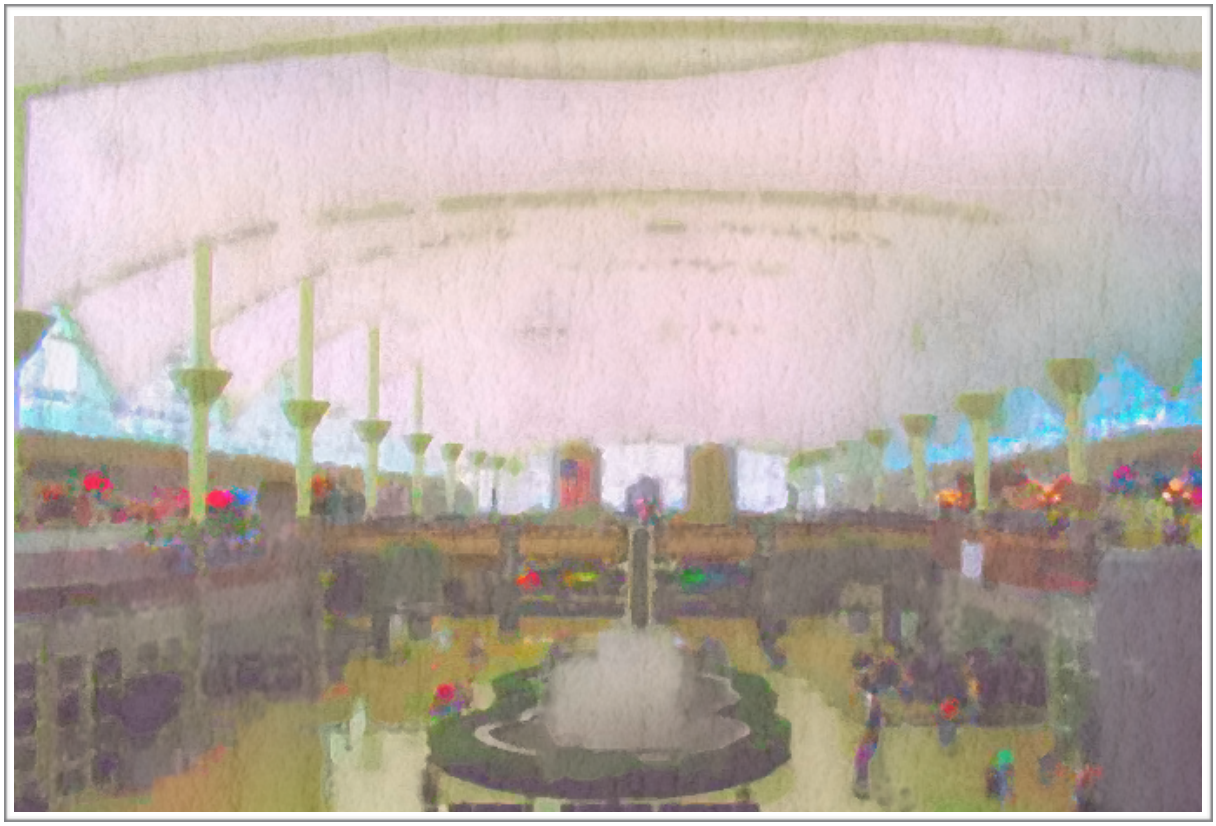
Interactive Computer Graphics
Final Project

Photo Water-Colorization



R05922004

資工所

王于青

Abstract

For my final project, I choose the topic, **style transfer on realistic photographs**, and would like to focus on the style transfer of watercolor. To implement such topic, I read the paper, *Towards Photo Water-colorization with Artistic Verisimilitude in IEEE 2014*[1], and try some methods and algorithms mentioned in that paper. I also replace some procedures with novel algorithms and add some other processing stages to stabilize the quality of the outcome.

Method

The paper mentioned above actually made use of stroke based methods, which is simulating the brushstroke placement process, to accomplish watercolor style transfer. Therefore, we can break the whole transferring procedure into several simulating stages and apply corresponding algorithms to generate the final desired result step by step.

Color Adjustment

To apply color transfer on realistic photographs, I simply collect a bunch of real world watercolor pictures (about 155 collections). Then, I calculate the mean and variance value of these pictures to obtain new color transformation function. With this transformation function, I can map colors in the original photograph to a new watercolor palette.

Photo Abstraction

Since artists would simplify regions that need not be emphasized by abstraction, it's one of the most important part in watercolor style transfer. In the reference paper, the author implemented photo abstraction by applying different size of mean filters on different image regions. To achieve this, the author first calculated the saliency distance of the original photograph, where the value of the distance stands for the importance of each pixel. Also, the author applied mean-shift algorithm to obtain image segments. Then, with the saliency distance and image segments, the author could decide the window size

of the mean filter and apply it on different regions to obtain the abstraction result.

Here I use the same algorithm to do image segmentation. Although mean-shift algorithm would result in ragged boundaries which are not desired in general segmentation tasks, it is actually a great side-effect that can approximate the outcome of brushstrokes. Therefore, I search the existing mean-shift toolkit online and finally use the open source code, Edison, by RIUL team[2].

Since the RIUL team wrapped the algorithm into an executable program that takes a customized command list file as the input (i.e., *.eds), I write code to automatically create the target command list file for each input photograph, and I use system() function to execute the Edison program in order to generate the final mean-shift segmentation result.

However, in stead of using the saliency distance that only contains rough importance information of each pixel, I've tried another idea, scene parsing, to get more accurate information. I was inspired by the topic: ***Scene Parsing Challenge 2016***, and the great results on semantic segmentation was quite impressive. Therefore, I use the MIT Scene Parsing Benchmark[3] and the pre-trained data to obtain the scene parsing result. Then I take the result as my saliency distant map to further calculate the corresponding mean filter size for each image region.

Wet-in-wet Effect

Wet-in-wet effect is also an important characteristic that makes watercolor significantly different from other painting styles. When artists paint with wet brushes, the pigments in wet regions will mix together and produce feather-like patterns along a region boundary.

In the reference paper, the author implemented wet-in-wet effect by randomly scattering seeds around region boundaries and then filtering these areas with an ellipse-shaped kernel oriented along the normal vectors of a region boundary.

Here I use a different way to obtain feather-like patterns along region boundaries. Instead of randomly scattering seeds around region boundaries, I only scatter one seed for each boundary pixel. The distant between each boundary pixel and its corresponding seed is decided by a random number in gaussian distribution. After that, I calculate the edge direction for each boundary pixel, and filter it with an ellipse-shaped kernel to get the feather-like pattern.

With this method, we can get nice wet-in-wet effect without scattering massive random seeds and calculating large amount of normal vectors. Also, I use a thicker ellipse-shaped kernel and apply filtering on only vertical and horizontal directions. It would help reduce more computation time while maintain good wet-in-wet result.

Pigment Variation

In order to create pigment variation effect (especially in flat regions), I apply mean-shift on current result image again to get image segments. Then, I ignore small regions and split the mean-shift image into several binary image layers. I apply binary image erosion to simply divide regions into central areas and boundary areas. Finally, I add more random gaussian noise in boundary areas and apply pixel brightening in central areas to create pigment variation effect.

Texture Blending

In the end, I add a simple feature that is not mentioned in the reference paper. I blend the overall result with texture images common for watercolor papers. With this effect, the result image would be much more like watercolor painting.

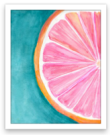
Result

After applying the mean and variance value calculated from my watercolor collection, we can see that the color tone, contrast, saturation, and temperature of the input photograph are transferred to the watercolor palette. The below is a part of my watercolor template collection.

Input

Color Adjustment

57896587959628051 09.jpg



a760a81c4b4b022541 2385bf36222792.jpg

a56212a5f950fe744df 7d6d58438228f.jpg

ABB6EED8FBF23970D D79A5DAB..._912.JPEG

aea2922b7c6f23cc1d0 aba176ba73b85.jpg

af172b6c87e79e53f70 c6783b9ae5989.jpg

b0eda5705448b9cc4f fb9cb5032430da.jpg

b22283e506bf06bf4bf 9e37148f6...836.c1.jpg

b977276d56db200f67 1a5d97c73672fb.jpg

bright-pastels-abstract-b...ground.jpg

c3a0ab6451fe3cc3fe9 ed0440e2a4f13.jpg

d0cc9fe3d09121cc90e 2ae3dbc69375e.jpg

d311c3ca0cbd867d21a 03e63cf8bf21f.jpg

dbe768141ed593607c 07713fbcc9103b.jpg

e2c73ce378f8cf015d0 54ada51511e21.jpg

e6fc0da561e457fb3e3 4c1269e8c90cf.jpg

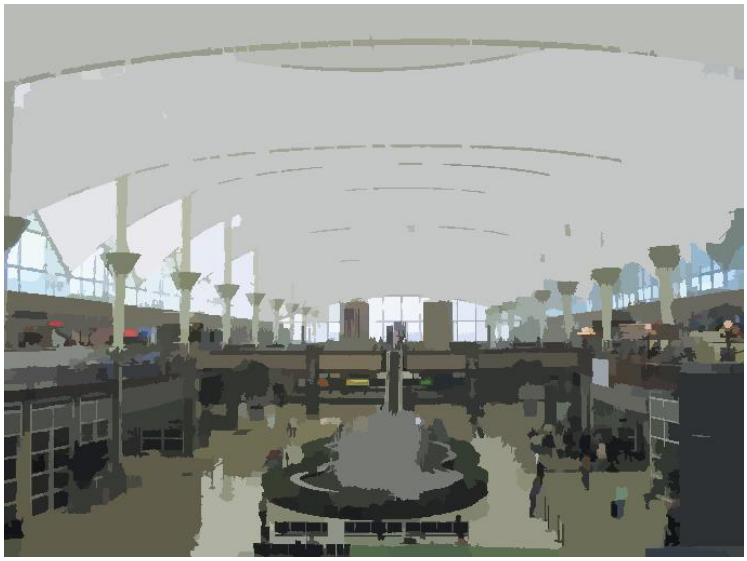
e363a6bf1fdb8571bd2 7df4558f1ae0f.jpg

e590a4f638b4abab00 5cf36376831533.jpg

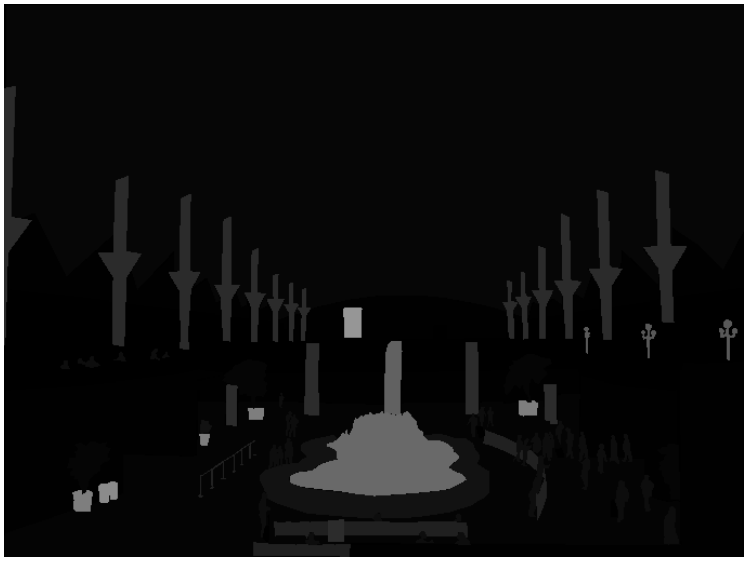
eaf0ead7f9047b4298 5bf2e73dd8a640.jpg

eccc250fc959074509 e05bc68cc72bfa.jpg

After color adjustment, I generate the mean-shift and scene parsing image respectively to calculate the final abstraction result.



Mean-shift Segmentation



Scene Parsing

Color Adjustment

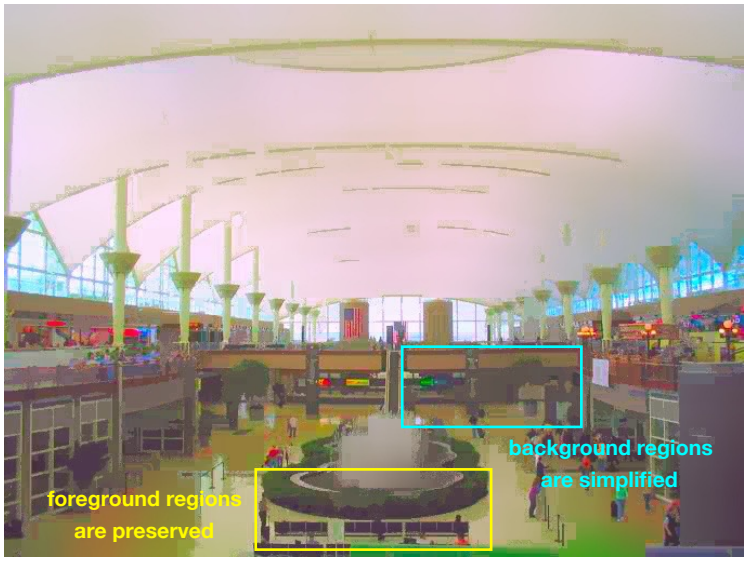
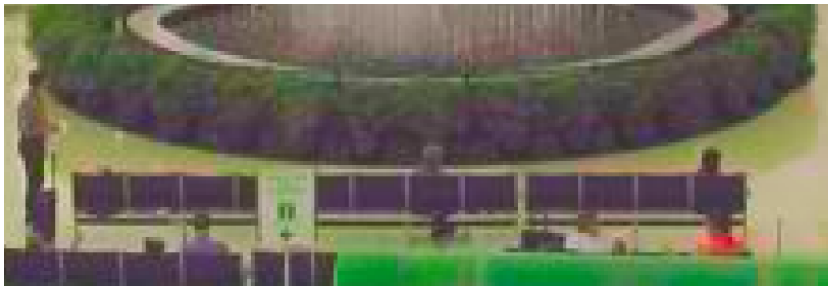
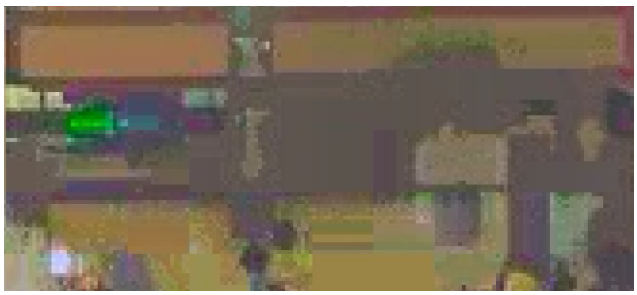


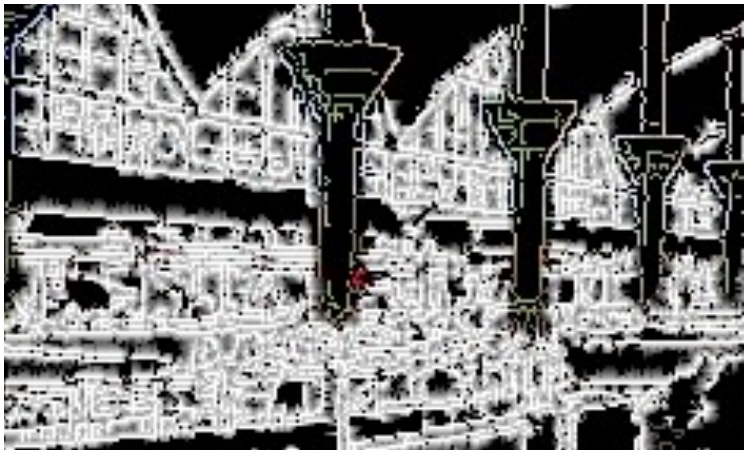
Photo Abstraction





With the abstraction image, I get the boundary image with canny edge detection and apply wet-in-wet effect. I only apply wet-in-wet effect on background boundaries in order to preserve detailed regions. Also, since my wet-in-wet algorithm may create some undesired artifact like dark noise, I apply a built-in openCV de-noising function and an additional median filter to generate better wet-in-wet results.

Canny Edge Detection (Top-left Part)



Scene Parsing (Top-left Part)

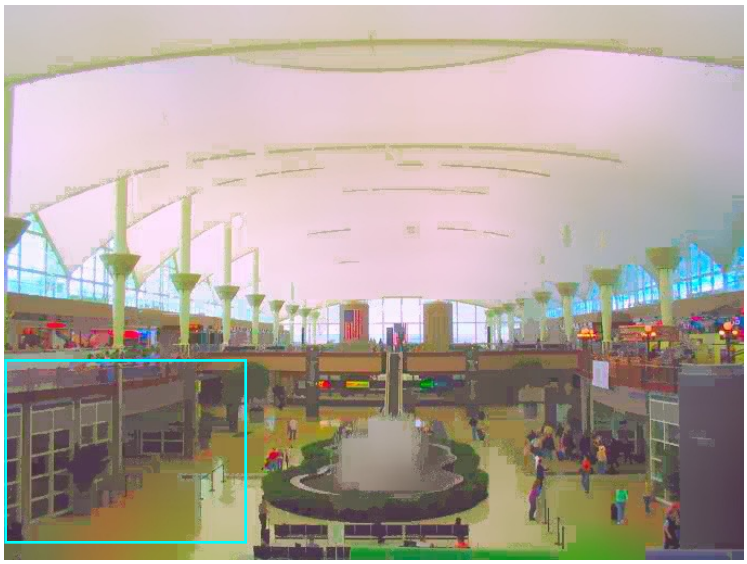
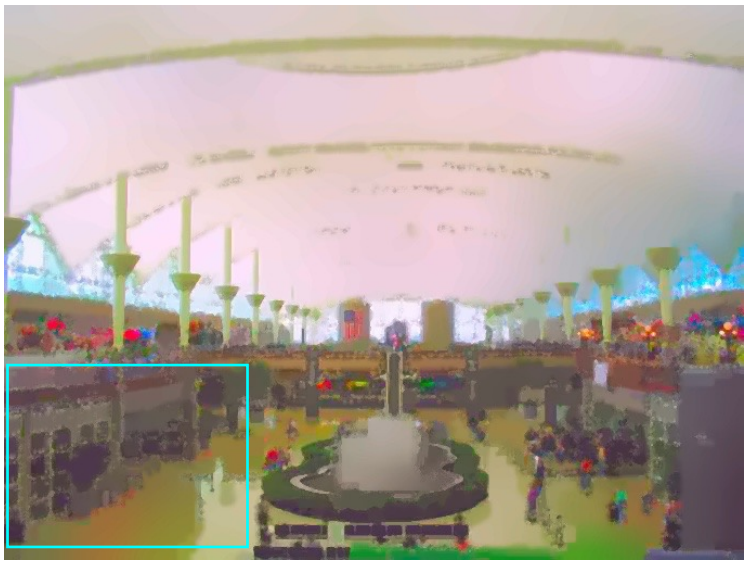
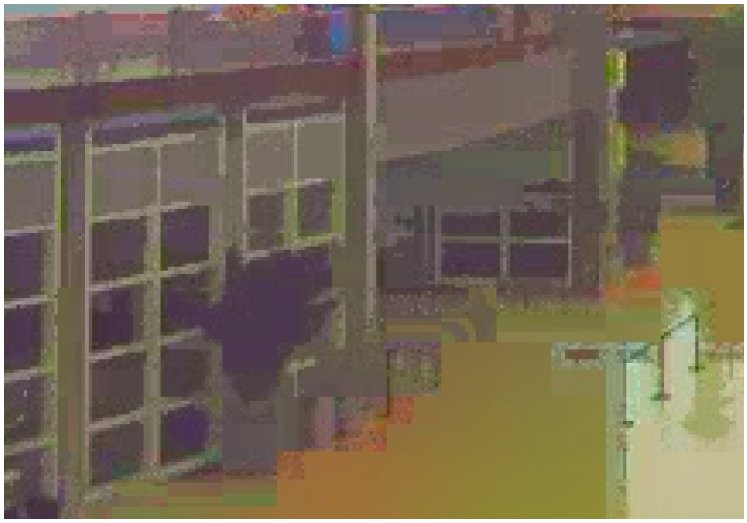
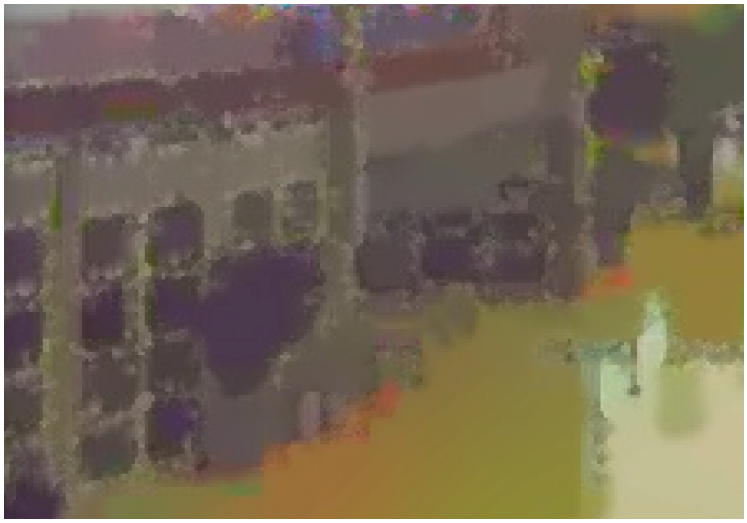


Photo Abstraction



Wet-in-wet Effect





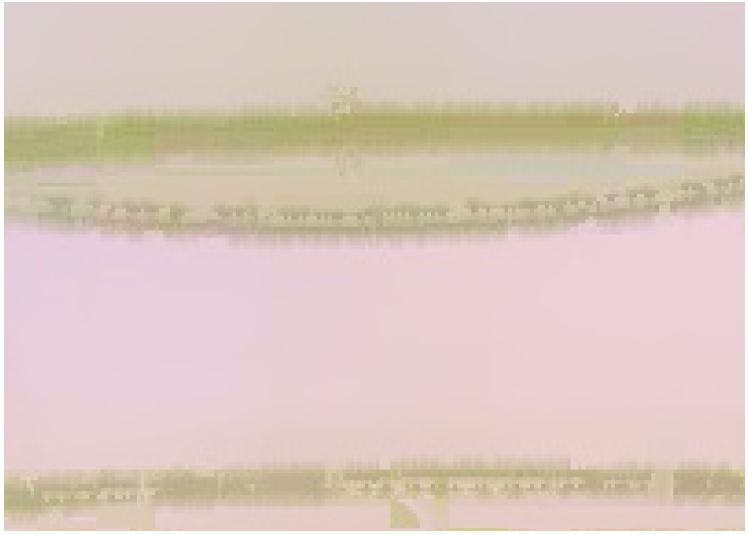
For pigment variation effect, I apply mean-shift again and divide the mean-shift image into segments.

Example Segment (celling)

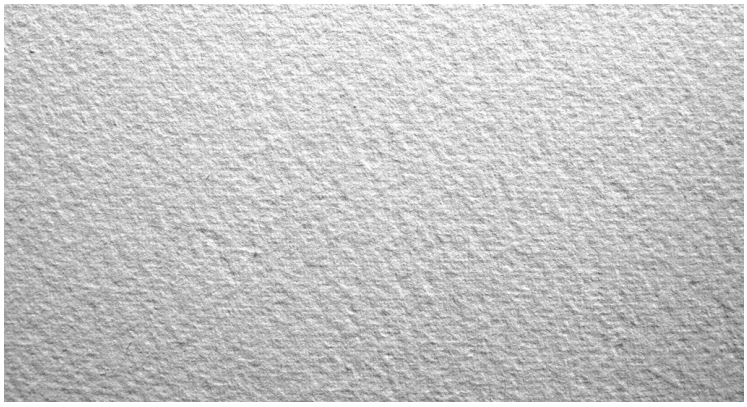
Example Segment (floor)

Wet-in-wet Effect

Pigment Variation Effect

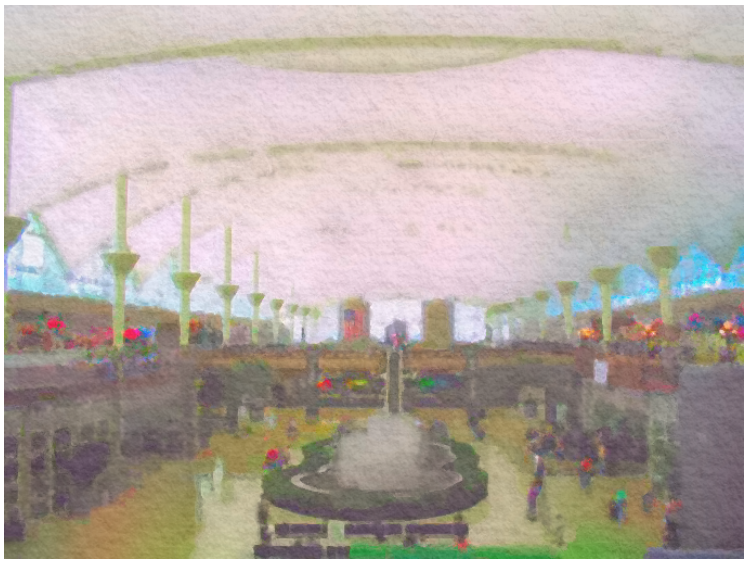


Finally, I blend the image with a paper texture image to generate more realistic watercolor result.

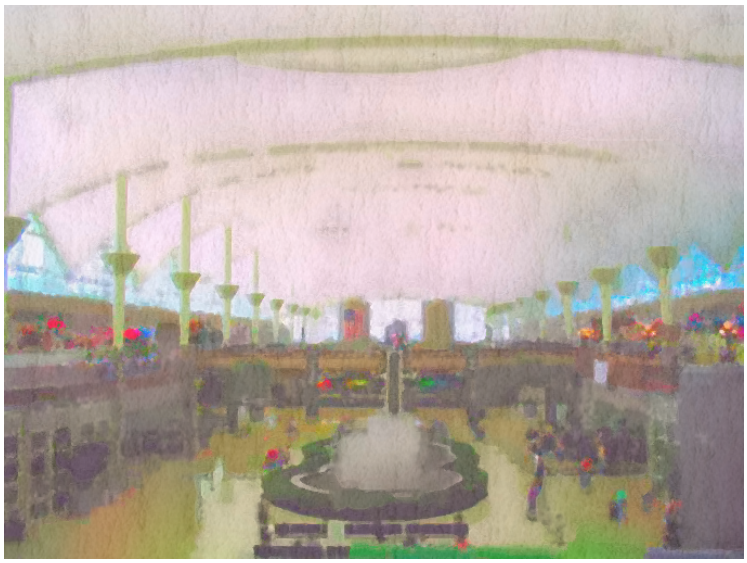


Example Paper Texture 1

Example Paper Texture 2

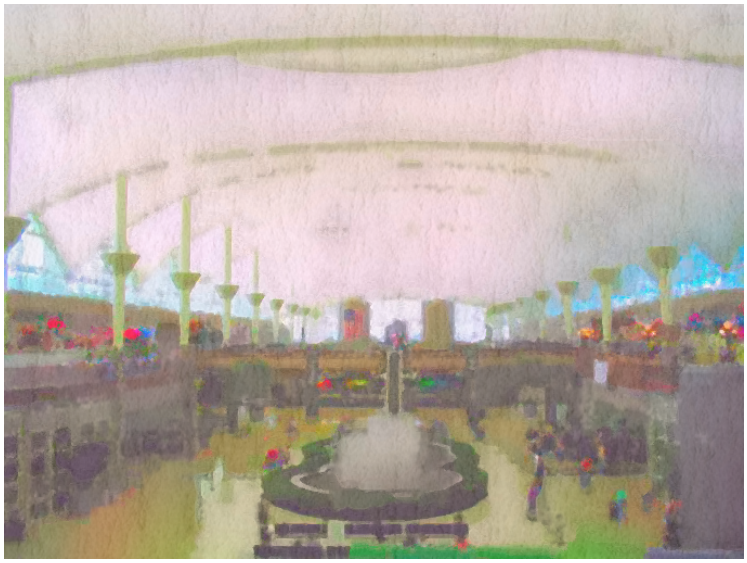


Result with Texture 1



Result with Texture 2

Photograph



Watercolor

Reference

[1] Towards Photo Water-colorization with Artistic Verisimilitude, IEEE 2014
https://www.ncbi.nlm.nih.gov/pubmed/26357391
[2] Code for the Edge Detection and Image SegmentatiON system
http://coewww.rutgers.edu/riul/research/code/EDISON/
[3] MIT Scene Parsing Benchmark
http://sceneparsing.csail.mit.edu/