Photo Painterly Rendering

Kylie Ying

Spring 2019

1 Introduction

If a picture is worth a thousand words, a painting can be worth a million emotions. Unfortunately, for the artistically challenged (like me), capturing an image on an iPhone is a significantly easier task than painting a picture. Luckily, I can program. In this project, I explore an algorithm that can generate an artistic masterpiece, rendering a given photograph as a digital painting.

2 Background

Painterly rendering aims to convert a photographic image into one that appears to have been painted. Paul Haeberli introduces the idea of different brush strokes using colors sampled from a local region on a photograph to render an image as a painting [1]. Barbara Meier discusses rendering animations in a painterly style [3]. By populating a surface with particles and applying brush strokes, she was able to render different styles of painting of the reference image. Furthermore, Aaron Hertzmann tackled the authenticity of the produced "painting", by exploring painterly rendering with curved brush strokes, to create painterly renderings that look more realistic [2].

3 Algorithm

The algorithm used to produce the painterly rendering is as follows. For N iterations, we randomly select a location (x, y). Then, we use rejection sampling based on an importance map, where we choose a number between 0 and 1 and if that number is not greater than importance (x, y), then we skip the iteration. If we use rejection sampling, then we iterate for N divided by average importance iterations to account for the rejected pixels. Otherwise, we get the color of the (x, y) location on the original image (the three RBG pixel values).

We modulate the color by multiplying it by a random amount proportional to noise, then replicate the color in the pattern of texture on the output image centered at (x, y). The texture is an image, so adding the new texture to the output image is done by taking a linear combination of the existing pixel value and the new pixel value for each pixel in the new texture.

For the oriented painterly rendering, the structure tensor was computed for the image and at each point and then the eigenvectors for each point were determined. We kept the minimum value of the angle between the eigenvectors and the horizontal axis for each point, and rotated the texture according to that angle. The rest of the painterly rendering process stays the same.

4 Implementation

I implemented functions that transform images into images that appear painting-like. I implemented a function called `brush` that takes in a location, color, and texture and transforms the output image such that the texture is replicated in the color specified, centered at the location specified. I also implemented a function called `singleScalePaint` that essentially does the aforementioned algorithm. Building off of these, I implemented `painterly`, which calls `singleScalePaint` twice: once using an importance map of all 1s

(i.e. paint anywhere) and using brush size `size`, and again using an importance map where only finer details are prioritized and using brush size `size`/4.

Then, I implemented `computeAngles`, which takes an image, extracts the structure tensor, computes the eigenvectors of that, and finds the angle of the smallest one with respect to the horizontal direction. Using this, I created `singleScaleOrientedPaint`, which does the same as `singleScalePaint`, but takes the angle into account. Similarly, `orientedPaint` does the same as `painterly`, but takes the angle into account.

Taking this one step further, I also implemented an option to paint using a palette, which is essentially using the color scheme of another painting/image. For example, I could input a palette of only blues and my output image would be in only blues. To do this, I basically created a dictionary holding an integer value of the sum of the RGB values multiplied by a constant and using the remainder after dividing by that constant. I implemented `getPalette`, which gets the dictionary of the RGB values, and `transformColor`, which takes a color as an input and returns a color within the palette.

5 Validation

The testing of the implementation was done visually. The output image should retain the basic shapes and colors of the original input (unless done with the palette, in which case the colors would be transformed). Initially, I tested the brush implementation to ensure that the textures would show up correctly, even if they overlapped. I tested with a color with different RGB values to ensure that the color would show up too. The output is as shown in Figure 1.

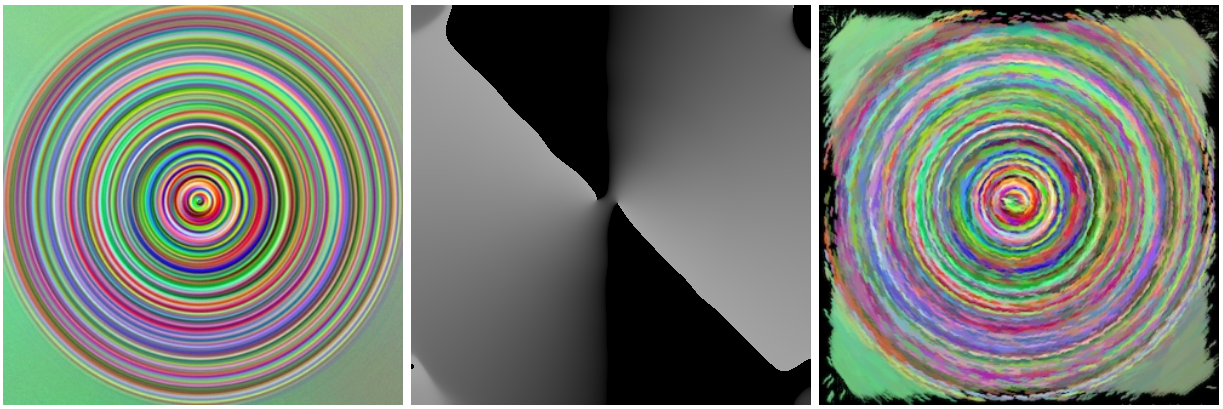
Figure 1: Brush strokes with coloring

Next, I tested the `singleScalePaint` and `painterly` functions. The input and outputs are shown in Figure 2. Then I validated the `computeAngle` function by looking at a map of the intermediate angle and trying the oriented paint on the circle image. The input, angle map, and output are shown in Figure 3.

(a) Input image (b) `singleScalePaint` output (c) `painterly` output

Figure 2: Naive painterly function outputs

Finally, I ran the palette paint image using Starry Night. The inputs and output are shown in Figure 4.



(a) Input image (b) Angled map (c) `orientedPaint` output

Figure 3: Painterly with angle consideration

(a) Input image (b) Starry Night palette input (c) Palette painted output

Figure 4: Painterly with palette

6 Results

The results of the algorithm are shown on many images in Figure 5.

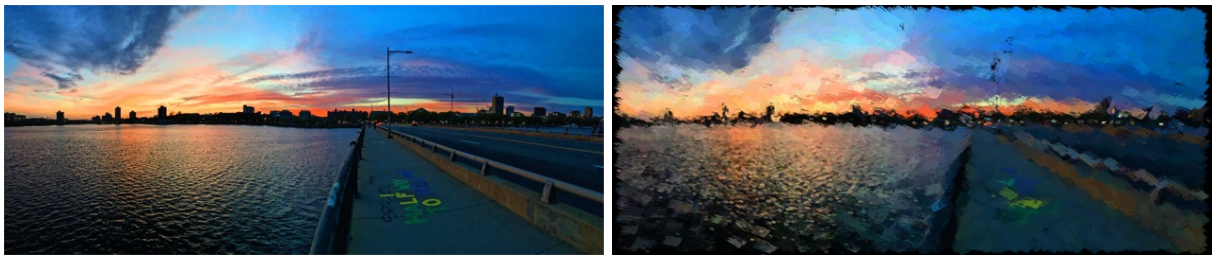
7 References

[1] Paul Haeberli. Paint by numbers: Abstract image representations. In *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '90, page 207–214, New York, NY, USA, 1990. Association for Computing Machinery.

[2] Aaron Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, page 453–460, New York, NY, USA, 1998. Association for Computing Machinery.

[3] Barbara J. Meier. Painterly rendering for animation. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, page 477–484, New York, NY, USA, 1996. Association for Computing Machinery.

(a) Input image (b) Oriented paint (c) Palette output (Starry Night)



(d) Input image (e) Oriented paint

(f) Input image (g) Oriented paint (h) Palette output (red sunset)

(i) Input image (j) Oriented paint (k) Palette output (Starry Night)

(l) Input image (m) Oriented paint (n) Palette output (red sunset)

Figure 5: Results on multiple photographs