A Unified Scheme for Adaptive Stroke-Based Rendering

Hyung W. Kang, Charles K. Chui, and Uday K. Chakraborty

University of Missouri - St. Louis
Department of Mathematics and Computer Science
One University Blvd. St. Louis, MO 63121, USA
Phone: 314-516-5841, Fax: 314-516-5400
{kang,chui,uday}@arch.umsl.edu

Abstract

This paper presents a comprehensive scheme for automatically generating a broad class of artistic illustrations from photographs. Using strokes as the major building blocks, our system optimizes the stroke attributes subject to the desired rendering style. The stroke attributes are computed adaptively to enable importance-based control of the abstraction level at each pixel. We propose a novel outline detection and refinement paradigm called 'edge painting' to construct an outline map, and from which to derive the pixel-wise importance. We also introduce an 'adaptive bilateral filter' to adaptively guide the curved stroke directions based on the importance map. Given the outline, importance, and direction maps, the system creates the illustration via selecting the representative colors, setting the style parameters, and optimizing the stroke attributes based on simulated annealing. The experimental results show that our scheme facilitates automatic production of artistic illustrations in a wide range of rendering styles.

Keywords: Non-photorealistic rendering, Stroke-based rendering, Adaptive edge detection, Adaptive bilateral filter, Simulated annealing

1. Introduction

Automatic generation of artistic illustrations from photographs is one of the most fundamental and actively studied problems in the field of non-photorealistic rendering (NPR). In general, a good illustration depicts the target objects or scenes in such a way that all the extraneous details are simplified or removed, while the salient features are preserved or emphasized. However, there are diverse ways to achieve this goal. In fact, a wide variety of illustrations can be generated even from a single photograph, depending on the rendering styles, simulated artistic media, types of features to preserve, and the desired levels of abstraction.

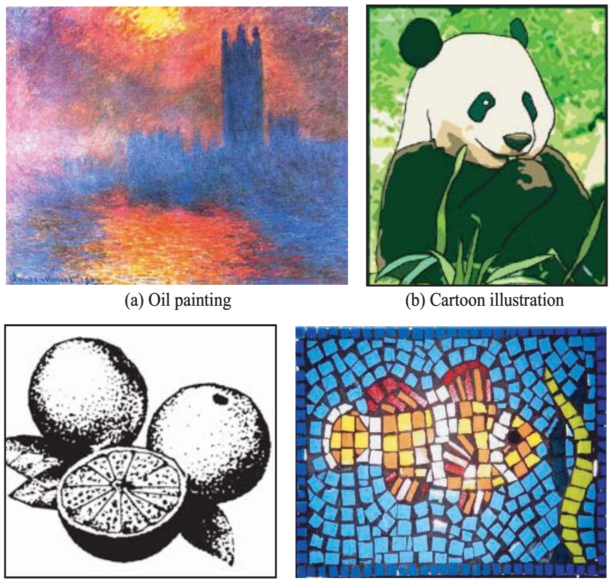

(a) Oil painting (b) Cartoon illustration (c) Stipple drawing (d) Tile mosaic

Figure 1. Various styles of illustrations

Fig. 1 shows some example illustrations manually created by artists, suggesting that a wide variation of artistic styles are available in producing illustrations. Notice the effective use of feature outlines in Fig. 1(b)-(d). Also, these examples show that 'less important' regions may be purposefully abstracted, to communicate 'important' features to the viewers more effectively. Although there have been numerous NPR approaches for automatic generation of illustrations [24, 2, 21, 15, 22, 26, 5, 27, 16, 13, 11, 4, 14], none of them is general enough to produce alone all of these example rendering styles. When we take into account the comprehensive class of artistic illustrations, the following statements can be made in general:

- First, artistic illustrations can widely vary in their rendering styles.

- Second, *line drawings* of the outlines often play important roles such as preserving/emphasizing salient features, enabling higher level of abstraction, and providing more effective communication. Also, line drawings can be easily mixed with other renderings to enhance the artistic styles.

- Third, the degree of abstraction may be adaptive (position-dependent), based on the 'importance' of the area. Here, the importance may be determined by the inherent complexity of the region, as well as the artist's insight or taste. All important features need to be preserved by minimizing the abstraction around them.

In this paper, we propose a novel scheme for automatic generation of artistic illustrations from photographs, which meets all of the requirements stated above. It is a unified scheme in that it enables the production of a broad class of stroke-based illustrations, including traditional oil painting, watercolor painting, cartoon illustration, line drawing, pen-and-ink illustration, stipple drawing, tile mosaics, etc. This paper makes several contributions to the study of stroke-based rendering (and NPR in general). In particular, we present: (1) a generalized scheme for producing a wide range of illustrations, (2) a novel technique for adaptive outline detection and refinement, (3) an adaptive level-of-abstraction control method based on pixel-wise *importance*, (4) an *adaptive bilateral filter* for deriving an adaptive stroke direction map, (5) a novel automatic color selection algorithm to support a wide range of styles, and (6) a generalized stroke optimization algorithm based on *simulated annealing*.

1.1 Related Work

There have been a number of techniques proposed in the field of NPR, for automatically creating artistic renderings from arbitrary reference photographs [24, 2, 21, 15, 22, 26, 27, 5, 16, 13, 11, 4, 14]. Each of these techniques is designed to simulate a distinct artistic style and specific artistic medium such as oil painting, pen-and-ink illustration, watercolor painting, pencil drawing, stipple drawing, engraving, mosaics, stylized abstraction, etc. However, the 'range' of styles and media supported by each technique is quite limited. While Hertzmann et al. [18] showed that texture synthesis can be used to simulate various styles, it requires sample input and output (filtered) images to be provided, and the control on the rendering result is indirect and restrictive. Moreover, texture-synthesis-based approach is unsuitable for certain types of illustrations where the outlines or stroke shapes must remain coherent and accurate, such as line drawing, cartoon illustration, or mosaics.

Many of the automatic illustration techniques are stroke-based, in that the basic elements for constructing the picture are *strokes* [17]. However, the strokes have been mainly used to fill the interior of regions, while the *outlines* (region boundaries) are often neglected. While there are some approaches that allow partial use of outlines to enhance their styles [5, 13, 4], their underlying frameworks are still not general enough to cover various artistic styles or media. In our approach, the outline information serves as an essential ingredient which plays two important roles. **First**, from the outline maps we produce line drawings, which may be further mixed with other interior stroke renderings, broadening the range of styles. **Second**, the outline map is used to derive key stroke attributes (such as width, length, orientation, etc.) 'adaptively', that is, in such a way that all unimportant regions are abstracted out while important features are preserved.

Typically, the 'importance' of the area is computed based on the inherent 'complexity' of the regional pixel color distribution [15, 26, 11]. Thus, a noisy or highly textured area is automatically considered important and assigned small strokes, although the artist may disagree on the actual importance of that area. For example, when illustrating a bird, the tiny, complex pattern of individual feathers may not be considered important by some artists. There are recent approaches supporting interactive region masking [16, 25], enabling users to assign the pixel-wise importance in an arbitrary fashion. These masking tools, however, are based on either manual control or specialized eye-tracking hardware. In this paper, we propose a novel semiautomatic technique called *edge painting*, which enables the selective construction and refinement of outlines, thus provides an effective way to control the regional importance (i.e., the level-of-abstraction) adaptively and arbitrarily.

1.2 Overview

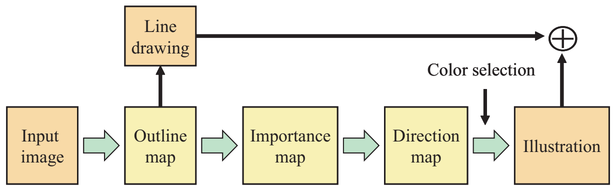


Figure 2. System overview

Fig. 2 shows the block diagram of our illustration system. As a preprocess, we perform automatic edge detection on the input photograph to construct an outline map, which may be further refined by our edge painting paradigm. The remaining processes are fully automatic. Given the outline map, we derive an importance map, which is then used to obtain the stroke direction field. In the next step, the system selects a certain number of representative colors to comprise the illustration. The final step is stroke rendering, which is essentially a process of determining the stroke attributes, guided by the importance and direction maps. Optionally, the outline drawing may be merged with the illustration to enhance the artistic style.

2. Outline Map Construction

We use Canny edge detector [1] to obtain the outline map (or edge map), denoted $E(\mathbf{x})$, which is a binary map where $E(\mathbf{x})$ is 0 if $\mathbf{x}$ is an edge pixel, and 1 otherwise. Canny's method, and other gradient-based variants, typically consist of three basic steps: (1) gradient estimation, (2) nonmaxima suppression, (3) hysteresis thresholding. Step (1) uses a scale parameter σ for Gaussian smoothing of input image I, that is, $I * G_\sigma$. Step (3) requires two gradient magnitude values g_h and g_l for two-level thresholding. These parameters are fine-tuned to construct a desirable outline map.

However, it is known that the global application of fixed parameters often results in an incomplete outline map with many missing outlines and/or insignificant edges, which makes it unsuitable as the source of line drawing or pixelwise importance derivation. While it is possible to reduce missing outlines by computing the parameters for each pixel adaptively based on local image statistics [8], the resulting outline map tends to be overly complex, and thus still may not agree with the viewer's own interpretation or insight. To overcome this limitation, we present a novel outline construction paradigm called *edge painting*, which facilitates the selective edge detection and refinement via local and adaptive control of the parameters.

2.1 Edge Painting

Edge painting is essentially a *local*, *incremental Canny edge detector*, employing a user-guided local window in which the outline map is incrementally updated. Following the user's cursor movement, the window sweeps the region of interest, and Canny edge detection is applied with the current set of parameters in the current window. This incrementally updates the edge map of the swept area, and thus the steerable local window operates like a 'painting brush' in the painting software. The size of the window, denoted η, is adjustable, and thus we have four parameters (σ, g_h, g_l, η) that may be adjusted between painting operations. We may add a circular *display kernel* inscribed in the window, so that only the contents in this circle are displayed on the screen, creating smooth 'painting brush' effect (see Fig. 3).

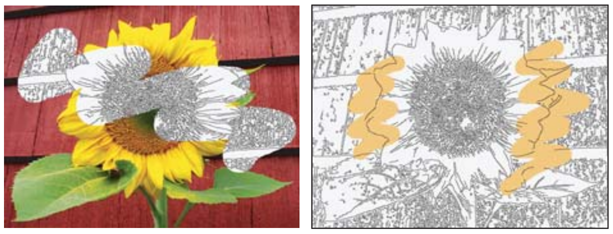

(a) Direct application (b) Edge map editing

Figure 3. Edge painting

Algorithm 1 Edge painting

```
1:  $i := 0$
2:  while mouse button down do
3:    if cursor moved or $i = 0$ then
4:      $w_i$ := window centered at current cursor location
5:      $\hat{w}_i := w_i - \Omega(w_i)$
6:      $\Phi(w_i) := \Omega(w_i) \cap \bigcup_{j=0}^{i-1} w_j$
7:      Estimate gradients in $w_i$
8:      Find maxima pixels in $\hat{w}_i$
9:      Mark maxima pixels $\mathbf{x}$ in $\hat{w}_i$, if $g(\mathbf{x}) > g_h$
10:     Mark maxima pixels $\mathbf{x}$ in $\Phi(w_i)$, if $g(\mathbf{x}) > g_l$
11:     Mark maxima pixels $\mathbf{x}$ in $\hat{w}_i$ if $g(\mathbf{x}) > g_l$ and $\mathbf{x}$ is connected to a pixel already marked
12:     Display the updated edge map within $\hat{w}_i$
13:     $i := i + 1$
14:   end if
15: end while
```

Algorithm 1 summarizes our incremental edge detection scheme. When the window is in motion, we perform local Canny edge detection in the current window w_i (at each time instance t_i). In the incremental approach, however, the boundary pixels of w_i, denoted $\Omega(w_i)$, need special care. Let $\hat{w}_i$ denote w_i 'eroded' by one pixel wide, that is, $\hat{w}_i = w_i - \Omega(w_i)$. Since the nonmaxima suppression requires information from the neighboring pixels, the gradients are estimated in the entire w_i, while nonmaxima suppression is done for the pixels in $\hat{w}_i$. To perform hysteresis thresholding, we first mark all the maxima pixels $\mathbf{x}$ in $\hat{w}_i$ whose gradient magnitude $g(\mathbf{x})$ is higher than g_h. The second-level thresholding with respect to g_l is based on the connectivity to these selected maxima pixels, and the edge segments surviving this test may extend to the next window. Thus, we take an incremental approach, and include for the

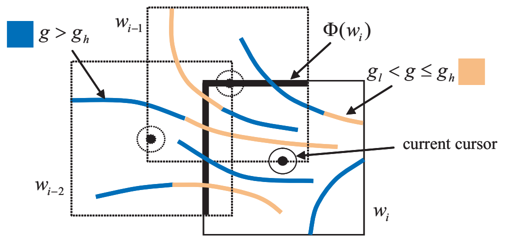


Figure 4. Incremental edge map expansion

connectivity test not only the maxima pixels in $\hat{w}_i$ whose magnitudes are above g_h, but also all the 'incoming' edge pixels from the previous windows whose magnitudes are above g_l. These incoming edge pixels are inspected on the portion of $\Omega(w_i)$, denoted $\Phi(w_i)$, intersecting the previous window sequence (see Fig. 4). After hysteresis thresholding, the updated outline map E (within $\hat{w}_i$) is displayed.

We provide two basic operation modes, named *paint* and *tune*. The *paint* mode refers to the user-guided incremental edge painting mode described above. In *tune* mode, the window is fixed, and the user adjusts any of the parameters and the local outline map is updated accordingly. The parameter values may be tuned up and down in a continuous fashion, until the optimal set of parameters is found for a given window area. There is an additional mode called *pick*, where the user is allowed to pick any specific pixel from the outline map so that its associated parameters are used for the subsequent painting operations. This is useful when the user wishes to achieve similar edge detection results in different parts of the image, after there have been some changes in parameter values.

The edge painting often operates like an 'intelligent eraser'. For example, with relatively large parameter values, the weak and extraneous edges are removed while strong, large-scale edges are preserved in the swept area (see Fig. 3b). As shown in Fig. 5, edge painting improves the quality of outline map by minimizing missing outlines or extraneous edges. Note that the resulting outline map itself can be viewed as 'line-drawing art' in its own right. Although the edge painting paradigm, compared with global Canny edge detector, helps obtain a cleaner outline map, its performance still may be limited in some images where there are many unclear outlines and/or closely-spaced outlines that have similar strengths but need to be separated. Additional stylization of the outlines is possible, such as varying the width of the outlines, adding stroke texture [19], or introducing opacity to each outline [3]. Our incremental edge detection algorithm works at interactive speed regardless of the image size, since the time complexity at each time t_i is strictly bounded by the window, which is typically much smaller than the image.

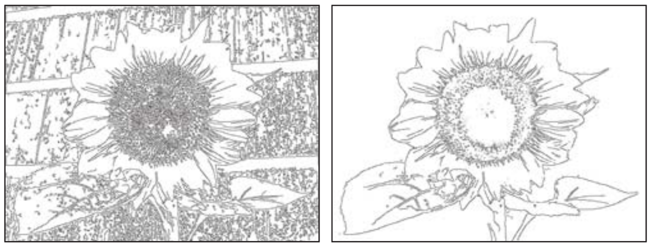

(a) Global edge detection (b) Edge painting

Figure 5. Advantage of Edge Painting

3. Direction Map Construction

Our stroke-based rendering system uses a curved stroke model associated with several attributes such as position, color, opacity, width, length, path, and so on (see Fig. 6).

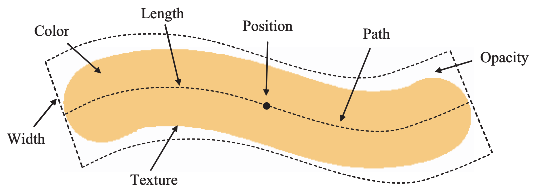


Figure 6. Stroke model

To allow for adaptive (pixel-dependent) stroke attribute computation, we derive an *importance map* from the outline map. Here, we measure the *distance* from the nearest edge. That is, the closer a pixel is to the nearby edges, the more important it is (and needs to be preserved by using small-scale strokes). Thus, the importance map $M(\mathbf{x})$ is essentially a *distance field*, where each pixel records the minimum distance to the neighboring edges:

$$M(\mathbf{x}) = [d(\mathbf{x}, E) / \max_{\mathbf{y}} d(\mathbf{y}, E)]^n \tag{1}$$

where $d(\mathbf{x}, E)$ denotes the minimum distance from $\mathbf{x}$ to the edges in E. M is normalized to have values in $[0, 1]$, and the factor n is optionally used to curve the importance. Note that the *lower* the $M(\mathbf{x})$, the *more important* $\mathbf{x}$ is. Fig. 7 shows an example importance map.

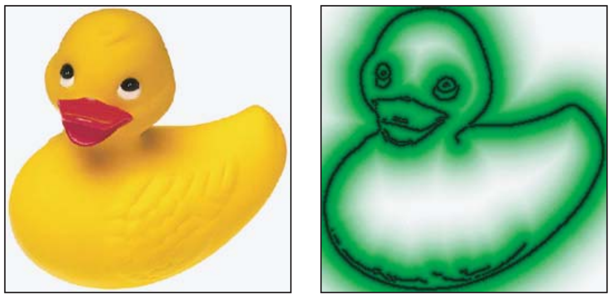

(a) Input image (b) Importance map

Figure 7. Importance map

3.1 Adaptive Bilateral Filter

Among various illustration styles, *line drawing* is the simplest one to produce, since it is directly obtained from the outline map. For all other styles, however, guiding the strokes in regions interior to the outlines is a non-trivial problem. Typically, artists draw interior strokes following the dominant outlines nearby. Previous approaches for constructing stroke direction field follow this principle, while they differ in their underlying techniques to convey the directions from outlines to the interior, which include scattered data interpolation [21, 14], variational calculus [20], and voronoi diagrams [13]. With these approaches, however, it is difficult to control the 'smoothness' of the direction path at each pixel adaptively. For example, to abstract out unimportant regions more severely, the direction paths in those regions should be smoothed out to a greater degree.

Thus, we propose a novel, effective technique to achieve this goal, based on *adaptive bilateral filter*. The bilateral filter is a powerful feature-preserving filter introduced by Tomasi and Manduchi [28], which has been mainly used for data denoising [6, 9, 10]. In our approach, we newly introduce an 'adaptive' version of bilateral filter, and apply it to the initial *tangent map* $\mathbf{t}(\mathbf{x}) = (t_x, t_y)$ (perpendicular to gradients) to obtain adaptively computed stroke direction field $\mathbf{d}(\mathbf{x}) = (d_x, d_y)$. The adaptive bilateral filtering is defined as follows:

$$\mathbf{d}(\mathbf{x}) = \sum_{\mathbf{y}\in\xi(\mathbf{x})} \frac{\mathbf{t}(\mathbf{y})W_c(\mathbf{x},\mathbf{y})W_d(\mathbf{x},\mathbf{y})}{\sum_{\mathbf{z}\in\xi(\mathbf{x})} W_c(\mathbf{x},\mathbf{z})W_d(\mathbf{x},\mathbf{z})} \tag{2}$$

where $\xi(\mathbf{x})$ denotes the neighborhood of $\mathbf{x}$. The *closeness weight function* W_c is defined as:

$$W_c(\mathbf{x},\mathbf{y}) = e^{-||\mathbf{x}-\mathbf{y}||^2/2\sigma_c(\mathbf{x})^2} \tag{3}$$

Note the parameter $\sigma_c(\mathbf{x})$ is not a constant, but a function of $\mathbf{x}$, implying the size of filter kernel is *adaptive* at each pixel. We assign a value proportional to the importance map, that is, $\sigma_c(\mathbf{x}) = a_c + b_c \cdot M(\mathbf{x})$ for some constants $a_c, b_c \geq 0$. Similarly, the size of $\xi(\mathbf{x})$ is proportional to $\sigma_c(\mathbf{x})$. Thus, less important pixels will have bigger smoothing kernel, searching further away to look for dominant tangent vectors to follow.

Also, the *feature-preserving weight function* W_d is defined as:

$$W_d(\mathbf{x},\mathbf{y}) = e^{[g(\mathbf{y})-g(\mathbf{x})]/\sigma_d^2} \tag{4}$$

where $g(\mathbf{x})$ denotes the gradient magnitude at $\mathbf{x}$. Note that our feature-preserving weight function is *monotonically increasing*, unlike the original version in [28]. The purpose of our filtering is to "smooth out the vector field while preserving dominant vectors". Thus, bigger weights must be given to neighboring pixels $\mathbf{y}$ whose gradient magnitudes are higher than that of $\mathbf{x}$ (which explains why W_d has to be monotonically increasing).

Fig. 8 shows example adaptive direction maps obtained by our filter with varying kernel size (from Fig. 7(a)). With large b_c, note that the direction vectors are smoothed out to a greater degree on pixels with lower importance (i.e., higher M), while the outlines are always preserved regardless of the kernel size. Also, a purely non-adaptive direction map can be obtained by setting $b_c = 0$. Thus, our adaptive bilateral filtering is by far the most general and versatile technique for stroke direction map construction. While the bilateral filtering is basically a non-iterative process, it may also be applied multiple times to further enhance the filter response, as pointed out in [7].

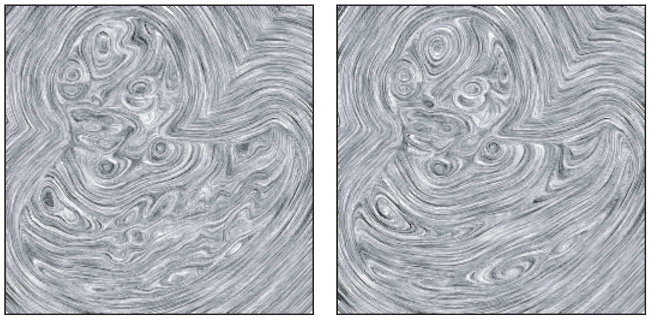

(a) Low b_c (b) High b_c

Figure 8. Adaptive direction maps

4. Color Selection

Given the three guiding maps (outline map, importance map, and direction map), we run the stroke rendering algorithm to create an illustration. Our rendering algorithm is associated with various style parameters, as will be detailed in the subsequent sections. Among these parameters, *the number of colors*, denoted N_c, is particularly important in providing a wide range of illustration styles. For example, traditional paintings normally use a large number of colors, while cartoon illustrations or tile mosaics typically require just a small number of colors, and pen-and-ink illustrations use a single, black color (on a white canvas).

Before rendering, N_c representative colors are automatically selected, forming the color palette for the strokes. The system goes through the following 3 steps to select colors: Initialization, K-means refinement, Saturation-Value (SV) adjustment.

Initialization: N_c colors are randomly picked from the input photograph I, forming the initial palette of colors $\mathbf{c}_i$ $(1 \leq i \leq N_c)$, which is then iteratively refined by replacing an old color with a new one sampled from I. Among the old colors, the closest one to the new sample is picked for replacement. The replacement occurs only if it increases the following function:

$$t \cdot \sum_{i,j} ||\mathbf{c}_i - \mathbf{c}_j||^2 + (1-t) \cdot \min_{i,j} ||\mathbf{c}_i - \mathbf{c}_j||^2 \tag{5}$$

where $t \in [0,1]$, $1 \leq i,j \leq N_c$ and $i \neq j$. This formulation aims at collecting sufficiently distinct colors, by maximizing the total and minimum color differences in the palette.

K-means refinement: The palette is further refined by an iterative K-means algorithm. At each iteration, we cluster the pixels in I based on the closest sample color in the palette, which is then replaced with the mean of the corresponding cluster. After a sufficient number of iterations, the following function is minimized:

$$\sum_{i}^{N_c} \sum_{j}^{N_i} ||\mathbf{c}_j^i - \mathbf{c}_i||^2 \tag{6}$$

where $\mathbf{c}_j^i$ denotes the color of the j^{th} member of cluster i, $\mathbf{c}_i$ the mean of cluster i, and N_i the cluster size. That is, N_c representative clusters are formed on I, and their mean colors are used to comprise the palette.

SV adjustment: The resulting RGB color values in the palette are converted to HSV colors, to adjust their saturation (S) and the brightness (V) by linear interpolation with the corresponding maximum possible values, that is, $S \leftarrow (1-t)S + t \cdot S_{max}$ and $V \leftarrow (1-t)V + t \cdot V_{max}$, where $t \in [0, 1]$. The purpose of this adjustment is to enhance the liveliness of the colors, as is often done by artists [12]. Fig. 9 shows example illustrations generated by our system, with different number of colors selected (see Fig. 10 for the input picture). Note that in Fig. 9(b), the outline map is overlaid on the illustration to enhance the cartoonish effect.

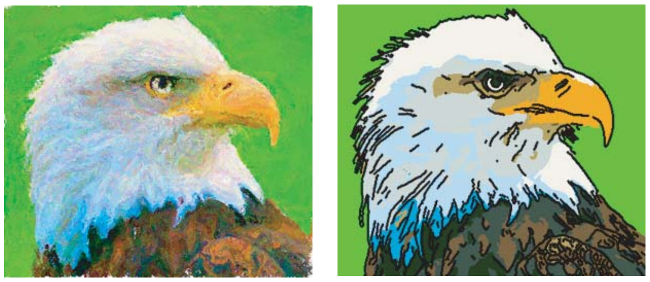

(a) Oil painting (N_c = 70) (b) Cartoon illustration (N_c = 10)

Figure 9. Color selection

5. Stroke Rendering

Our rendering algorithm is summarized in Algorithm 2. In stroke-based rendering, the output is represented by an ordered list of strokes, denoted S. Our algorithm is based on stochastic stroke optimization, to find an optimal S with respect to the following objective function:

$$f(S) = \sum_{\mathbf{x}} ||\mathcal{F}(I(\mathbf{x})) - I_S(\mathbf{x}) * G_\sigma||^2 \tag{7}$$

where I denotes the input image, and I_S denotes the rendering of S. We use the Gaussian-blurred rendering for comparison to cover a wide range of rendering styles, especially stipple drawing, pen-and-ink illustrations, or mosaics, which normally encourage spacings between strokes. For other rendering styles, σ may be set to zero. Function $\mathcal{F}$ denotes the possible initial filtering of the input image. In general, we use an adaptive Gaussian smoothing filter for $\mathcal{F}$. The size of the smoothing kernel at each pixel is proportional to the importance value M, and thus adaptively affects the noisiness (and smoothness) of the resulting illustrations. For black-and-white illustrations (such as pen-and-ink or stippling), the initial filtering also includes color-to-gray-scale image conversion.

Our optimization algorithm employs the framework of *simulated annealing* based on iterative random stroke operations, to minimize f in Eq. 7. The optimization process is associated with an *overlap constraint*, that is used to maintain the required interval between neighboring strokes. This allows for more sophisticated style control. Its implementation will be discussed later in this section.

We provide five basic stroke operations, *add*, *move*, *resize*, *recolor*, and *delete*:

- **Add:** add a new stroke at a random location.
- **Move:** move a stroke to a neighboring position.
- **Resize:** slightly perturb the size of a stroke.
- **Recolor:** slightly perturb the color of a stroke.
- **Delete:** delete a stroke.

At each iteration, one of these operations is randomly selected and performed on S, producing a hypothetical update S'. The five operations may be associated with predefined probabilities for selection, such as {0.5, 0.3, 0.1, 0.0, 0.1}, depending on the intended rendering style. Basically, only an operation that reduces f is accepted and allowed to update the rendering, that is, replace S with S'. In simulated annealing, however, even an operation which increases f may be accepted with the probability of $e^{-[f(S')-f(S)]/T}$, where T is called the *system temperature*. The higher the temperature, the greater the probability to accept a false move. After each iteration, T decreases by a *cooling factor* γ ($0 < \gamma < 1$). The role of T is to reduce the chances of getting trapped in a local minimum in early stages. The whole iteration process stops after a sufficient number of consecutive operations have failed, depending on the style and the image complexity. In essence, our optimization framework resembles the random-descent algorithms suggested in [29, 16]. Note, however, our algorithm is "less greedy" (allowing false moves) and thus improves the possibility of finding a better solution in the long run [23]. Also, unlike previous approaches, it is designed to support a wide range of stroke-based illustration styles.

Algorithm 2 Stroke Optimization

```
1:  $S$ := null, $T$ := $T_0$
2:  Randomly put $N_b$ strokes in $S$
3:  repeat
4:     Choose a stroke operation $\mathcal{O}$
5:     $s$ := a stroke associated with $\mathcal{O}$
6:     $\mathbf{x}_s$ := a pixel location associated with $\mathcal{O}$
7:     Produce $S'$ by applying $\mathcal{O}$ on $S$ at $\mathbf{x}_s$
8:     if $Eval(s, \mathbf{x}_s, S, S', T)$ = $true$ then
9:        $S$ := $S'$
10:    end if
11:    $T$ := $T \cdot \gamma$
12: until convergence
```

We first put random initial strokes on the canvas sufficiently. The number of initial strokes, denoted N_b, may depend on the rendering style. At each iteration of the optimization loop, we perform a (hypothetical) operation on S to produce S'. For 'add', we assign the attributes of a new stroke s centered at a randomly-picked pixel $\mathbf{x}_s$. The stroke color $\mathbf{c}_s$ is picked from the palette, as the closest one to the color of pixel $\mathbf{x}_s$ on I. Random perturbation of the stroke color, denoted $\mathbf{p}_c$, is allowed in certain rendering styles (such as traditional paintings) to increase randomness. That is, $\mathbf{c}_s \leftarrow \mathbf{c}_s + \mathbf{p}_c$. The width and length of the stroke are randomly assigned within the ranges between some minimum and maximum values (denoted $[min_w, max_w]$ and $[min_l, max_l]$, respectively), that are proportional to the importance value, $M(\mathbf{x}_s)$. Depending on styles, a non-adaptive stroke size may be used instead. The curved path of the stroke follows the direction map $\mathbf{d}(\mathbf{x})$, elongating from $\mathbf{x}_s$ in both directions up to the half of its length. With all of these computed attributes, the new stroke s is inserted in S to produce S' (and $I_{S'}$). In most cases, s is simply 'overlaid' on I_S, but for watercolor painting, the alpha blending factor (denoted α, where $0 < \alpha \leq 1$) is applied for smooth blending of I_S and s. For 'move', we randomly pick a stroke $s \in S$, and assign a new center location $\mathbf{x}_s$ from its neighborhood, while preserving all other attributes. The 'resize' and 'recolor' operation slightly perturbs the size (width/length) and the color, respectively, of a randomly selected stroke s, centered at $\mathbf{x}_s$. The amount of perturbation allowed depends on the style. For 'delete', a selected stroke is removed from S.

Algorithm 3 Function $Eval(s, \mathbf{x}_s, S, S', T)$

```
1: if $overlap(s, \mathbf{x}_s, S')$ returns $false$ then
2:    Return $false$
3: end if
4: if $f(S') < f(S)$ then
5:    Return $true$
6: else if a random number $\in (0,1) < e^{-[f(S')-f(S)]/T}$ then
7:    Return $true$
8: end if
9: Return $false$
```

The legitimacy of the given operation is evaluated by $Eval$ routine (see Algorithm 3). Using Eq. 7, we check if $f(S')$ is lower than $f(S)$. For computational efficiency, this test is done only in the bounding box of s, denoted $B(s)$.

The *overlap constraint* is also tested in $Eval$. It is performed on S', between the test stroke s and its neighboring strokes r:

$$\bigcup_{r \in B(s)} [\mathcal{D}(s, \rho(\mathbf{x}_s)) \cap r] = \phi \quad (8)$$

where $\mathcal{D}(s, \rho)$ denotes the stroke s dilated by the width of ρ. If the dilated stroke intersects any of the neighboring strokes, this operation is canceled. Thus, the bigger value of ρ will enforce a larger interval. The dilation width may be adaptively defined proportional to the gray-scale pixel brightness, denoted I_B, that is, $\rho(\mathbf{x}) = a_o + b_o \cdot I_B(\mathbf{x})$ for some constants a_o and b_o. If $b_o = 0$, it is non-adaptive. Note ρ can be negative, in which case the operation becomes an 'erosion', allowing some amount of overlap. If $\rho \leq -max_w$, the system allows unlimited overlapping. The addition of this constraint is crucial in handling a variety of rendering styles, especially, stipple drawing, pen-and-ink drawing, or mosaics, where the interval between strokes plays an important role.

6. Results

Our style parameters include N_c (number of colors), N_b (number of initial strokes), $|\mathbf{p}_c|$ (amount of color perturbation), min_w (minimum width), max_w (maximum width), min_l (minimum length), max_l (maximum length), α (blending factor), σ (blur factor in Eq. 7), ρ (dilation width in Eq. 8), γ (cooling factor). These parameters may be set automatically or manually, and are applied to all rendering styles except line drawing. Note that many of them can be set either adaptively or non-adaptively. Table. 1 contains some rough parameter value ranges suitable for each of the possible rendering styles, showing that our scheme is indeed general enough to cover a wide range of illustration styles. Note that this table merely provides a general suggestion based on our experiments. In fact, there are no clear cut lines between styles, and any combination of parameter values may be used to achieve an intended rendering style. For example, even within the oil painting framework, while the standard parameter values are suited to produce *impressionist* style, increasing $|\mathbf{p}_c|$ gets it closer to *expressionist* style, and setting $min_l = max_l = min_w = max_w$ results in *pointillist* style.

Fig. 11 contains various illustration results generated by our system, from the sample photographs in Fig. 10. Due to the page restrictions, we show one example for each style. Fig. 11(a) shows an oil painting style obtained by using rough, textured strokes. Fig. 11(c) shows some procedural hatching patterns created by allowing strokes to form

Rendering style	N_c	N_b	$\lvert\mathbf{p}_c\rvert$	min_w	max_w	min_l	max_l	α	σ	ρ	γ
Oil painting	high	high	high	low	high	low	high	high	0	$-max_w$	high
Watercolor painting	medium	high	low	low	high	low	high	low	0	$-max_w$	high
Cartoon illustration	low	high	0	low	high	low	high	1	0	$-max_w$	medium
Pen-and-ink illustration	1	low	0	1	1	low	high	1	high	> 0	low
Stipple drawing	1	low	0	1	1	1	1	1	high	> 0	low
Tile mosaics	low	low	low	c	c	c	c	1	low	0	low

Table 1. Recommended parameter ranges for various rendering styles

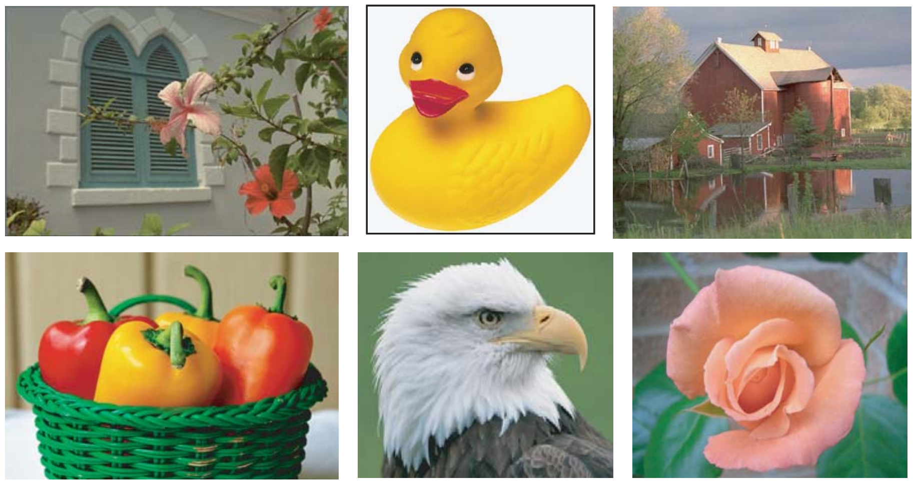


Figure 10. Sample photographs

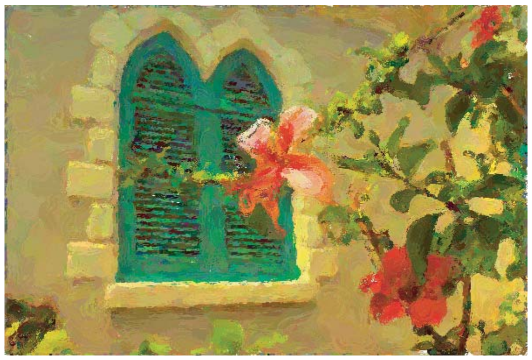
angles with the direction vectors. Fig. 11(f) was obtained from the original picture with the outlines overlaid on it. As shown in Fig. 11(b), (c), and (f), we can enforce adaptive or non-adaptive overlap constraints among strokes. Also, note that line drawings can be naturally mixed with other rendering styles (see Fig. 11(b), (e), and (f)). All of these examples (except stippling) use adaptive stroke sizes following the importance map. Fig. 12 shows how the level-of-abstraction is adaptively controlled by the outline map. The outline map in Fig. 12(c) is obtained by applying different edge detection parameters for each face, affecting the final illustration accordingly (see Fig. 12(e)). In our current implementation (on a 3GHz PC), the rendering time varies widely, ranging from the order of tens of seconds to tens of minutes, depending on the style, image size, and the image complexity. In general, stippling (small stroke size) and mosaics (strong constraints on stroke shape and overlapping) converge slowly. Possible future extensions to expedite the process include maintaining a candidate set of pixels (or strokes) to choose the next move from, and/or incorporating some style-specific heuristics.

7. Conclusions

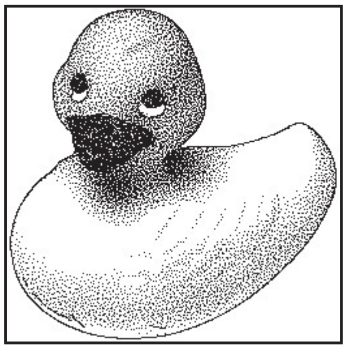
We have presented a comprehensive, and unified framework for automatically generating various styles of stroke-based illustrations from photographs, which also supports adaptive control of the abstraction level while preserving important features. To support a broad class of illustration styles, the system provides a number of parameters that can be easily tailored to the target style, either adaptively or non-adaptively. From technical point of view, we proposed several new techniques to effectively achieve our goal, including edge painting, adaptive bilateral filter, color selection algorithm, and stroke optimization framework based on simulated annealing. The experimental results show that our system successfully produces high-quality illustrations from arbitrary photographs in a wide range of styles.

Acknowledgement

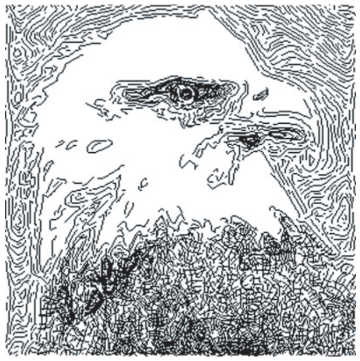
This research was partially supported by ARO Grant #W911NF-04-1-0298 and DARPA/NGA Grant #HM-1582-05-2-2003. It was also supported by UM Research Board and the MIC (Ministry of Information and Communication), Korea, under ITRC (Information Technology Re-



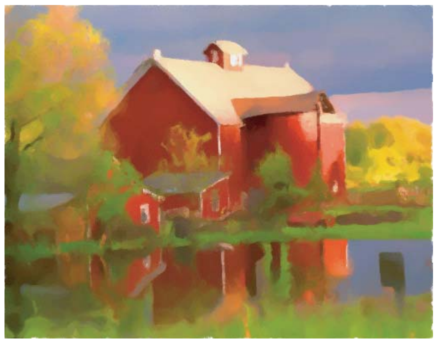
(a) Oil painting



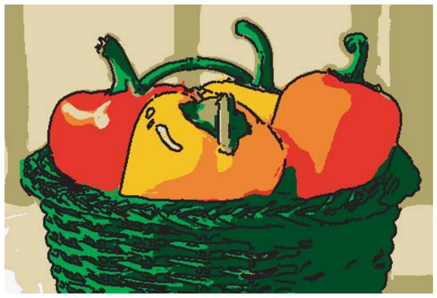
(b) Stipple drawing



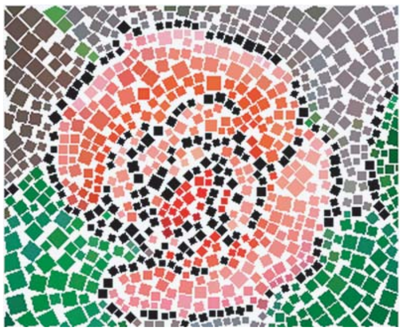
(c) Pen-and-ink illustration



(d) Watercolor painting



(e) Cartoon rendering



(f) Tile mosaic

Figure 11. Various illustrations generated by our system

search Center) support program supervised by IITA (Institute of Information Technology Assessment).

References

[1] J. Canny. A computational approach to edge detection. *IEEE Trans. on Pat. Anal. Mach. Intel.*, 8(6):679–698, 1986.

[2] C. Curtis, S. Anderson, J. Seims, K. Fleischer, and D. Salesin. Computer-generated watercolor. In *Proc. ACM SIGGRAPH*, pages 421–430, 1997.

[3] D. DeCarlo, A. Finkelstein, and S. Rusinkiewicz. Interactive rendering of suggestive contours. In *Proc. Non-Photorealistic Animation and Rendering*, pages 15–24, 2004.

[4] D. DeCarlo and A. Santella. Stylization and abstraction of photographs. In *Proc. ACM SIGGRAPH*, pages 769–776, 2002.

[5] O. Deussen, S. Hiller, C. van Overveld, and T. Strothotte. Floating points: A method for computing stipple drawings. *Computer Graphics Forum*, 19(3):41–50, 2000.

[6] F. Durand and J. Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. In *Proc. ACM SIGGRAPH*, pages 257–266, 2002.

[7] M. Elad. On the bilateral filter and ways to improve it. *IEEE Transactions on Image Processing*, 11:1141–1151, 2002.

[8] J. Elder and S. Zucker. Local scale control for edge detection and blur estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(7):699–716, 1998.

[9] S. Fleishman, I. Drori, and D. Cohen-Or. Bilateral mesh denoising. *ACM Transactions on Graphics*, 22:950–953, 2003.

[10] R. Garnett, T. Huegerich, C. K. Chui, and W. He. A universal noise removal algorithm with an impulse detector. *IEEE Transactions on Image Processing*, 14:1747–1754, 2005.

[11] B. Gooch, G. Coombe, and P. Shirley. Artistic vision: Painterly rendering using computer vision techniques. In *Proc. NPAR*, pages 83–90, 2002.

[12] P. Haeberli. Paint by numbers: Abstract image representations. In *Proc. ACM SIGGRAPH*, pages 207–214, 1990.

[13] A. Hausner. Simulating decorative mosaic. In *Proc. ACM SIGGRAPH*, pages 573–578, 2001.

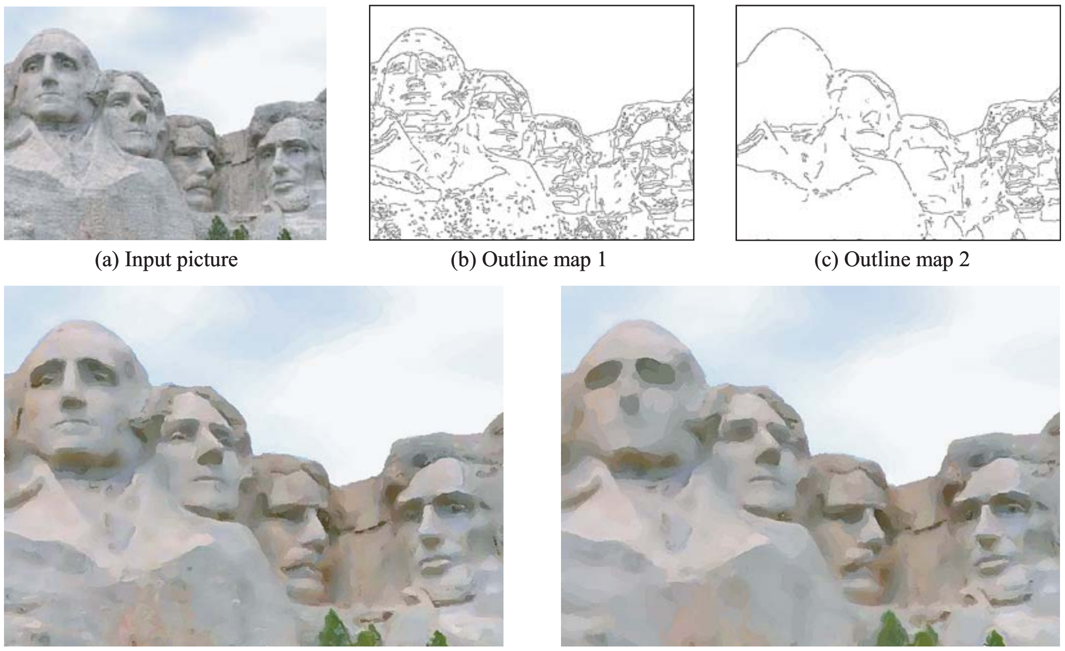
[14] J. Hays and I. Essa. Image and video-based painterly animation. In *Proc. NPAR*, pages 113–120, 2004.

[15] A. Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *Proc. ACM SIGGRAPH*, pages 453–460, 1998.

[16] A. Hertzmann. Paint by relaxation. In *Proc. Computer Graphics International*, pages 47–54, 2001.

[17] A. Hertzmann. A survey of stroke-based rendering. *IEEE Computer Graphics and Applications*, 23(4):70–81, 2003.

[18] A. Hertzmann, C. Jacobs, N. Oliver, B. Curless, and D. Salesin. Image analogies. In *Proc. ACM SIGGRAPH*, pages 327–340, 2001.



(a) Input picture (b) Outline map 1 (c) Outline map 2

(d) Painting using outline map 1 (e) Painting using outline map 2

Figure 12. Adaptive stroke-based illustration

[19] S. Hsu and I. Lee. Drawing and animation using skeletal strokes. In *Proc. ACM SIGGRAPH*, pages 109–118, 1994.

[20] H. Kang, W. He, C. K. Chui, and U. Chakraborty. Interactive sketch generation. *The Visual Computer*, 21(8):812–830, 2005.

[21] P. Litwinowicz. Processing images and video for an impressionist effect. In *Proc. ACM SIGGRAPH*, pages 407–414, 1997.

[22] V. Ostromoukhov. Digital facial engraving. In *Proc. ACM SIGGRAPH*, pages 417–424, 1999.

[23] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical recipes in C++*. Cambridge University Press.

[24] M. Salisbury, M. Wong, J. Hughes, and D. Salesin. Orientable textures for image-based pen-and-ink illustration. In *Proc. ACM SIGGRAPH*, pages 401–406, 1997.

[25] A. Santella and D. DeCarlo. Abstracted painterly renderings using eye-tracking data. In *Proc. NPAR*, pages 75–82, 2002.

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

[27] M. Sousa and J. Buchanan. Observational models for graphite pencil materials. *Computer Graphics Forum*, 19(1):27–49, 2000.

[28] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proc. ICCV*, pages 839–846, 1998.

[29] G. Turk and D. Banks. Image-guided streamline placement. In *Proc. ACM SIGGRAPH*, pages 453–460, 1996.