a level of significance of 0.01. The sample size is five in every case. Since we have no reason to believe that the sample variances are the same, we cannot use the standard two-sample $t$-test. Therefore, we use the Smith-Satterthwaite test$^{19}$ corresponding to unequal variances. The test statistic is given by

$$t_{-\text{statistic}} = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

and its sampling distribution can be approximated by the $t$-distribution with

$$\frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{(s_1^2/n_1)^2}{n_1-1} + \frac{(s_2^2/n_2)^2}{n_2-1}}$$

degrees of freedom (rounded down to the nearest integer), where $\bar{x}_1 = \text{sample mean 1}$, $\bar{x}_2 = \text{sample mean 2}$, $s_1 = \text{sample standard deviation 1}$, $s_2 = \text{sample standard deviation 2}$, $n_1 = \text{sample size 1}$, $n_2 = \text{sample size 2}$.

Tables 3 and 4 present, for each image, the following:

- standard error of the difference of the two means
- $t$-statistic
- degrees of freedom (df)
- the $t$ value (obtained from standard tables of $t$-distribution) at 99% (i.e., right-tail probability of 0.01) and for the specified degrees of freedom, df
- 99% confidence interval for the difference of the two means
- one-tailed $p$-value (probability obtained from standard tables) for the specified degrees of freedom at $t = \text{the computed value of } t$-statistic

The results show that for each image and each comparison, the $t$-statistic exceeds $t_{0.01}$ for the relevant degrees of freedom. Therefore the null hypothesis is rejected in all of these cases. Again, none of the confidence intervals contains zero. This

<table>
<thead>
<tr>
<th>image</th>
<th>method</th>
<th># of strokes</th>
<th>fitness value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>mean</td>
</tr>
<tr>
<td>Flowers</td>
<td>Random Descent</td>
<td>1350</td>
<td>3603852.4</td>
</tr>
<tr>
<td></td>
<td>Single-level EA</td>
<td>1350</td>
<td>2634114.7</td>
</tr>
<tr>
<td></td>
<td>Proposed Method</td>
<td>1350</td>
<td>2135417.8</td>
</tr>
<tr>
<td>Venus</td>
<td>Random Descent</td>
<td>1250</td>
<td>2006320.7</td>
</tr>
<tr>
<td></td>
<td>Single-level EA</td>
<td>1250</td>
<td>1532160.3</td>
</tr>
<tr>
<td></td>
<td>Proposed Method</td>
<td>1250</td>
<td>1315364.7</td>
</tr>
<tr>
<td>Mountain</td>
<td>Random Descent</td>
<td>1450</td>
<td>2146665.8</td>
</tr>
<tr>
<td></td>
<td>Single-level EA</td>
<td>1450</td>
<td>1594201.1</td>
</tr>
<tr>
<td></td>
<td>Proposed Method</td>
<td>1450</td>
<td>1360634.2</td>
</tr>
</tbody>
</table>

Table 2. Comparison of the three methods
allows us to conclude that the improvements produced by the proposed method are statistically significant.

Figure 9 and Table 5 show the performance of our method on input images of three different sizes. Even with an increase in the number of pixels (and a corresponding increase in the search space size), our method continues to generate high-quality results. The final column (ave. diff.) in Table 5 shows that the average pixel-wise color differences (difference/pixel) are consistent regardless of the image size. Figure 10 shows the effect of using texture and opacity attributes. In each row, the brush texture map used is shown in the upper-right corner. The brightness of each pixel in the map indicates the pixel opacity. Thus, each pixel color is blended with the corresponding pixel color of the strokes lying underneath, with the blending weights computed from the opacity value (the lower the opacity, the higher the blending weight that goes to the background color). In the first row, an irregular texture pattern with constant white color is used, to create a rough oil painting effect without much blending. The second row shows a smooth watercolor effect obtained from a brush texture with smoothly and regularly distributed opacity values. Our system is implemented on a PC equipped with 3 GHz CPU, 1 GB Memory, and GeForce® FX 5950 GPU. In our current implementation, the rendering time varies widely, ranging from tens of minutes to several hours, mainly depending on the population size, the image size and its visual complexity.

<table>
<thead>
<tr>
<th>Image</th>
<th>Result</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flowers</td>
<td>standard error of difference</td>
<td>0.6782.583</td>
</tr>
<tr>
<td></td>
<td>t-statistic</td>
<td>21.890</td>
</tr>
<tr>
<td></td>
<td>degree of freedom</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>t at .01 and df = 4</td>
<td>3.747</td>
</tr>
<tr>
<td></td>
<td>99% confidence interval of difference of means one-tailed p value</td>
<td>1159586.388 to 1777282.812 less than 0.0001</td>
</tr>
<tr>
<td>Venus</td>
<td>standard error of difference</td>
<td>1.2759.779</td>
</tr>
<tr>
<td></td>
<td>t-statistic</td>
<td>13.141</td>
</tr>
<tr>
<td></td>
<td>degree of freedom</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>t at .01 and df = 4</td>
<td>3.747</td>
</tr>
<tr>
<td></td>
<td>99% confidence interval of difference of means one-tailed p value</td>
<td>448878.699 to 933033.301 less than 0.0001</td>
</tr>
<tr>
<td>Mountain</td>
<td>standard error of difference</td>
<td>40479.676</td>
</tr>
<tr>
<td></td>
<td>t-statistic</td>
<td>19.418</td>
</tr>
<tr>
<td></td>
<td>degree of freedom</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>t at .01 and df = 4</td>
<td>3.747</td>
</tr>
<tr>
<td></td>
<td>99% confidence interval of difference of means one-tailed p value</td>
<td>599663.174 to 972400.026 less than 0.0001</td>
</tr>
</tbody>
</table>

Table 3. Results of unpaired t-tests (random descent vs. proposed method)

5. Conclusion and Discussion

We have presented a novel method for optimizing the rendering quality for painterly rendering, based on an evolutionary algorithm that uses new, problem-specific mutation operators and a variable-dimensional chromosome. Our method outperforms
Input: Peppers

300 × 200 400 × 267 500 × 334

Fig. 9. Rendering results for images of different sizes

(a1) Input image: Butterfly  (a2) Output image

(b1) Input image: Desert  (b2) Output image

Fig. 10. Effect of texture and opacity
some known approaches such as a simple random descent method or a single-level evolutionary algorithm, approaching the global optimum solution for the problem more closely. Unpaired t-tests also proved our method to be statistically significantly better. Thus we believe our approach is likely to be used as a framework for further research in this area. It should be noted, however, that we are comparing our method with other published methods in terms of their underlying ‘principles’ in their raw forms, but not their ‘entire rendering procedures’. In fact, it is difficult to make a completely accurate or fair comparison between different painterly rendering algorithms, since in general they all have somewhat different goals and objective functions, and each algorithm is supplemented by various distinct image processing techniques and heuristics for further enhancements in quality and efficiency, all of which have been disregarded in our study. It should also be noted that none of the above-mentioned algorithms is guaranteed to converge to global optimal solutions, and thus they are all ‘weak’ optimization methods.

We are currently extending the work to design better fitness functions capable of capturing the rendering quality more accurately. It also remains to be explored whether the use of other selection schemes\textsuperscript{18} would improve performance. The preference for a lower stroke count can be made stronger by, for example, squaring the multiplying factor in Eq. (3.2). We believe our approach will be applicable to other graphics-related problems as well, such as non-photorealistic scene navigation\textsuperscript{20,21},

<table>
<thead>
<tr>
<th>Image</th>
<th>Result</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flowers</td>
<td>standard error of difference</td>
<td>33534.087</td>
</tr>
<tr>
<td></td>
<td>t-statistic</td>
<td>14.871</td>
</tr>
<tr>
<td></td>
<td>degree of freedom</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>t at .01 and df = 5</td>
<td>3.365</td>
</tr>
<tr>
<td></td>
<td>99% confidence interval of difference of means</td>
<td>363487.461 to 633906.340</td>
</tr>
<tr>
<td></td>
<td>one-tailed p value</td>
<td>less than 0.0001</td>
</tr>
<tr>
<td>Venus</td>
<td>standard error of difference</td>
<td>23638.638</td>
</tr>
<tr>
<td></td>
<td>t-statistic</td>
<td>9.171</td>
</tr>
<tr>
<td></td>
<td>degree of freedom</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>t at .01 and df = 6</td>
<td>3.143</td>
</tr>
<tr>
<td></td>
<td>99% confidence interval of difference of means</td>
<td>129167.169 to 304424.032</td>
</tr>
<tr>
<td></td>
<td>one-tailed p value</td>
<td>less than 0.0001</td>
</tr>
<tr>
<td>Mountain</td>
<td>standard error of difference</td>
<td>5932.271</td>
</tr>
<tr>
<td></td>
<td>t-statistic</td>
<td>39.372</td>
</tr>
<tr>
<td></td>
<td>degree of freedom</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>t at .01 and df = 6</td>
<td>3.143</td>
</tr>
<tr>
<td></td>
<td>99% confidence interval of difference of means</td>
<td>211575.972 to 255557.828</td>
</tr>
<tr>
<td></td>
<td>one-tailed p value</td>
<td>less than 0.0001</td>
</tr>
</tbody>
</table>

Table 4. Results of unpaired t-tests (single-level EA vs. proposed method)

<table>
<thead>
<tr>
<th>Image</th>
<th>size</th>
<th>fitness value</th>
<th>ave. diff.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peppers</td>
<td>300 × 200</td>
<td>1363324.2</td>
<td>22.72</td>
</tr>
<tr>
<td></td>
<td>400 × 267</td>
<td>2277514.8</td>
<td>21.32</td>
</tr>
<tr>
<td></td>
<td>500 × 334</td>
<td>3271442.8</td>
<td>19.60</td>
</tr>
</tbody>
</table>

Table 5. Results with varying image sizes
mesh optimization, streamline visualization, texture synthesis, etc. In regards to the time efficiency, since the graphical scan-conversion process for evaluating the fitness value remains a huge bottleneck, a GPU-based implementation of the algorithm would significantly speed up the mutation operations.

Acknowledgments

Thanks to the anonymous referees for their comments and suggestions. This research was partly supported by UM Research Board and the MIC (Ministry of Information and Communication), Korea, under ITRC (Information Technology Research Center) support program supervised by IITA (Institute of Information Technology Assessment).

References