CSCE448-500 Project 4

Lam Nguyen

March 5, 2023

1 Task 1: 2D Poisson Blending

1.1 Overview

In this assignment, the task is to take a source image, target image, and binary mask and be able to seamlessly stitch these images together to make interesting images. Of course we can simply just cut and paste our source to the target but the result leads to noticeable seams regardless of the how similar or different the backgrounds are. Instead what can be done is maximally preserve the gradients in the source region without changing any of the target pixels via Poisson Blending.

The general idea of the implementation is that we have a matrix A of sparse dimension of height*width x height*width, and vector b of height height*width, and we need to compute them to solve Ax = b. To compute A, we must traverse through all the pixels of the mask and if it is 0 at an index, matrix A would get 1 at that pixel index, and vector b would receive the pixel value of the target. If the value is 1 at an index, matrix A would get 4 and the neighboring pixels would get -1 in A. The value given to vector b is given by the following equation:

$$4 * s(i,j) - s(i-1,j) - s(i+1,j) - s(i,j-1) - s(i,j+1)$$

where s is the source image, and (i,j) are the columns and rows of that image

The implementation of this in the code is rather straightforward with this algorithm but there were some hiccups with the code. Firstly was the speed at which matrix A was computed. My initial approach was to set up a lil_matrix and set values in it appropriately, but I learned that it uses a linked list and traversing to set things would be extremely slow. Instead what I chose to do was pre-compute all necessary values and at the end create my csr_matrix given my data and dimensions. Another issue I came up upon, mostly because of my lack of reading the prompt, was that the source and target images are both colored images, meaning they have RGB channels that need to be accounted for when calculating the vector b. After adjusting my code to calculate a b vector for each channel, all of my errors of setting b[i] to an array of rgb values went away.

The following images in the following subsections are my results from running the given images and my own personal example. Each result taking about 1-2 seconds each to finish thanks to using sparse matrix:

1.2 Results

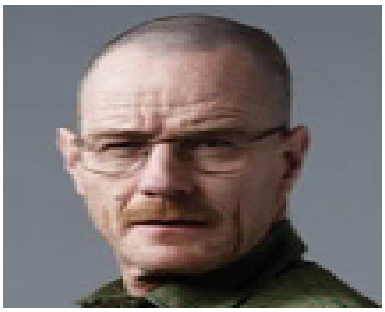
(a) Bear Image

(b) Swimming Child Image

(c) Naive Blend Image

(d) Poisson Blend Image

Figure 1: Result 1



(a) Heisenburg/Mr. Chemistry Image

(b) Mona Lisa Image

(c) Naive Blend Image

(d) Poisson Blend Image

Figure 2: Result 2

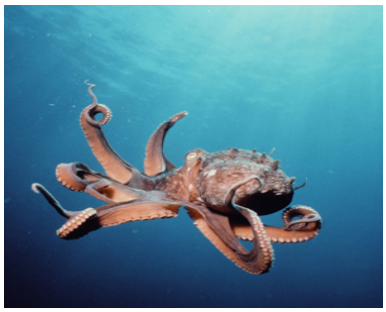
(a) Delorean Image

(b) Winterfell Image

(c) Naive Blend Image

(d) Poisson Blend Image

Figure 3: Result 3



(a) Squid Image

(b) Aircraft Carrier Image

(c) Naive Blend Image

(d) Poisson Blend Image

Figure 4: Result 4

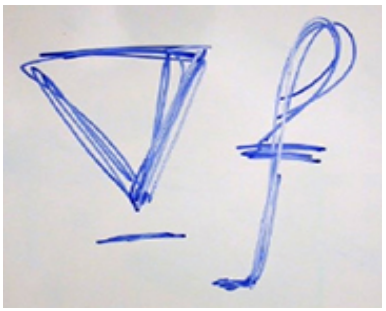
(a) F-16 Image

(b) Sky Image

(c) Naive Blend Image

(d) Poisson Blend Image

Figure 5: Result 5



(a) Gradient of F Image

(b) Wall Image

(c) Naive Blend Image

(d) Poisson Blend Image

Figure 6: Result 6

(a) Rainbow Image

(b) Island Image

(c) Naive Blend Image

(d) Poisson Blend Image

Figure 7: Result 7

(a) John Travolta Confused Image

(b) Godzilla v. Kong Image

(c) Naive Blend Image

(d) Poisson Blend Image

Figure 8: Personal Example

1.3 Analysis

As we can see, the images provided were able to be blended well using poisson blending except maybe figure 6 and 7, but can be fixed via mixing gradients as part of the extra credit I did not do. One thing to note is that poisson blend does work well in happy cases where the all provided images have similar lighting for the most part. In the case where the pixel intensity of the source does not match the target (e.g. dark versus light images), poisson blending may not work well.