Image-Based Chinese Water-Ink Style Rendering with Color

Water-Ink Rendering

NAME: Zhiping (Patricia) Xiao
SID: 24965096

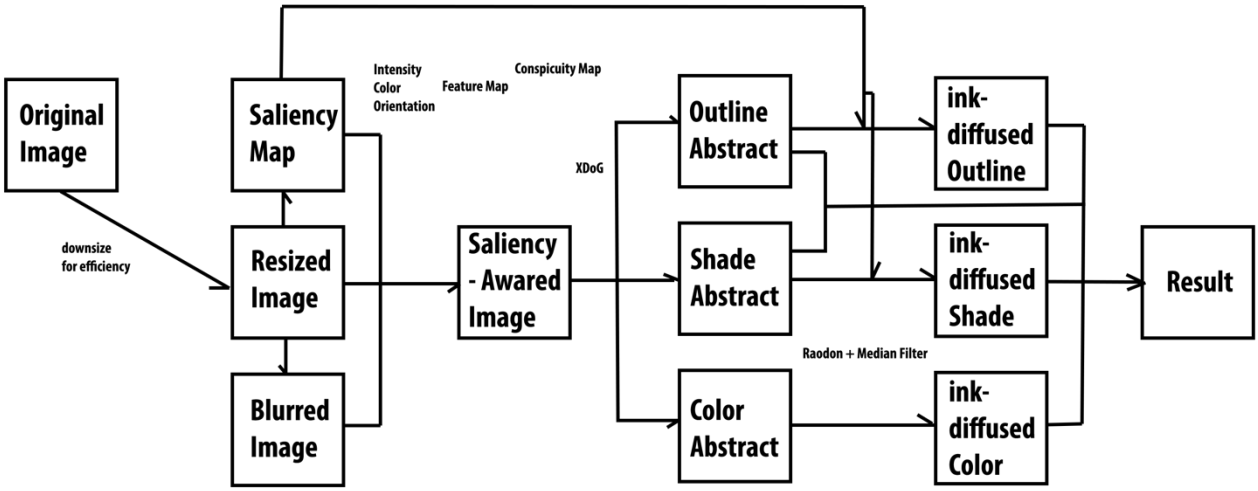


Fig 1: Pipeline

Abstract

Water-Ink painting is a Chinese traditional painting style. Most of the water-ink paintings seem to be simple while actually not. Generally speaking, it is not so easy to render a given image into this style. Although there are some existing researches related to this painting style, study on image-based water-ink style rendering is rarely seen up till now.

In this project, I have:

- Designed a pipeline (Figure 1) of rendering an image to water-ink style;
- Implemented a system to render any input image into water-ink style painting;
- Promoted its performance and analyzed the results;
- Given some ideas about possible future improvement.

Please refer to the following sessions for details.

Key Words: Water-Ink Painting, Image-Based Rendering

I. Introduction

Water-Ink painting is a Chinese traditional painting style. Compared with other forms of art, it is abstract, and is somehow hard to simulate with computing. The reason is quite complex, including that, there hasn't been adequate quantitative studies on Chinese folk art, and that it remains highly reliant on personal sense beyond words, such as "feelings", and "artistic conception", to create the ancient style of charm.

(a) elaborate-style (b) freehand-style

Fig 2: Categories

There are two major categories of water-ink paintings, one is more accurate and detailed, called elaborate-style water-ink painting, and the other is extremely abstract, called freehand-style water-ink painting.

At the first glance, elaborate-style painting seems to be more difficult to implement either manually or by computer, while the truth is the opposite. For elaborate-style painting, all you need is enough patience. The way you draw the flowers, the branches, the leaves, the birds, or human being's body parts, have strict rules to follow. Green hand artists could refer to some picture copybooks, or manuals, called Hua Pu (画谱) (see Figure 3), to do a great job, and the experienced artists have a large collection of figures in their minds to refer to – some of them are professional enough to publish their own Hua Pu for others to learn from.

However, on the other hand, the freehand style is hard to do since artists are given too much freedom to create. There are some abstract rules, such as remaining adequate empty space for imagination, called Liu Bai (留白), while the rules are never clear enough to be followed accurately.

Fig 3: Elaborate-Style References

As is shown in Figure 2, the two styles have obvious differences compared with each other.

Could computer do water-ink painting? The answer is obviously yes. Generally speaking, elaborate-style painting could be done by the following steps:

1. Collect adequate elaborate-style data;
2. Abstract the features of the items in the input image;
3. Figure out what type of items are they and choose a proper "expression" from the elaborate-style dataset;
4. Coloring the image.

As far as I can see, machine-learning would be extremely useful in elaborate-style rendering.

But how about freehand painting style? The answer is "yes" again. And we don't even bother using machine-learning to do it. That is to say, good results could be achieved via a simple pipeline of processing. The structure of the pipeline is shown above in Figure 1.

In this project, I will focus on freehand-style water-ink rendering. Generally speaking, I used saliency map, extended difference of Gaussian (XDoG) edge detection, and ink-diffusion to create the result. The whole pipeline could be divided into 4 phases in brief:

1. Pre-processing: reshape and blur some parts of the original image (according to the saliency map) to create a new image to use in latter steps;

2. Separating: abstract “color” part and “ink” part separately from the processed image;
3. Diffusion simulating: simulate the effect of ink diffusion and color diffusion separately (saliency map is used again);
4. Combining: put the ink part and the color part together to get a final result.

There could be an additional step afterwards, phase 5: after-processing, including adding paper texture, adding poems & stamps, etc. I implemented a paper-texture blending reusing mixed blending I’ve implemented in CS294-26, project 4 previously, and it turns out to be nice, as is shown in the Results part.

In the following sections, I’ll introduce my method in details. Related Work section contains a brief introduction of the papers and studies that inspires me in this project; in Method section, I will walk you through each phase and each step of my pipeline in detail, providing you with some explanation about the reasons behind the decisions as well; in the Results part I’ll show you some of the images my system produced; finally, in the Discussion section I’ll conclude my work and talk about future work.

II. Related Work

Before starting, I searched for existing research papers related to my topic, and hope to learn from them.

Not surprisingly, most of the papers are in Chinese, and also, the majority of them is not image-based – they are 3D rendering instead – where the methods used to abstract edges, colors and shadings are totally different from this case I’m in. Many early researches applied user-defined patterns or texture maps, and hand-made effect were simulated by utilizing brush stroke textures. The main feedback of these methods is that they rely too much on users and datasets.

Luckily enough, I managed to find a paper written by Lixing D. et al [1], which gives a pipeline of image-based real-time water-ink blending. In this paper, the authors stated that:

> *Chinese ink painting, also known as ink and wash painting, is a technically demanding art form. Creating Chinese ink paintings usually requires great skill, concentration, and years of training.*

I am quite thankful for that they inspired me to use Saliency Map, XDoG, and gave me the algorithm of ink diffusion simulation in detail. Before their work, saliency map has already been applied to watercolor style rendering, but not water-ink style.

However, I didn't follow their pipeline, for that I have different understandings of water-ink paintings.

First of all, the authors, and many other researchers, shouldn't have assumed that water-ink paintings are colorless. It doesn't make sense. Actually, most of the color-ink paintings do use ink as the main tool, while they still use colors as a very important assistive feature of expressing. Had they asked any professional artist in advance, they should be told that gray-scale water-ink painting is rarely seen. Or, in other words, gray-scale water-ink paintings are mostly works of practice. In formal paintings or paintings given away to others as gifts, seldom do people use ink-only paintings.

Besides, I can't agree with that they did ink-diffusion equally to the whole image. The spirit of water-ink painting is abstracting and emphasizing, no one would treat every point in an image equally. Instead, by controlling the amount of water at the root of the brush hair, artists are able to apply clear contour to some parts and make some other parts vague.

There are some other less significant opinions they had that I don't agree with. After all, I decided to learn from their ideas and create my own pipeline of rendering.

My contribution is a novel pipeline that takes all aspects: colors, shadings, outlines, saliencies, etc. of an input image into consideration, and create a water-ink style output image. Saliency map, XDoG, ink-diffusion and mixed gradient blending are used.

XDoG, or extended difference of Gaussian, is based on the simplest DoG (difference of Gaussian) edge detection method, which is quite familiar to us after the semester of study. Borrowing ideas from Holger et al [2], I simplified it and used it to detect outlines and shadings.

There are many materials online about saliency map. I found a useful technical explanation here: https://www.tu-chemnitz.de/informatik/KI/scripts/ws0910/Attention_Saliency.pdf , which is extremely helpful for understanding and implementing Saliency Map. Reading the two papers mentioned [3, 4] also helps a lot.

III. Method

Phase 1: Pre-Processing

Before starting to abstract colors, lines and shadings from the image, I pre-processed the image to generate a saliency-aware image.

First of all, I resize the image to a limited size. I could guarantee its performance this way, otherwise the algorithm might be running really slowly. I resize the image to no more than 1000 pixels in width and no more than 1000 pixels in height.

The next step is to calculate the saliency map of the resized image (Figure 4). The saliency map is not so hard to implement, costs little and indeed quite useful. Generally speaking, you just build a Gaussian pyramid, calculate its features and their significance from colors, intensity and orientations perspectives, and finally use the results to generate a saliency map – it is a 2D matrix, but I converted it to an image.

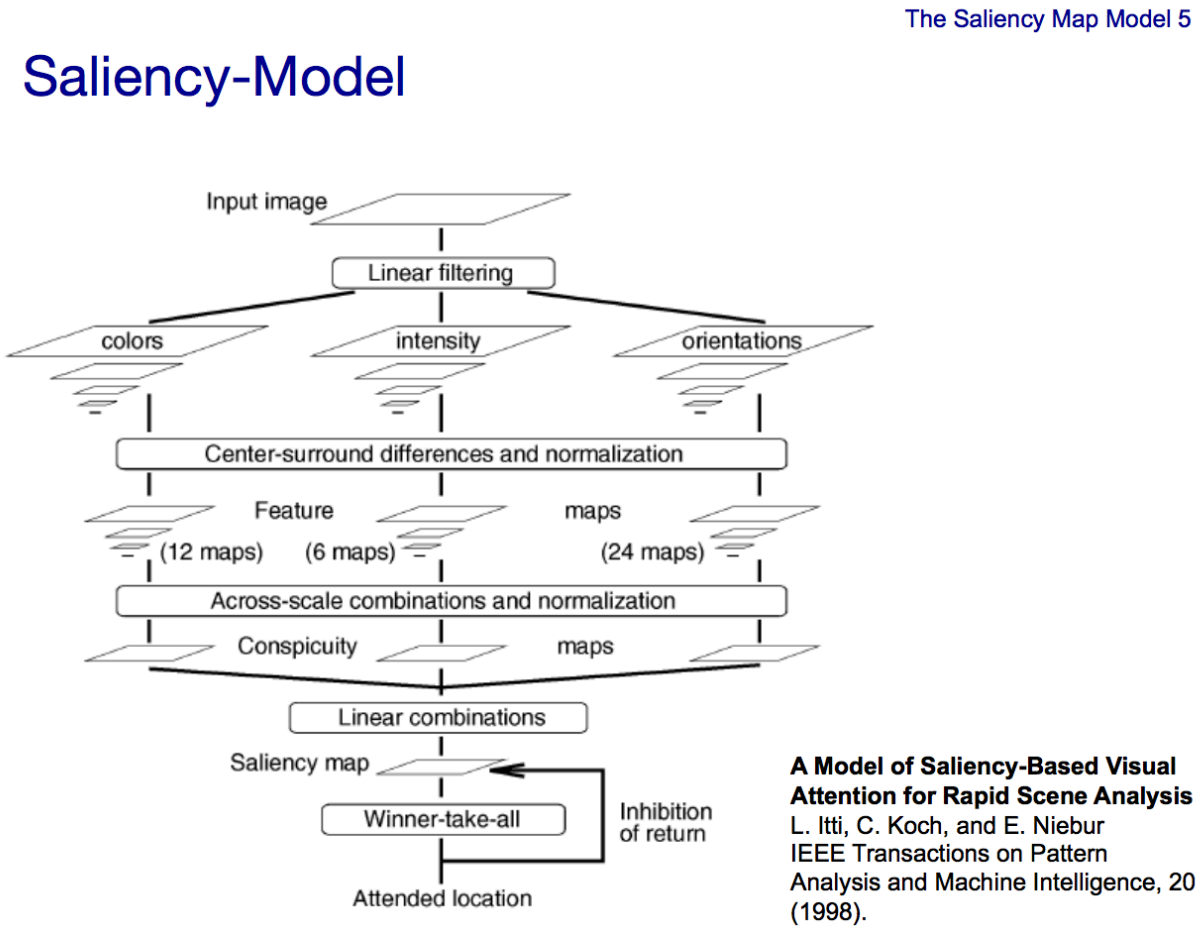


Fig 4: Saliency Model

Then, I'm goanna create a saliency-aware image.

I have tried three ways of generating the saliency-aware image:

1. Do saliency-aware Gaussian filtering: kernel size automatically adapted to the corresponding value of the pixel in saliency map;
2. Blend blurred image and the resized image together according to the saliency map;
3. Blend color-segmented image and the resized image together according to the saliency map;
4. Blend the resized image and color white / average color of the image, according to the saliency map.

After testing all of them, I decided that No.2: blending blurred image and the resized image to generate a saliency-aware image, has the best outcome. As is shown below (Figure 5):

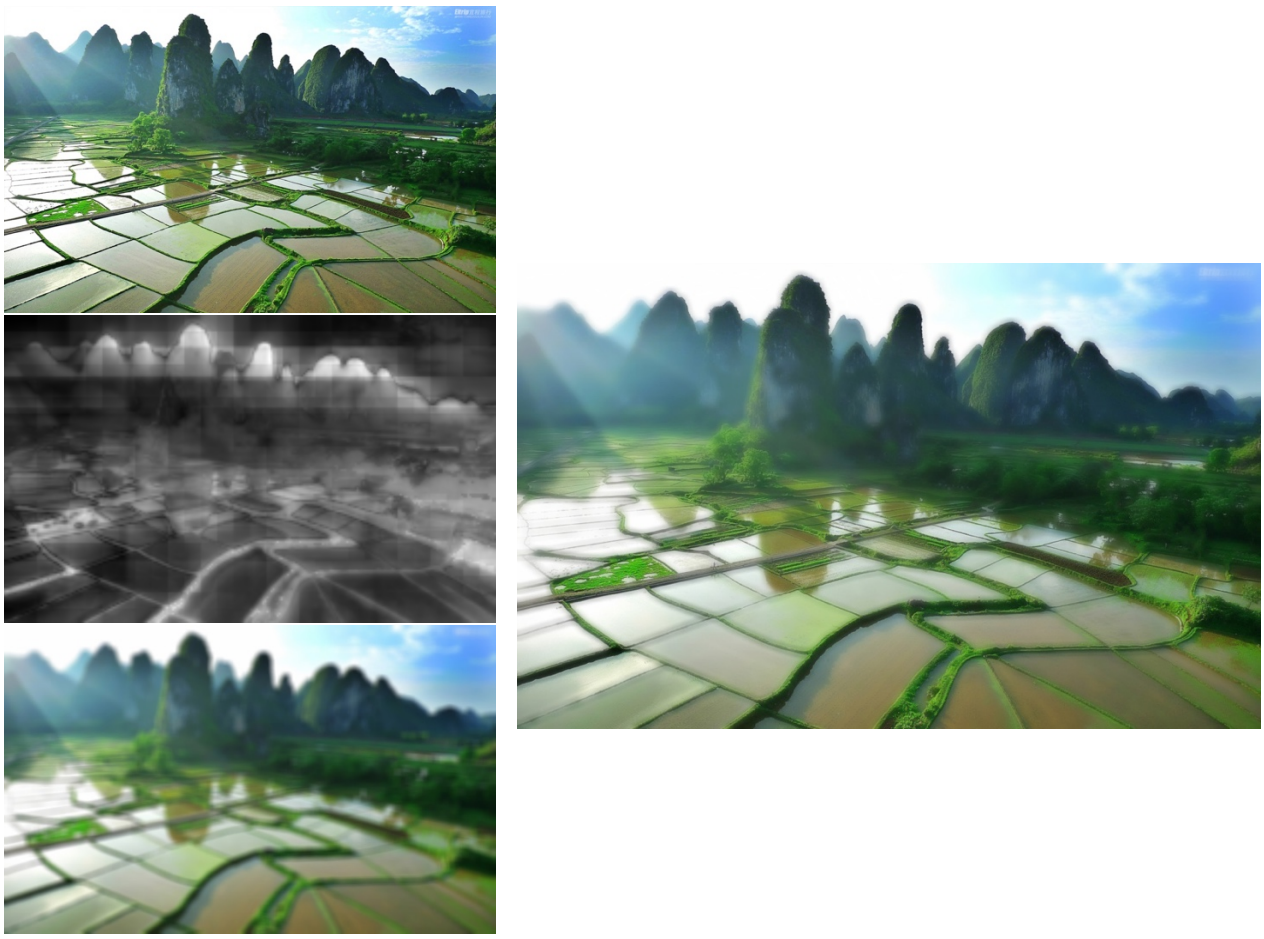


Fig 5: Saliency-Aware Image

Phase 2: Separating

In this phase, color, outline and shade are separated and treated differently afterwards.

For color part, the original image is transferred from BGR field (file i/o used OpenCV) to LAB field, and the colors are segmented according to the L value. L varies from 0 to 100, so I segmented them into range of 10.

The outcome is quite good, as is shown below in Figure 6.

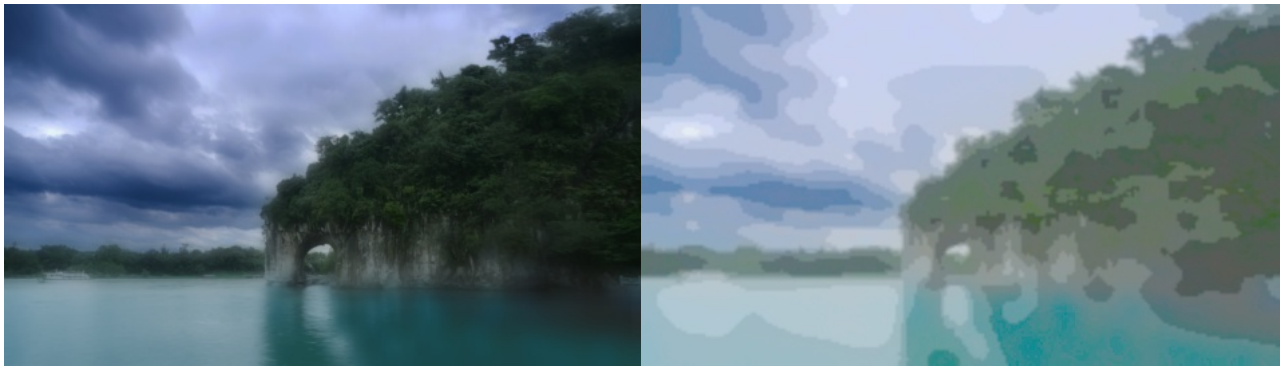


Fig 6: Color Segmenting (left: saliency-aware image; right: segmented image)

However, simply segmenting the color is not enough. As I previously mentioned, water-ink painting tend to use empty space. Therefore, instead of coloring it realistically, it would be better if the colors are turned shallow. I need to find a function, namely **f(x)**, where **x** refers to a value of a pixel (R/G/B), to map the colors, so that: (1) f (0) = 0; (2) f (1) = 1; (3) f is a continuous function in range [0, 1]; (4) f is an increasing function in range [0, 1]; (5) while x ∈ [0, 1], f (x) ∈ [0, 1].

I have tried linear function, sigmoid function, power function, exponential function, log function, etc., and their combos. Finally, I figured out that a piecewise quadratic function would perform the best.

The function f's expression is:

$$f(x) = \begin{cases} -\frac{1}{a^2}(x-a)^2 + 1 & (x < a) \\ 1 & (x \geq a) \end{cases}$$

And the result of the above case (Figure 6) is as shown in Figure 7.

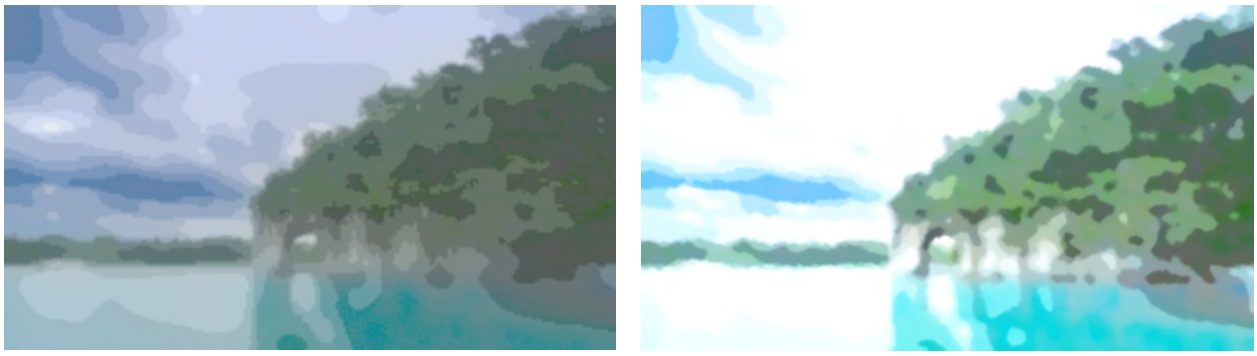


Fig 7: Re-mapping the color (left: before; right: after)

Shades and outlines are extracted based on difference of Gaussian, but their value is mapped to a different range.

Edge is assigned a Laplacian value of the saliency-aware image; using it, we can get the image sharpened, by adding the outline to the original image.

Using the edge and the sharpened image, outline and shading could be calculated accordingly.

Outline is generated using the edge (Laplacian). First, generate a gray-scale version of the edge, assume that the gray-scale image is G_1. Generate a new image L with only the outline, by a transformation function F, $\forall$ u = G_1[i, j], L [i, j] = F(u).

$$\mathrm{F(u)} = 1 - \tanh(\varphi \times u) \qquad (\varphi > 0)$$

Shading is generated using sharpened image, according to the reference paper [2]. First, generate a gray-scale sharpened image G_2. Generate a new image S which contains only the shadings from the image G_2, transformed by a function T, $\forall$ u = G_2[i, j], S [i, j] = T(u).

$$T(u) = \begin{cases} 1 & (u \geq \varepsilon) \\ 1 + \tanh(\varphi \times (u - \varepsilon)) & (u < \varepsilon) \end{cases}$$

An example of the extracted outline and shading is as shown below (Figure 8):

Fig 8: XDoG results (left: outline; right: shading)

Phase 3: Diffusion Simulating

Ink diffusion is quite simple but effective in this system.

I have tried Gaussian Blur combined with other methods (such as simulate the process of spreading by calculating the distance toward the ink points' centers, etc.), while it turns out that this one is the best: simply by (1) for each pixel in the image, randomly pick a point in the nearby square with "radius" (half the length of side) of R, and replace the temporary pixel with the randomly selected one; (2) apply a median filter with kernel size n to the randomized image. Empirically, the best performance is achieved when $R \in [n, 4\times n]$.

The ink-diffusion is applied separately to color, outline and shading.

For the color part, all pixels are treated equally, while for the ink parts (outline & shading), the pixels whose corresponding value in the saliency map is higher (more significant) has smaller n and R value while doing ink-diffusion, so that the concentrated parts look more clearly in the final results (Figure 9).

Fig 9: Before (top) and After (bottom) Ink-Diffusion

Phase 4: Combining

Simple addition of the pixels' value is obviously the opposite of proper in this case. Neither does averaging a good idea. As the goal is to simulate the effect of drawing on paper, colors "addition" should actually be implemented by subtraction.

There are a few websites where we can find instructions on how to simulate the paint blending (instead of light blending):

- http://www.ebaomonthly.com/window/photo/lesson/color_01.htm
- http://www.technotype.net/tutorial/tutorial.php?fileId=%7BImage%20processing%7D&sectionId=%7Bconverting-between-rgb-and-cmyk-color-space%7D
- http://blog.apao.idv.tw/read.php/38.htm

In general, it is easy to reach such a conclusion, that adding black ink to another image means subtracting a certain amount of value from all 3 (RGB) channels.

Assume that we have: C for color (after segmenting, re-mapping and ink-diffusion), S for shading before ink-diffusion, SD for shading after ink-diffusion, LD for outline after ink-diffusion, and parameters $\alpha < 0$, $\beta < 0$, $\gamma < 0$.

Therefore, the final result is:

$$\alpha \times S + \beta \times SD + \gamma \times LD + C$$

Phase 5 (*): Paper Texture

The way I implemented paper texture stuff is to use mixed gradient blending implemented in project 4 (https://inst.eecs.berkeley.edu/~cs194-26/fa16/upload/files/proj4g/cs194-26-add/mixed_gradient.html).

Source image is the result image generated by my pipeline, and the paper texture background is the target image. For the temporary version, the paper texture image need to be provided by user. In latter iterations, it is possible to generate a background of any size automatically.

** not necessarily included in the pipeline*

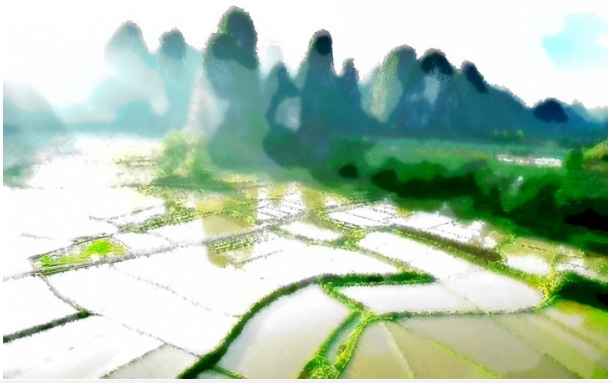
IV. Results

Generally speaking, the outcome is quite satisfactory. Here are some outcomes and the corresponding original images.

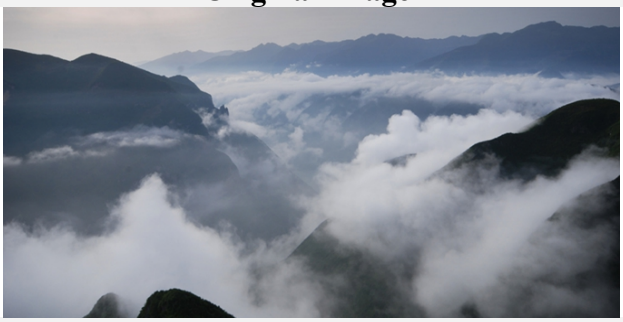
Original Image

Result Image

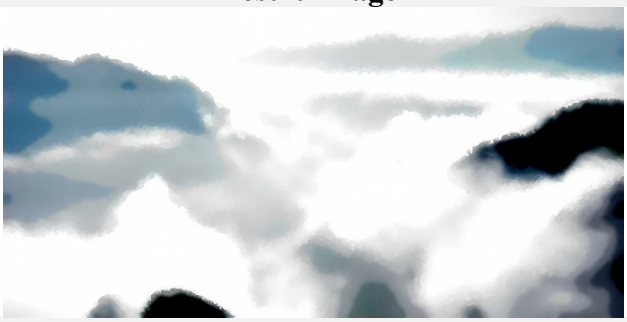
Original Image



Result Image

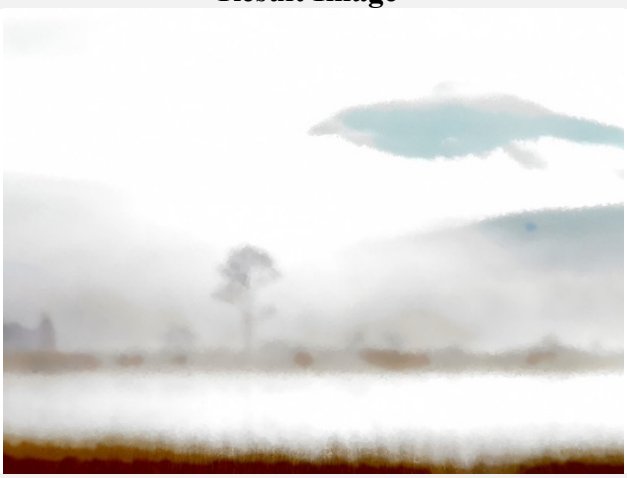


Original Image

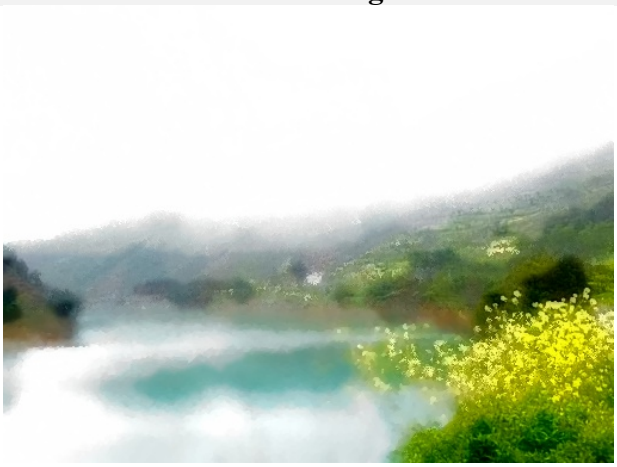


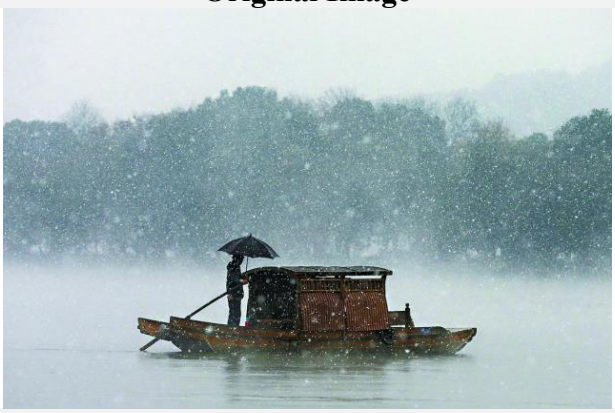
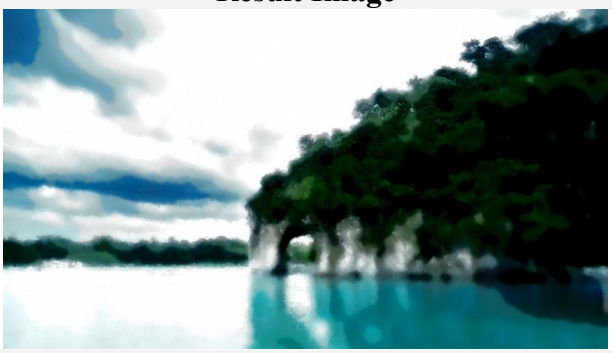
Result Image

Original Image



Result Image



Original Image	Result Image
	
Original Image	**Result Image**
	
Original Image	**Result Image**
	
Original Image	**Result Image**

An example of paper-texture blending is as shown below:

Original Image (left) and processed image (right)

Paper Texture Added

You can see that by using mixed blending, the white (highlights) parts within a painting could be remained, which is so good in this case, for water-ink paintings do use white paints frequently.

V. Discussion

1. Conclusion

In this project, I successfully implemented a system that takes in an image and transfer it into a free-hand water-ink style image, with color, and paper texture (optional).

By increasing the significance of ink usage and decreasing the significance of color, the system managed to get rid of the watercolor style outcomes, which once happened to some of the input images with dark colors.

The outcome is quite pleasing, I'm quite sure that this system could do much better than most people. On the other hand, not surprisingly, it is not good enough to be compared with human experts.

Except for the succeed attempts I made, which are described in detail in the Method section, there are also some attempts ended up in failure.

2. Failed Attempts

Not all the strategies are suitable for this rendering pipeline. Some of the trying and failures are listed below, as a brief reminder and a useful reference of further improvements possible.

	Results	Reason of Failure
Using Seam Carving	Easy to implement, but useless	It could help reduce the image size in most of the cases, but not so much, especially when the goal image size is smaller than every single area in the original image; plus, in color-ink, "empty space" is the most important stuff. Seam carving is taking away the empty space.
Using a Finite Color Set	Couldn't be implemented in such limited time	I thought this would be easy but actually no. Image set couldn't be finite… Any colors could mix with each other, and any one of them could add water or ink…

	Results	Reason of Failure
Speeding up Median Filter	Implemented but useless	My previous thought that speeding up the process of shifting the windows would help turned out to be wrong. In fact, the main cost should be on calculating the median.

3. Further Improvements

Failures aren't thankless efforts. They've told me something, and that they exist presents me with booming of new ideas about what could be tried next.

First, the system could optimize the layout of an image by using seam carving in another way – increasing the size of the image by expanding the white space. However, simply expanding it by the same way of downsizing an image using seam carving is improper in this case – it'll result in severe distortion. Some optimization should be made to the original algorithm. I've seen similar researches, there should be some advisable algorithms that have already existed.

Second, for now the system could only do the most abstract works, while actually in real water-ink paintings, details are equally important. In other words, some methods should be introduced to improve its performance in details. For the pipeline I have for now, "details" means photo-realistic lines, so it shouldn't be implemented this way; instead, using machine learning to abstract features and to find the best match from a large dataset of brushstroke / items might be a good idea.

Third, creating a unique water-ink style color set is worth trying. My failure this time is heavily due to the strict time limit. If I have time to continue this project, I think I would like to figure out how to decide "legal" colors of Chinese water-ink painting. The reason is that traditionally there are rules of using certain colors for certain items in water-ink painting. For example, you should paint a girl's finger tips a little bit red, although her hands might not really look that way.

Also, the outline and the shadings are all based on thin lines, although looks good with ink diffusion, they shouldn't be that way. I think some method of expanding thin lines to be wider and smooth would help.

Finally, what I wanted to do most was to improve the system's performance in doing portrait for humans. However, capturing facial expressions and features and creating a water-ink style face accordingly without machine learning's help is way more difficult than I expected. Although I tried

to use an API of face detection (Face Plus Plus, I already used in previous projects), the result turned out to be terrible. I might get back to it with enough knowledge on machine learning someday.

References

1. Lixing D., & Shufang L., & Xiaogang J., Real-Time Image-Based Chinese Ink Painting Rendering, Multimedia Tools and Applications manuscript (Submitted October 1, 2011; Revised March 22, 2012)
2. Holger W., & Jan E. K., & Sven C. O., XDoG: An eXtended difference-of-Gaussians compendium including advanced image stylization, Computers & Graphics, Vol. 36, Issue 6, 2012, pp. 720–753
3. Itti, L., & Koch, C., & Niebur, E. (1998) A model of saliency-based visual attention for rapid scene analysis, IEEE Transactions on Pattern Analysis and Machine Intelligence, 20:1254-1259.
4. Itti, L., & Koch, C. (2000) A saliency-based search mechanism for overt and covert shifts of visual attention. Vision Res., 40:1489-1506.