Towards Photo Watercolorization with Artistic Verisimilitude

Miaoyi Wang, Bin Wang, Yun Fei, Kanglai Qian, Wenping Wang, Jiating Chen, and Jun-Hai Yong

Abstract—We present a novel artistic-verisimilitude driven system for watercolor rendering of images and photos. Our system achieves realistic simulation of a set of important characteristics of watercolor paintings that have not been well implemented before. Specifically, we designed several image filters to achieve: 1) watercolor-specified color transferring; 2) saliency-based level-of-detail drawing; 3) hand tremor effect due to human neural noise; and 4) an artistically controlled *wet-in-wet* effect in the border regions of different wet pigments. A user study indicates that our method can produce watercolor results of artistic verisimilitude better than previous filter-based or physical-based methods. Furthermore, our algorithm is efficient and can easily be parallelized, making it suitable for interactive image watercolorization.

Index Terms—non-photorealistic rendering, watercolor rendering, image processing, artistic-verisimilitude.

1 Introduction

Due to the attractive features and artistic appeals of watercolor paintings, various rendering methods have been proposed for processing images, videos and 3D scenes into watercolor works. There are popular software tools for watercolorization, such as Adobe Photoshop that provides powerful control for professional artists, as well as CinemaFX on mobile devices for ordinary casual users. Most of these tools are based on image filters [1], [2], [3], [4], [5], since the image filter approach is usually efficient, easy to implement, GPU friendly, and allows flexible user interaction.

However, there is still significant room for the improvement on the results by these methods. For example, some distinct physical characteristics in watercolor paintings, such as the wet-in-wet effect, are usually ignored in filter-based methods due to their complexity. Physical simulation using shallow water hydrodynamics [6], [7], [8] has been employed for paper-stroke interaction, providing realistic appearance for pigments and strokes. Despite the promising potential of physical-based methods, their dependence on physical laws makes the process inflexible for user interaction. Moreover, these physical-based methods require much more computations and thus are more time-consuming than the filter-based methods.

Artists with different styles and experiences may draw watercolor paintings using different techniques, color choices and layouts. Many artistic skills are required to determine image regions to apply appropriate physical effects. Besides, because of the noise in the human nervous system [9], an artist hardly paints exactly the same result twice for the same scene. In our discussion with many professional artists these considerations haven been found to be crucial for a watercolor painting to appear *watercolor-like*.

Our goal in this work is to develop an artistic-verisimilitude driven system that is capable of producing highly *watercolor-like* results by more user control. To this end, we have adopted a filter-based approach to meet the demands of professional painters. A user study has been conducted to validate the acceptance of our method. In most cases our results achieve scores very close to the artworks of watercolorists, and are preferred by the professional artists to the results by computed-assisted watercoloring methods.

In sum, our method has made the following contributions:

- A new framework for automatically selecting appropriate painting techniques, color, and level-of-detail (LOD) for different regions based on color abstraction and saliency information.
- A new image filter for simulating the wet-in-wet effect that can be controlled either locally or globally.
- A new image filter to simulate the hand tremor effect that is natural in the real watercoloring.
- A GPU-based image watercolorization system with interactive performance.

Figure 1 shows some watercolor results created by our system from images and photos. An overview of the pipeline is shown in Figure 2.

In the following we will first introduce some related work in Section 2 and present the technical details

- *M. Wang, B. Wang, Y. Fei, K. Qian, J. Chen and J.-H. Yong are with School of Software, Tsinghua University, Beijing, CO 100084, P.R. China. E-mail: sallybear119@gmail.com, wangbins@tsinghua.edu.cn, fyun@acm.org, qiankanglai@gmail.com, chenjt04@gmail.com, yongjh@tsinghua.edu.cn.*
- *B. Wang is the corresponding author.*
- *W. Wang is with Department of Computer Science, the University of Hong Kong, Pokfulam Road, Hong Kong. E-mail: wenping@cs.hku.hk.*

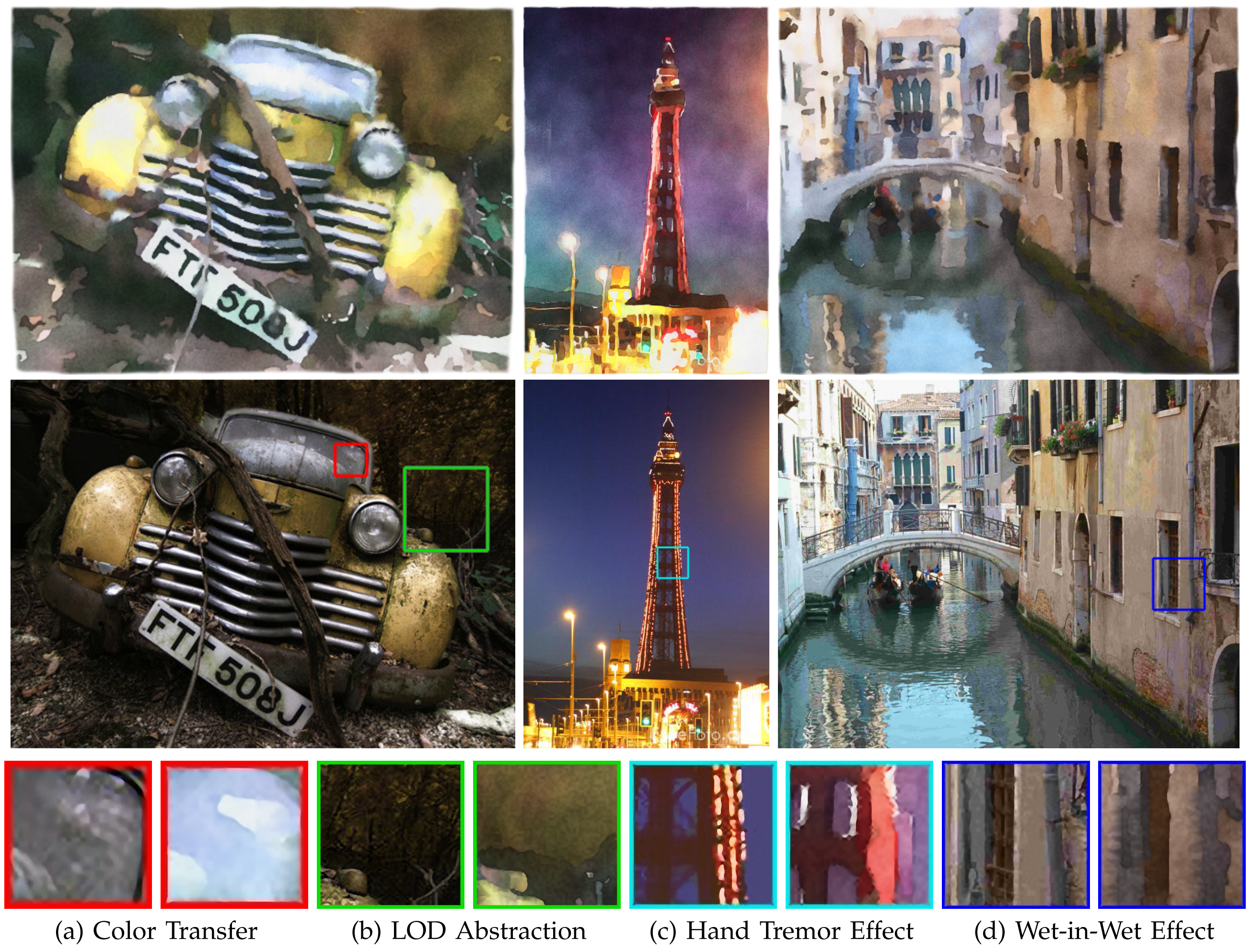


Figure 1. Samples of watercolor paintings created by our system from images or photos. Top row: our results. Middle row: the input. Bottom row: the zoomed regions corresponding to the novel effects proposed in this paper. In (a) the color has been tuned to the one that is more common in watercolor paintings; in (b) the details of the background are blurred while the interesting foreground can be emphasized; in (c) the boundaries of strokes are distorted with overlap and gap due to hand tremor phenomenon; and in (d) the wet-in-wet effect can be well simulated.

in Section 3. We will then provide our watercolor results and discuss the findings in a comprehensive user study in Section 4.

2 Related Work

Methods for simulating watercolor effects fall largely into three categories: 1) physical-based methods [6], [7], [10]; 2) stroke-based methods [11], [12], [13]; and 3) filter-based methods [1], [2], [3], [4]. Our method belongs to the third category, which is suited for automatically converting images and photos into watercolor paintings.

The earliest work on physics-based simulation of watercolor dates back to 1991 when Small [10] simulated the flow of watercolor pigments on paper with a Connection Machine. This work inspired the three-layer model by Curtis et al. [6] that simulates pigment flow more accurately. Then this work was further improved by Laerhoven et al. [7] to achieve an interactive frame rate using a distributed paper model. An interactive oriental ink paint system *Moxi* was developed by Chu and Tai [8], which simulates percolation of ink with the Lattice Boltzmann Equation. These methods can simulate water-paper interaction with accurately modeled virtual brushes, but they are not suited for converting images into watercolor paintings. When using the stroke method to simulate watercolor painting, strokes will mix together and light color strokes cannot override dark color ones, which makes the result look like oil paintings.

Image processing methods focus on using various image filters to recreate watercolor effects. Johan et al. [11] generated strokes along a vector field defined over an input image. Its results are easily distinguished from real watercolor. Kang et al. [12] proposed a unified scheme to create paintings with strokes in different styles relying on edge detection and optimization. Recently, DiVerdi et al. [14] proposed a vectorized watercolor painting system aiming at better user interactivity.

Lum and Ma [1] presented a multi-layer texturing method for rendering surfaces in watercolor style. Their method operates in 3D object space for better temporal coherence. However, natural watercolor appearance is not achieved in their results. Lei and Chang [2] presented a rendering pipeline using 2D filters to simulate watercolor effects. They used a Sobel filter to simulate the edge darkening effect, which is widely used in later studies. Luft et al. [3] presented several approaches to watercolor rendering. Their key idea is to decompose the image into several canvas layers and process these layers using different image filters or image space techniques, and then combine these layers to produce the final image. Bousseau et al. [4], [5] used random noise to simulate granulation and used fractal Perlin noise [15] to simulate the turbulence flow effect. Although parameters are provided for controlling these effects, changes made must be applied to the whole image globally, thus it not possible to apply different effects in different parts of the image.

Level-of-detail (LOD) abstraction is an important step in image stylization because watercolor artists often just want to emphasize details in the image regions of interest. Different methods have been proposed for abstraction. These methods require the users to interactively specify important regions [16], [17], [18], rely on semantic information [19], [20], or identify local features based on gradient, curvature or texture complexity [12], [21], [22]. To automatically identify global important regions without semantic information or special devices, we perform saliency detection [23] to achieve LOD abstraction for watercolor effect control.

3 New Approach

Our system first adjusts the color of the input images to better emulate the color features that are characteristic to real watercolor paintings (Section 3.1). In order to highlight the salient regions of the image (Section 3.2), we then perform an abstraction algorithm (Section 3.3) on the image. Next, we add the wet-in-wet (Section 3.4) and hand tremor (Section 3.5) effects, as well as some other effects (Section 3.6), including as edge darkening, granulation and turbulence flow. The flow chart of the system pipeline is shown in Figure 2.

3.1 Color Adjustment

Because only a limited number of colors are available in watercolor paintings, artists often have to paint with colors that are somehow different from what they actually see. Color choices are also affected by the personal aesthetic preferences (see Figure 3). In order to adopt proper color adjustment, we collected on the Internet and studied over 700 real watercolor paintings by 17 artists, and found that the colors used in a particular artist's works are usually similar and consistent, which probably help define the distinct style of each individual artist. In our system, we classify these paintings by their color characteristics (Figure 4). Then, given an input image, we choose the class that best matches the input image as the color scheme for the output image.

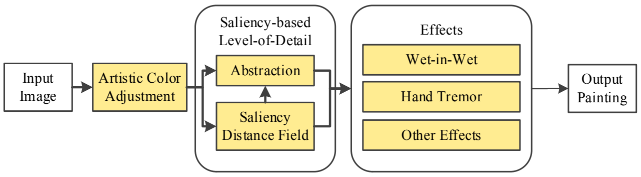


Figure 2. Pipeline of our image watercolorization.

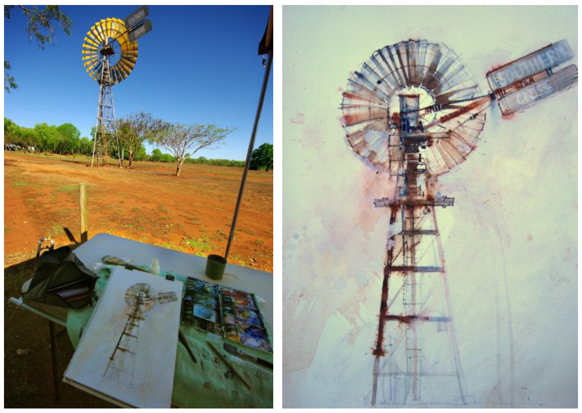


Figure 3. An artist's painting of a windmill. Note that the colors of the painting (right) are different from the observed colors (left).

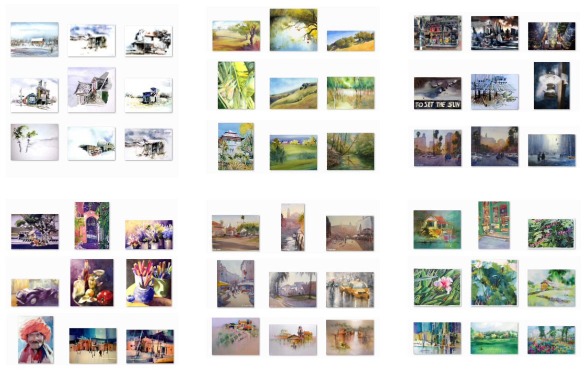


Figure 4. Examples of partial categories in our watercolor database, which are created by clustering images using a specified color feature.

Specifically, we define color features using the CIELab (Illuminant E) color space (L, a, b), which is designed to fit human vision and usually performs better than the RGB color space for various image processing applications [24]. We compute the mean and standard deviation of (L, a, b)

for each image as a six-dimensional feature vector $(L_{ave}, a_{ave}, b_{ave}, L_{std}, a_{std}, b_{std})$. This feature vector indicates the brightness, contrast, and main color tone of an image. We use the its components of this feature vector, weighted using the method in [25], to cluster the collection of the 700 watercolor paintings.

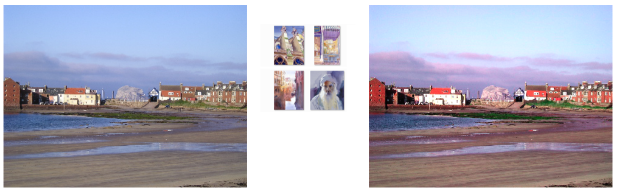


Figure 5. Color adjustment according to the color feature of a suitable class.

When computing color adjustment, the colors of an input image are adjusted to be its closest class of color features, using the color transfer algorithm presented by Reinhard et al. [26]. Sometimes there is no discernible difference between the adjusted result and the original, that is because the original is already very close the color schema of one class of watercolor paintings we considered. We have tested several color spaces and conclude that the CIELab (Illuminant E) achieves the best results, confirming the recommendation in [27]. Figure 5 shows an example of color adjustment with the referenced painting class. Although we allow the user to choose any class for color adjustment, in most cases the default choice generates acceptable results for user. The results provided in this paper and user study are all generated in this way.

3.2 Saliency Distance Field

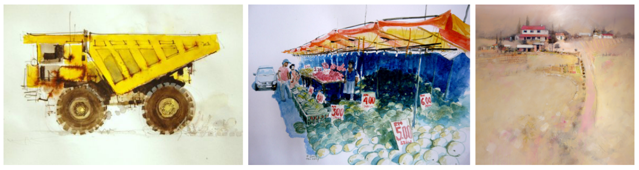


Figure 6. In real watercolor paintings, unimportant regions are often omitted, less saturated or more blurred.

Artists usually wish to emphasize some objects of interest in a painting while depicting the other regions with less detail, either less saturated or more blurred, as shown in Figure 6. To simulate this effect, we perform saliency detection [23] to identify the regions that are likely to be emphasized (Figure 7b). According to the detected saliency, a normalized distance field is generated with its value ranging from 0 to 1, where the smaller distance in the saliency region indicates higher importance, as shown in Figure 7. This saliency distance field is used later to guide the implementation of the abstraction effect, the wet-in-wet, and the hand tremor effects.

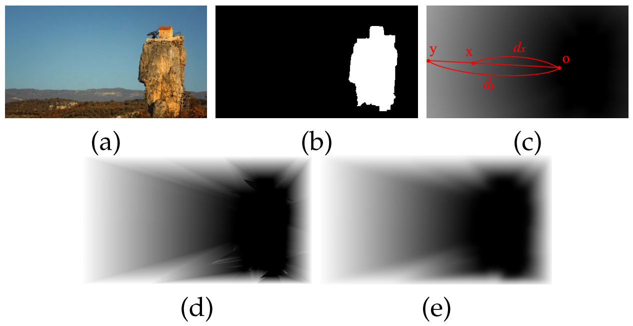


Figure 7. Generating a saliency distance field. (a) The input image. (b) Salient region detection [23]. (c) The absolute distance computed from (b) using the Jump Flooding Algorithm. (d) The normalized distance field computed from (c) by replacing d_x with d_x/d_y. (e) Smoothing (d) to eliminate discontinuous artifacts which could be observed around the bottom-right corner.

To compute the saliency distance field, we use the Jump Flooding Algorithm [28] (Figure 7c) and then normalize the values to be within $[0, 1]$: for each pixel x, its nearest pixel o on the boundaries of salient regions is detected as well as its collinear pixel y on the image border. The absolute distance d_x is replaced by d_x/d_y to generate a normalized distance field. For salient regions with concave boundaries or disconnected regions, the normalized distance field may be discontinuous because some pixels cannot find a proper border pixel y for normalization. In such a case we apply a recursive Gaussian filter [29] to smooth these discontinuous artifacts.

3.3 Abstraction

In watercoloring, regions that need not be emphasized will be simplified by abstraction (see Figure 8).

For abstraction, we segment the input image into color regions using mean shift segmentation in L*a*b with $(h_s, h_r, M) = (8, 5.5, 50)$. Then we apply a mean filter to create color boundaries with small color variations inside individual segments. For the segments inside the salient region, only the pixels in the same segment are smoothed by mean filtering with kernel size 5; for the segments in the non-salient regions, pixels belonging to different neighbor segments that have color difference smaller than $0.3d$ are also smoothed where d is the value in the distance texture for current pixel, with a larger kernel size $clamp(5 * 2 * (d + 0.3), 4, 9)$. As a result, variations in salient regions will be better preserved while the surrounding regions will tend to be be abstracted.

These parameters keep the same for all images in our experiments.

3.4 Wet-in-Wet

Wet-in-wet is an important and frequently used technique in watercolor paintings (Figure 9). Artists paint

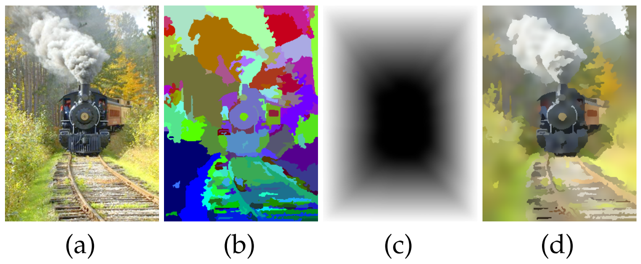


Figure 8. LOD abstraction. (a) The input image. (b) Segmentation. (c) Saliency distance field. (d) Abstracted result in which the middle areas keep more details than the surrounding areas.

with a wet brush very a freshly painted wet region so that the pigments are mixed to produce feather-like patterns along a region boundary. Among the previous methods for generating watercolor paintings, only those based on physical simulation approaches are capable of generating the wet-in-wet effect by simulating diffusion. However, where and how much this effect should be applied cannot be controlled due to the complexity of the shallow water hydrodynamics technique employed. We shall propose a technique based on image-filtering for simulating the wet-in-wet effect.

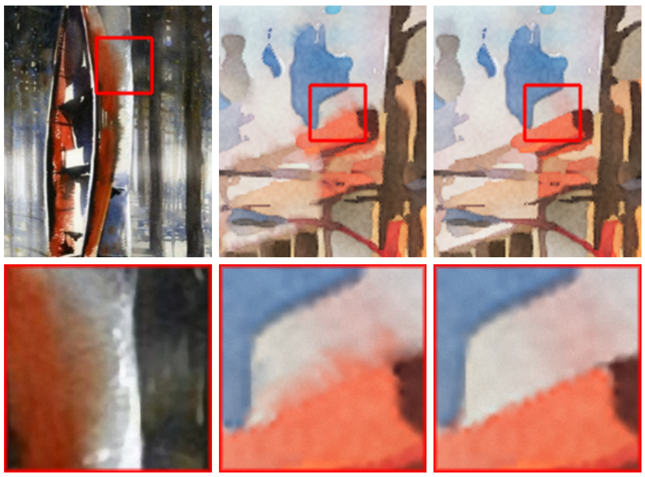


Figure 9. Wet-in-wet effect in the real watercolor painting (left) and our results with (middle) and without (right) this effect. The representative parts are enlarged in the bottom row.

The feather-like effect is caused by diffusion and vaporization involving water-carried pigment particles. In the wet-in-wet technique, the darker color is usually painted later for better control of color mixing. To generate this effect, we first randomly scatter some seeds around the boundaries, and then filter these areas with an ellipse-shaped kernel oriented along the normal vectors of a region boundary, as illustrated in Figure 10. If the filter is circle-shaped with a large radius, it will simulate painting effect with much water. Note that the seeds are distributed only on the brighter side of the boundary and assigned with the darker color of the opposite side.

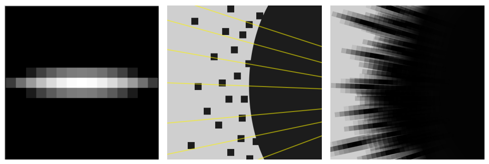


Figure 10. Creating feather-like effect. Left: The filter kernel. Middle: Scattering noise pixels around the boundary. Right: Filtering the noise area along the normal vectors of the boundary (the yellow lines) to generate a feather-like pattern.

The wet-in-wet effect should only be applied to certain type of boundaries. As a rule of thumb, the wet-in-wet effect should keep fine details in important regions. By studying real watercolor paintings, we propose the following rules according to color and saliency information to help determine the wet-in-wet areas so as to generate satisfactory results. Let δ_h and δ_i be the hue difference and intensity difference between the two adjacent regions, respectively. Let $\vec{g}_c$ be the averaged color gradient in the current region under consideration, and $\vec{g}_{cx}$, $\vec{g}_{cy}$ stand for $\vec{g}_c$ in x, y directions. The wet-in-wet technique is applied if one of the following conditions is satisfied:

- $||\vec{g}_{cx}|| \geq 0.1$ and $||\vec{g}_{cy}|| \geq 0.1$;
- Inside salient regions and $\delta_h < 20°$;
- Outside salient regions and $\delta_h < 90°$.

These rules are default settings. To enhance interaction and user control, our system provides a parameter *wet-in-wet mode* that has four settings: 1) default; 2) no wet-in-wet effect near the boundaries of salient regions; 3) no wet-in-wet effect inside the salient regions; and 4) no wet-in-wet for the whole image. In most cases the default setting generates satisfactory results. The process of creating the wet-in-wet effect is shown in Figure 11.

3.5 Hand Tremor Effect

A watercolorist hardly draws straight lines without breaks or wiggling [30], [31], [32], because human muscles can hardly be controlled accurately due to the noise in the human nervous system [9]. The hand tremor effect is visually different from the micro-scale distortion caused by paper roughness [4]. It has not been investigated in previous methods for watercolorization. According to comments from several professional artists, this is often one of the most crucial factors that distinguish a real watercolor painting from a computer-generated one.

We have implemented this important effect in our system. The hand tremor effect can be realized as *overlaps and gaps*, as shown in the areas A in Figure 12, where the boundaries of adjacent color regions are distorted and therefore do not match exactly. But for

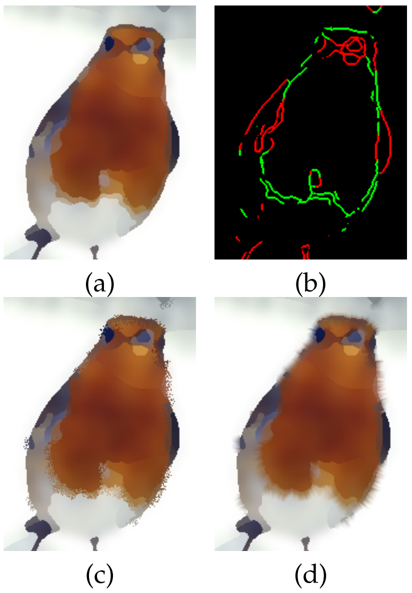


Figure 11. The process of creating the wet-in-wet effect. (a) The input. (b) Edge detection using sobel operator. The areas along the the green lines are determined to apply wet-in-wet effect. (c) Scattering noise pixels on the brighter side. (d) The final wet-in-wet effect.

the areas B, the darker color is painted over the brighter color since it would not mess the colors.

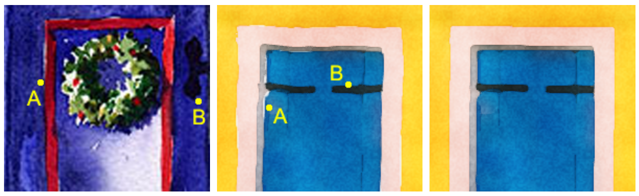


Figure 12. Hand tremor effect in real watercolor paintings (left) and our result with (middle) and without (right) this effect.

In our implementation, region boundaries are classified into the following three groups with regard to the application of the hand tremor effect.

1) Boundaries in a wet-in-wet area should not be distorted;
2) Boundaries of regions with similar hues are distorted without overlaps and gaps;
3) The other boundaries are distorted with overlaps and gaps, as supposed to be generated by the hand tremor effect.

The wet-in-wet effect is applied only to the first type of boundaries. For the second type of boundaries, based on the wave analysis of tissue noise [33], we use a low-frequency fractal Perlin noise texture to simulate the distortion effect. The new color $f_1'(\boldsymbol{x})$ of pixel $\boldsymbol{x}$ is calculated by:

$$\begin{cases} f_1'(\boldsymbol{x}) = f(\boldsymbol{x} + \boldsymbol{t_1}), \\ \boldsymbol{t_1} = (P_1(\boldsymbol{x}), P_2(\boldsymbol{x})). \end{cases} \tag{1}$$

where $\boldsymbol{t_1}$ is an offset vector computed from two different Perlin noise textures [15] $P_1(\boldsymbol{x})$ and $P_2(\boldsymbol{x})$. From the above equation we get f_1' by *distorting* f.

To simulate the third type of boundaries, we use Equation 2 to compute the new color f_2', where $\oplus$ is an operation for mixing the two colors, where f_A and f_B in the figure denote two virtual textures for illustration. We use gradient and color information to find in which region $\boldsymbol{x}$ and $\boldsymbol{x} + \boldsymbol{t}$ are located.

$$\begin{cases} f_2'(\boldsymbol{x}) = f_A(\boldsymbol{x} + \boldsymbol{t_1}) \oplus f_B(\boldsymbol{x} + \boldsymbol{t_2}), \\ \boldsymbol{t_1} = (P_1(\boldsymbol{x}), P_2(\boldsymbol{x})), \\ \boldsymbol{t_2} = (P_3(\boldsymbol{x}), P_4(\boldsymbol{x})). \end{cases} \tag{2}$$

In implementation, the magnitude of tremor can be adjusted with scale between $\boldsymbol{t_1}$ and $\boldsymbol{t_2}$. The smaller $\boldsymbol{t_i}$ is, the straighter the corresponding boundary is.

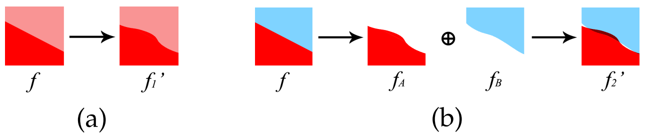


Figure 13. (a) shows the result of applying Equation 1, (b) shows the result of applying Equation 2.

3.6 Other Watercolor Effects

For completeness, we also implemented some other watercolor effects that have been proposed in previous methods [2], [3], [4]. The implementation details can be found in the original references, so are skipped here. These effects include:

- **Edge darkening:** darkened stroke boundaries due to water evaporation;
- **Granulation:** high-frequency boundary distortion due to rough paper surface;
- **Turbulence flow:** low-frequency pigment separation due to uneven water density.

Since pure white and black colors can hardly appear in watercolor paintings, we tune the image with a piecewise continuous function to eliminate these colors before adding the granulation effect. Finally, a post-processing filter [34] is applied for full-screen anti-aliasing. FXAA [35] is applied to produce smooth boundaries.

Figure 14 shows the intermediate results after each step of the pipeline for a specific input image.

4 Results

Figure 15 shows some watercolor paintings produced with our system. For comparison, we have also implemented the filter-based method proposed by

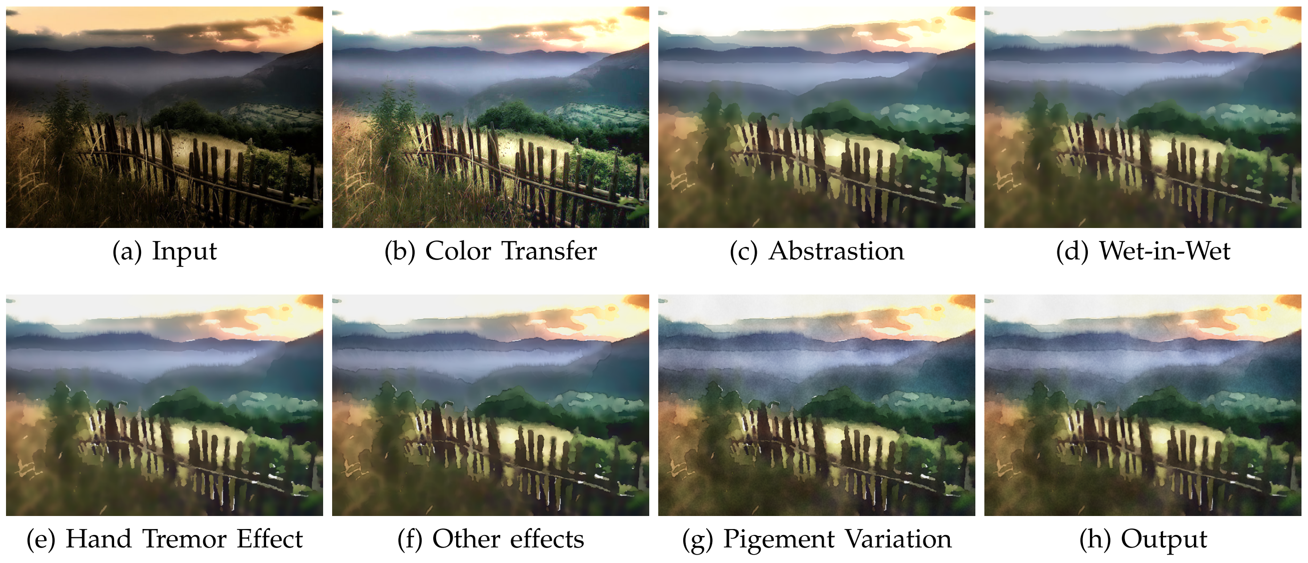


Figure 14. We show the intermediate results at different stage of the pipeline. In (f) *Other effects* we show the result by applying the other effects mentioned in section 3.6.

Bousseau et al. [4] (Figure 17). Note that the morphological filter adopted by their method indiscriminately eliminates small but important regions, such as the details in the center of the train image. The watercolor of the car image in Figure 17 is as dark as the input, a treatment not commonly found in real watercolors. Without considering hand tremor, their results look dry and contain exactly matched boundaries of adjacent regions which are unusual in real watercolor. In comparison, our method produces better results due to the simulation of a variety of key watercolor effects.

Our system runs on an Intel Core i7 3770 processor and an nVidia GeForce GTX 580 video card. It takes about 1.5 seconds to render an image at 800x600, fast enough for interactive application. The computation process scales linearly to the number of the input pixels according to Table 1. The most time-consuming parts are salient region detection and image segmentation, which take up 54% and 19% of the total time, respectively.

Table 1
System performance for image with different scales.

Size (pixels)	4.1×10^4	1.6×10^5	6.5×10^5	2.6×10^7
Performance (seconds)	0.6	0.9	2.2	5.8

User Study To evaluate the artistic-verisimilitude of our results, we conducted a user study online on a social network platform, involving 110 people with different backgrounds, most of whom are students. These people were divided into the professional group (37 people with training in painting for more than two years and familiar with watercolor) and the non-professional group (the others). The images used for the user study also include result images by other methods from the original publications.

In the first part of the user study, we compare artistic verisimilitude of the results of our method and that of the methods by Bousseau et al. [4] and Johan et al. [11], which are chosen as the representatives of filter-based and stroke-based watercolor rendering methods. The images were collected from the original papers, the Internet or generated by our method, and are displayed in a random order. The participants were asked to score each painting on the scale of 0 to 10, where “0” means “this is definitely generated by computer” and “10” means “this is definitely a real watercolor painting”. The results are shown in Figure 16. In the non-professional group, our results get nearly the same score as real watercolors. In the professional group, the score for the real paintings is higher, since presumably the professionals have a more discerning eye. Nevertheless, with this group of the professionals, the scores of our results are still measurably higher than those of real watercolors.

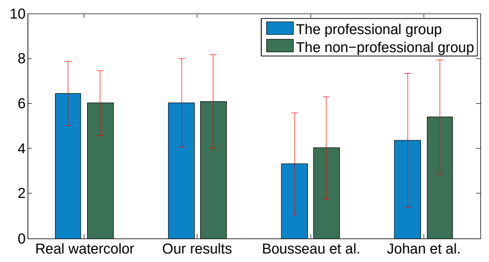


Figure 16. The averaged scores and standard deviations (marked by the red lines).

In the second part, every participant was asked to

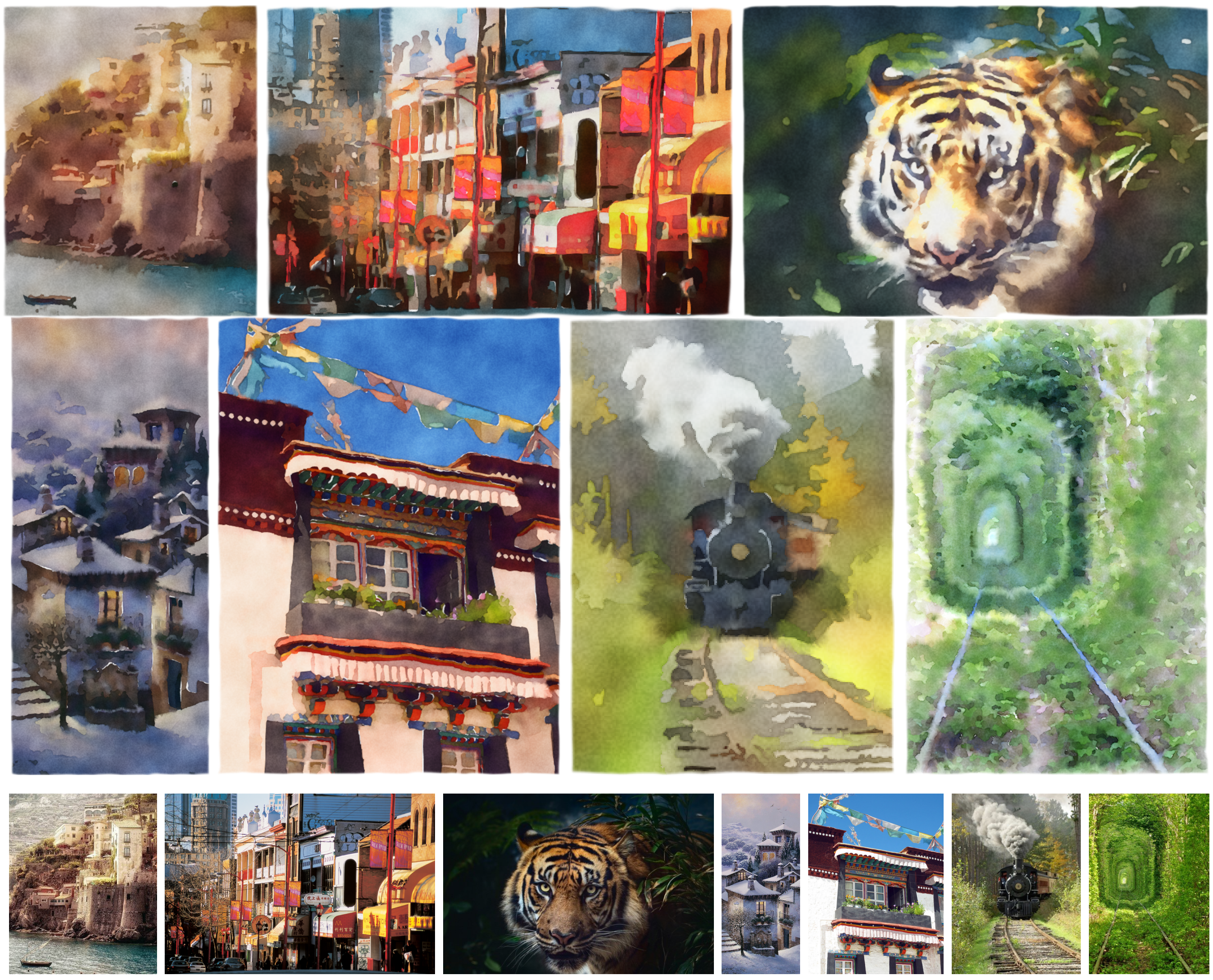


Figure 15. Watercolor paintings generated with our method. The inputs are shown in the bottom row.

vote between our results and the results generated from other methods, including Curtis et al. [6], which is the representative method of physical-based watercolorization. The results from the professional group in Figure 18 show that our method received higher scores than the others.

5 Conclusion and Future Work

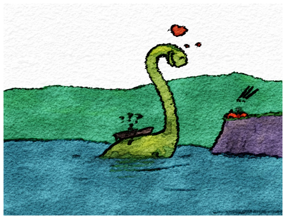
We have proposed a system for automatically synthesizing watercolor paintings from photos. Our system adjusts the color of the input image to achieve a better watercolor appearance and detects the salient scenes to emphasize important objects while abstracting other less important parts, following the practice of watercolorists. Our method also simulates the wet-in-wet effect in a simple and efficient way. Furthermore, we proposed a technique to emulate the hand tremor effect characteristic to real watercolor painting. A user study indicated that the effects proposed in our system are favored by both the professional artists and ordinary users.

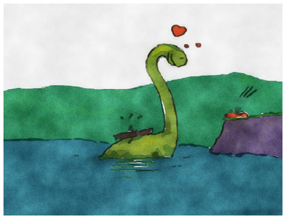
Future Work There are more watercolor effects that can be integrated into our system such as dry brush, splattering, alcohol and salt effects, etc. We stipulate that an exemplar-driven approach such as Lu et al. [36] might be an appropriate solution to extracting the principles for selecting and applying these complex watercolor effects.

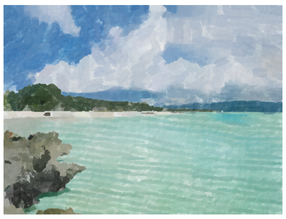
An extension to our work would be video watercolorization. The main challenge there is the frame coherence problem facing frame segmentation, saliency-based abstraction and effect control. This problem may be resolved by applying an efficient labeling scheme for optical flow, as done by Lang et al. [35].

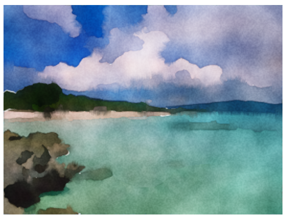
Acknowledgment

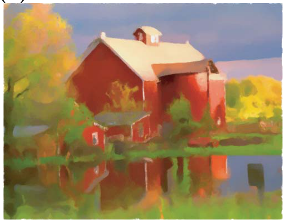
This work was supported by National Basic Research Program of China (2010CB328001), National Science Foundation of China (61373071), National High-tech R&D Program (2012AA040902), and Open Project Program of the National Laboratory of Pattern Recognition (NLPR).

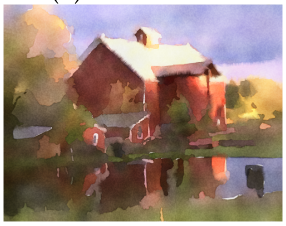

(a) Bousseau et al.: 5.4%

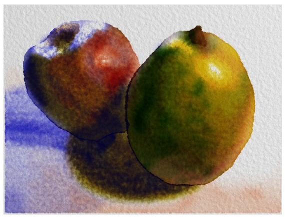

(b) Ours: 94.6%


(c) Johan et al.: 29.7%


(d) Ours: 70.3%


(e) Kang et al.: 5.4%


(f) Ours: 94.6%


(g) Curtis et al.: 23.8%

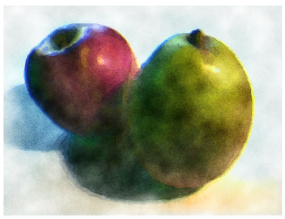

(h) Ours: 76.2%

Figure 18. Vote results from the professional group, who were also asked for some comments: compared with (a), (b) is more saturated; compared with (c), (d) looks more real with the wet-in-wet effect and the distorted boundaries, and has some white space and dirty color mixtures as expected in real watercolor; compared with (e), (f) is better for its rich detail levels; finally, compared with (g), (h) looks better since it has a more natural outline, especially in the shadows and dark regions. The original input images can be found in their publications.

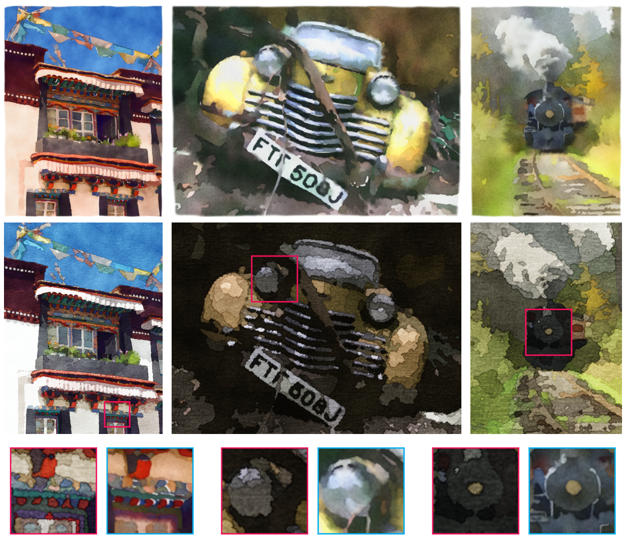


Figure 17. Bousseau et al. [4]'s results. Selected details (in red frames) are compared with the details of our results (in blue frames) in the bottom row. The input images are shown in Figures 1 and 15, respectively.

REFERENCES

[1] E. B. Lum and K. L. Ma, "Non-photorealistic rendering using watercolor inspired textures and illumination," in *Proceedings of the PG '01*. IEEE, 2001, pp. 322–330.

[2] E. Lei and C.-F. Chang, "Real-time rendering of watercolor effects for virtual environments," in *Proceedings of the PCM '04*, 2004, pp. 474–481.

[3] T. Luft and O. Deussen, "Interactive watercolor animations," in *Poster of the PG '05*, 2005, pp. 7–9.

[4] A. Bousseau, M. Kaplan, J. Thollot, and F. X. Sillion, "Interactive watercolor rendering with temporal coherence and abstraction," in *Proceedings of the NPAR '06*, 2006, pp. 141–149.

[5] A. Bousseau, F. Neyret, J. Thollot, and D. Salesin, "Video watercolorization using bidirectional texture advection," *ACM Trans. Graph.*, vol. 26, no. 3, Jul. 2007.

[6] C. J. Curtis, S. E. Anderson, J. E. Seims, K. W. Fleischer, and D. H. Salesin, "Computer-generated watercolor," in *Proceedings of the SIGGRAPH '97*, 1997, pp. 421–430.

[7] T. Van Laerhoven, J. Liesenborgs, and F. Van Reeth, "Real-time watercolor painting on a distributed paper model," in *Proceedings of the CGI '04*, 2004, pp. 640 –643.

[8] N. S.-H. Chu and C.-L. Tai, "Moxi: real-time ink dispersion in absorbent paper," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 504–511, Jul. 2005.

[9] A. A. Faisal, L. P. Selen, and D. M. Wolpert, "Noise in the nervous system," *Nature Rev. Neurosci.*, vol. 9, no. 4, pp. 292–303, 2008.

[10] D. Small, "Modeling watercolor by simulating diffusion, pigment, and paper fibers," in *Proceedings of SPIE '91*, vol. 1460, 1991, pp. 140–146.

[11] H. Johan, R. Hashimoto, and T. Nishita, "Creating watercolor style images taking into account painting techniques," *J. Jpn. Soc. Art Sci.*, vol. 3, no. 4, pp. 207–215, 2004.

[12] H. W. Kang, C. K. Chui, and U. K. Chakraborty, "A unified scheme for adaptive stroke-based rendering," *Visual Comput.*, vol. 22, no. 9, pp. 814–824, 2006.

[13] S. DiVerdi, A. Krishnaswamy, R. Mech, and D. Ito, "A lightweight, procedural, vector watercolor painting engine," in *Proceedings of the I3D '12*, 2012, pp. 63–70.

[14] ——, "Painting with polygons: A procedural watercolor engine," *IEEE Trans. Vis. Comput. Graph.*, vol. 19, no. 5, pp. 723–735, 2013.

[15] K. Perlin, "An image synthesizer," in *Proceedings of the SIGGRAPH '85*, 1985, pp. 287–296.

[16] A. Hertzmann, "Paint by relaxation," in *Proceedings of the CGI '01*, 2001, pp. 47–54.

[17] D. DeCarlo and A. Santella, "Stylization and abstraction of photographs," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 769–776, Jul. 2002.

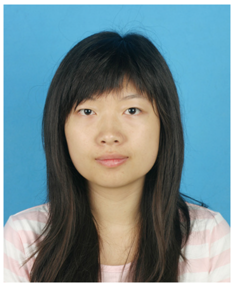
[18] F. Cole, D. DeCarlo, A. Finkelstein, K. Kin, K. Morley, and A. Santella, "Directing gaze in 3d models with stylized focus," in *Proceedings of the EGSR '06*, vol. 2, 2006.

[19] K. Zeng, M. Zhao, C. Xiong, and S.-C. Zhu, "From image parsing to painterly rendering," *ACM Trans. Graph.*, vol. 29, no. 1, pp. 2:1–2:11, 2009.

[20] L. Lin, K. Zeng, H. Lv, Y. Wang, Y. Xu, and S.-C. Zhu, "Painterly animation using video semantics and feature correspondence," in *Proceedings of the NPAR '10*, 2010, pp. 73–80.

[21] J. P. Collomosse and P. M. Hall, "Painterly rendering using image salience," in *Proceedings of the EG '02*, 2002, pp. 122–128.

[22] H. Kang and S. Lee, "Shape-simplifying image abstraction," *Comput. Graph. Forum*, vol. 27, no. 7, pp. 1773–1780, 2008.
[23] M.-M. Cheng, G.-X. Zhang, N. J. Mitra, X. Huang, and S.-M. Hu, "Global contrast based salient region detection," in *Proceedings of the CVPR '11*, 2011, pp. 409–416.
[24] G. Wyszecki and W. S. Stiles, *Color science: concepts and methods, quantitative data and formulae*. Wiley-Interscience, 1982.
[25] B. Guo, S. R. Gunn, R. I. Damper, and J. D. B. Nelson, "Band selection for hyperspectral image classification using mutual information," *IEEE Geosci. Remote Sens. Lett.*, vol. 3, no. 4, pp. 522–526, 2006.
[26] E. Reinhard, M. Ashikhmin, B. Gooch, and P. Shirley, "Color transfer between images," *IEEE Comput. Graph. Appl.*, vol. 21, pp. 34–41, 2001.
[27] E. Reinhard and T. Pouli, "Colour spaces for colour transfer," in *Proceedings of the CCIW '11*, 2011, pp. 1–15.
[28] G. Rong and T.-S. Tan, "Jump flooding in GPU with applications to voronoi diagram and distance transform," in *Proceedings of the I3D '06*, 2006, pp. 109–116.
[29] D. Nehab, A. Maximo, R. S. Lima, and H. Hoppe, "GPU-efficient recursive filtering and summed-area tables," *ACM Trans. Graph.*, vol. 30, no. 6, pp. 176:1–176:12, Dec. 2011.
[30] S. C. Hsu and I. H. Lee, "Drawing and animation using skeletal strokes," in *Proceedings of the CGI '94*. ACM, 1994, pp. 109–118.
[31] C. J. Curtis, "Loose and sketchy animation," in *ACM SIGGRAPH 98 Electronic art and animation catalog*. ACM, 1998, p. 145.
[32] C. Lu, L. Xu, and J. Jia, "Combining sketch and tone for pencil drawing production," in *Proceedings of the NPAR '12*, 2012, pp. 65–73.
[33] K. J. Hayes, "Wave analyses of tissue noise and muscle action potentials," *Journal of Applied Physiology*, vol. 15, no. 4, pp. 749–752, 1960.
[34] J. Jimenez, D. Gutierrez, J. Yang, A. Reshetov, P. Demoreuille, T. Berghoff, C. Perthuis, H. Yu, M. McGuire, T. Lottes *et al.*, "Filtering approaches for real-time anti-aliasing," in *Courses of the SIGGRAPH '11*, 2011, pp. 1–14.
[35] M. Lang, O. Wang, T. Aydin, A. Smolic, and M. Gross, "Practical temporal consistency for image-based graphics applications," *ACM Trans. Graph.*, vol. 31, no. 4, p. 34, 2012.
[36] J. Lu, C. Barnes, S. DiVerdi, and A. Finkelstein, "Realbrush: Painting with examples of physical media," *ACM Trans. Graph.*, vol. 32, no. 4, 2013.


Miaoyi Wang received her B.S. degree in 2008 and her master degree in 2012 from School of Software, Tsinghua University. Currently she is working at NetEase, Inc. as a 3D game engine developer. Her research interest is Non-photorealistic rendering.

Bin Wang is currently an associate professor at School of Software, Tsinghua University, China. He received his B.Sc. degree in Chemistry in 1999, and his Ph.D. in Computer Science from Tsinghua University in 2005. He was a research assistant at Department of Computer Science, Hong Kong University, and had postdoctoral research training at ISA/ALICE Research Group, INRIA-LORIA, France. His research interests include photorealistic rendering, non-photorealistic rendering, and 3D model retrieval. (http://cgcad.thss.tsinghua.edu.cn/wangbin/)

Yun Fei is a student pursuing his master degree in the computer science department of Columbia University. Previously, He got his bachelor degree in the School of Software of Tsinghua University, Beijing, China. His interests cover physical simulation, rendering and GPU-acceleration. (http://cgcad.thss.tsinghua.edu.cn/feiyun/)

Kanglai Qian received his B.S. degree from School of Software, Tsinghua University in 2012. Currently he is pursuing his master degree in Department of Computer Science and Technology, Tsinghua University. His research interest is rendering, GPU acceleration and image processing.

Wenping Wang is currently a professor of Computer Science at the University of Hong Kong, China. He received his B.Sc. and M.Eng. degrees in Computer Science from Shandong University, China, in 1983 and 1986, respectively, and his PhD in Computer Science from University of Alberta, Canada, in 1992. His research interests include computer graphics, geometric computing, and computational geometry. (http://www.csis.hku.hk/~wenping/).

Jiating Chen received his B.Sc. degree from School of Software, Tsinghua University in 2008, and his Ph.D. degree from Department of Computer Science and Technology, Tsinghua University in 2013. His research interests include texture synthesis, stochastic sampling, and photorealistic rendering.

Junhai Yong is currently a professor in School of Software at Tsinghua University, China. He received his B.S. and Ph.D. in computer science from the Tsinghua University, China, in 1996 and 2001, respectively. He held a visiting researcher position in the Department of Computer Science at Hong Kong University of Science & Technology in 2000. He was a post doctoral fellow in the Department of Computer Science at the University of Kentucky from 2000 to

2002. His research interests include computer-aided design and computer graphics.