Nonphotorealistic Rendering
Advanced Computational Photography

Long Phi Nguyen[1]

Department of Electrical Engineering & Computer Science, Massachusetts Institute of Technology

Abstract. Artists' styles have evolved since the 1800s, ranging from abstract to impressionist to painterly. In this final project, I investigate the implementation of painterly image modification in Python 2.7 using structure tensors, matrices that summarize predominant changes in the gradient of an image. On top of implementing a direction-oriented painterly tool in Python, I added another layer of art style on top, outlining edges black as is done in the video game series *Borderlands*. The technique works well, and images rendered are indeed painterly.

Keywords: 6.865 · photography · art · painterly · style

1 Introduction

This project came from Assignment 11, *Nonphotorealistic Rendering*, of 6.865's Fall 2013 offering. I chose this assignment because as a photographer, I love seeing images in many different editing contexts: more contrast, more pop, more vibrance, et cetera. I was excited to try a project in which I would edit images' *style*. Most tutorials on the Internet use combinations of esoteric functions available in Photoshop to change image style, and this is far too tedious.

In this project, I implemented the following (most of which came from the original assignment):

- basic brushes that sample randomly for where to "splat" on an image using a color and a texture,
- two-scale painterly rendering (coarse and fine strokes) with importance sampling,
- oriented painterly rendering, an extension to the above to follow content direction,
- black outlines around edges to replicate the art style of *Borderlands*.

2 Background

The basic idea behind this concept is artistic style. We want to be able to computationally edit images such that they reflect a variety of art styles—similar to real life artists and the many ways they can express themselves.

In 2014, Wang et al. performed significant research specifically on watercolorizing images [3]. They simulated human idiosyncrasies such as hand tremor and wetness bias to generate realistic watercolor images from natural photos. They used the CIELAB color space for their filters to emulate watercolor.

Modern researchers have also looked to machine learning for intelligent image editing and generation. Gatys, Ecker, and Bethge used deep neural networks to *generate* images from scratch with arbitrary mixes of various artistic styles [1]. In practice, images generated here can easily fool the common eye.

Even more interesting to me is the extension of these deep neural networks towards *sharing photographic styles* between real photographs, known as "deep photo style transfer" [2]. This technique builds on previous work in painterly research to generate photorealistic mixes of styles. Luan et al. constrain the color transformation process to be affine and treat this constraint as an "energy" term to achieve their results.

Regardless of the style generated, each contribution above uses image processing as its primary motivation. This project in particular focuses on the existing work in painterly image generation and reimplements the core technique behind it.

3 The approach: high-level overview

Imagine that we are trying to generate, given some known photograph P, a painterly rendition of it. Let us work with P directly to do this. As a first step, we can randomly decide where we want to smear our brush on the canvas, taking the colors already existing in P. However, doing this with the existing colors only leads to the same exact image. If our brush is pure (dry, colorless, etc.), then this is useless.

To make things more interesting, let us "muddle" our brush a little bit to simulate real conditions. We could specify how willing the brush is to accept certain colors from certain areas of P, or how widely spread our area of brush sampling is, or even how wet it is (beyond the scope of this project). This is where we modify the weights we give to the original colors in P and our own brush—we get a linear combination describing what the output should look like as a painterly image.

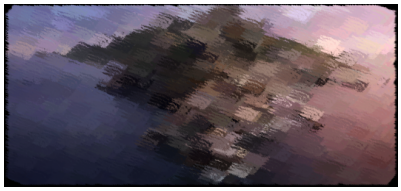
This is, in essence, the basic pipeline behind painterly rendering. We take a brush, apply some characteristics of our choice to it that best resemble real conditions, and add brush strokes to our existing canvas in P.

We must now also account for the direction of content in P. This is where we deal with the *structure tensor*, a matrix that describes gradient changes in local neighborhoods. Using structure tensors across the entire input, we can *orient* our brush as a real artist would in different directions. If we have a circle in our input, our structure tensors would allow us to brush circularly. This optimization lets us achieve painterly renderings while not sacrificing on the integrity of the original image.

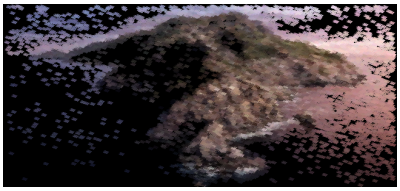
As a last modification, we can take two passes:

1. Coarse: We make large strokes to get an overall muddy painterly rendering of the image.
2. Fine: We make denser, concentrated strokes of small size to preserve details in packed regions.

Doing so allows us to leave big areas like skies and meadows/fields more "sloppy" (since they require less concentrated brushing) and finer details like individual leaves and objects clear, as a real artist might do. See Fig. 1 for the intermediates in one of my own outputs, Catalina Island from macOS.



(a) Coarse pass, large strokes.



(b) Fine pass, preserving dense details.

Fig. 1: A demonstration of the coarse and fine passes (each without the other) when rendering painterly images. Notice that the details in the island are preserved in the isolated second pass.

4 Contribution

4.1 Assignment

As mentioned before, I downloaded Assignment 11 from the Fall 2013 offering of 6.865, implemented it in its entirety, and added one last touch of my own to resemble the game series *Borderlands*. See `https://www.cs.williams.edu/~morgan/SRG1/borderlands.pptx.zip` for slides from SIGGRAPH on realizing *Borderlands*'s art style.

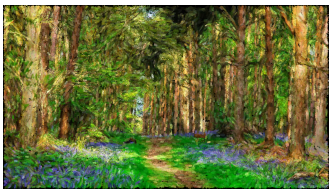
4.2 What my work does

(a) Beautiful forest. 1920x1080.

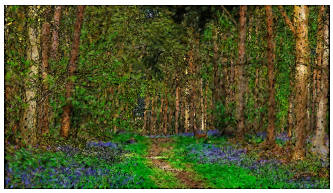
(b) macOS Catalina default wallpaper. 650x300.

Fig. 2: The images I personally ran the project on on top of the provided inputs.

The high-level details behind my work are summarized in Section 3, and the technical details used to implement it are in the Assignment 11 handout. My implementation modifies photos and writes painterly versions of them to disk (see the `Output` folder in my submission). On top of the existing assignment, I wrote 5 new test cases as per the final project specification that include outputting both passes, coarse and fine, during two-scale painterly rendering (`nprTestCases.py`). I also used a Sobel kernel to isolate edges and outline them in black (Fig. 4c).



(a) Forest, painterly style.

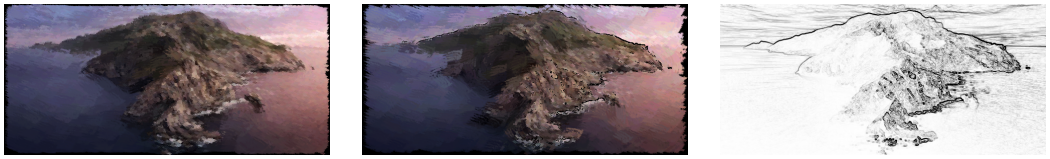


(b) Forest with *Borderlands* style black outlining.

(c) Isolated black outlines.

Fig. 3: The three images I got from using my implementation/extension on a picture of a beautiful forest.

To provide some metrics, I wrapped my test cases with a basic profiler that tells the user how long renders take. As expected, render times increase significantly with image size (see 1920x1080 result). This was in Python and not C++, so I couldn't try Halide as an extension for performance engineering. My times are below.



(a) macOS Catalina wallpaper, painterly style.

(b) With *Borderlands* style black outlining.

(c) The isolated black outlines.

Fig. 4: The three images I got from using my implementation/extension on the macOS Catalina wallpaper.

```
- - - - - - - - - - - - - - - - - - - - - - -
Operating on image of size (533, 800, 3)...
TEST 1: painterly on provided VPD image
  > time to completion: 2.145785s
- - - - - - - - - - - - - - - - - - - - - - -
Operating on image of size (1080, 1920, 3)...
TEST 2: oriented brushing on beautiful forest image
  > time to completion: 51.662745s
- - - - - - - - - - - - - - - - - - - - - - -
Operating on image of size (1080, 1920, 3)...
TEST 3: painterly on forest (outputting intermediate passes)
  > time to completion: 27.769287s
- - - - - - - - - - - - - - - - - - - - - - -
Operating on image of size (300, 650, 3)...
TEST 4: oriented brushing on macOS Catalina wallpaper
  > time to completion: 5.976726s
- - - - - - - - - - - - - - - - - - - - - - -
Operating on image of size (300, 650, 3)...
TEST 5: painterly on Catalina wallpaper (outputting intermediate passes)
  > time to completion: 5.784689s
```

4.3 Difficulties

Understanding the analogous image functions in the Python version of the class was definitely the hardest part of transitioning from C++. I grew used to the nice static typing, and moving to `numpy` for image processing was also challenging. It took a few hours to grasp how to use the libraries to implement the project, but I got it! Results were great, and I learned a lot about subtleties in the painterly art style.

References

1. Gatys, L.A., Ecker, A.S., Bethge, M.: A neural algorithm of artistic style (2015)
2. Luan, F., Paris, S., Shechtman, E., Bala, K.: Deep photo style transfer (2017)
3. Wang, M., Wang, B., Fei, Y., Qian, K., Wang, W., Chen, J., Yong, J.: Towards photo watercolorization with artistic verisimilitude. IEEE Transactions on Visualization and Computer Graphics **20**(10), 1451–1460 (Oct 2014). https://doi.org/10.1109/TVCG.2014.2303984