CS 194-26: Image Manipulation and Computational Photography, Fall 2022

Project 1: Images of the Russian Empire

Colorizing the Prokudin-Gorskii photo collection

Ethan Gnibus

Overview

Text giving a brief overview of the project, and text describing your approach. If you ran into problems on images, describe how you tried to solve them. The website does not need to be pretty; you just need to explain what you did.

In the early 1900's, Sergei Mikhailovich Prokudin-Gorskii travelled around the Russian Empire with a goal in mind to photograph everything he saw in color. This was before color photography existed, so Sergei planned to take pictures of the same scene three times in a row. One photograph would be taken with a filter over the lens so that only red light would be captured. The second would only capture green light. The third would only capture blue. Sergei hypothesized that one could project all three images together to reconstruct said scene in full color! In this project I aim to take individual R, G, B photos from Sergei's collection, align them using code, then output them as a full color image.

To implement this image-aligning program, I first split the scans of Sergi's photos into three channels: R, G, and B. Next, I implemented a function that aligns two images using exhaustive search. This funcion translates the G and R channels by every (x, y) displacement vector where x is chosen from the range -15 to 15 and y is chosen from the range from -15 to 15. At each displacement vector, I used an image matching metric to score each displacement vector. I implemented Sum of Squared Differences (SSD) and normalized cross-correlation (NCC), but I found that NCC's results were better so I chose to use the latter. After scoring every displacement vector for a channel, I took the one that scored the best, and used it to tranlate the input channel by said vector. This way, the R and G channels can be aligned to the B channel if they were to be ovelapped. I finished up by writing code that took these three individual R, G, and B channels, then combined them to be shown as a full color image.

This implemetation worked for small images (it worked perfectly on 256x256 images for example), but as the dimensions of an image increased, the program took far too long to compute. Not only that, but the quality of alignment dropped and eventually stopped working. This was because the (-15, 15) range of displacement I tested and ranked became obsolete as the image sizes I tested increased since they had lower frequency change between pixels. To account for this, I implemented an image pyramid function to speed up alignment and ensure that it worked across all image sizes.

To construct this image pyramid, I created a mipmap-like collection of rescaled versions of the input channel and the channel to compare it to. I downsampled the input channels by a factor of two, stored them downsampled channels, then repeated the process until the downsampled input channel was small enough to accurately align with the downsampled compare channel using displacement vectors with changes in x being from the range -15 to 15 and changes in y from the range -15 to 15. Once I had the displacement vector at the lowest resolution in the pyramid, I multiplied the displacement vector by a factor of 2, applied the transition to the input channel one resolution layer above it, then tested aligning the input channel at this layer with the output layer at the same layer using displacement vectors with changes in x being from the range -3 to 3 and changes in y from the range -3 to 3. I lowered the ranges of values that we choose from because the adjustments we need to make after aligning one layer below are small (and the program will run faster with a smaller range of tested displacement vectors). I then add these displacements to the previous displacement vector, multipy the displacement vector by a factor of 2, and iteratively repeat the process on all levels of the image pyramid until we reach the highes resolution channels. Once the program reaches the highest resolution channels, they should align nicely when translated by the total accumulated displacement vector. By following this algorithm, I was able to align images that normally took minutes to compute in less than 10 seconds.

This implemetation worked for many images and was fast, but for some images it failed to align images properly. Because of this, I decided to implement edge detection and image normalization to improve my accuracy. See the "Bells and Whistles" section for more details.

Result on example images

The following is the result of my algorithm on all the provided example images:

cathedral.tif

Before displacement

After displacing G and R channels by
G: (2, 4)
R: (2, 12)

church.tif

Before displacement

After displacing G and R channels by
G: (0, 24)
R: (0, 62)

emir.tif

Before displacement

After displacing G and R channels by
G: (24, 48)
R: (40, 106)

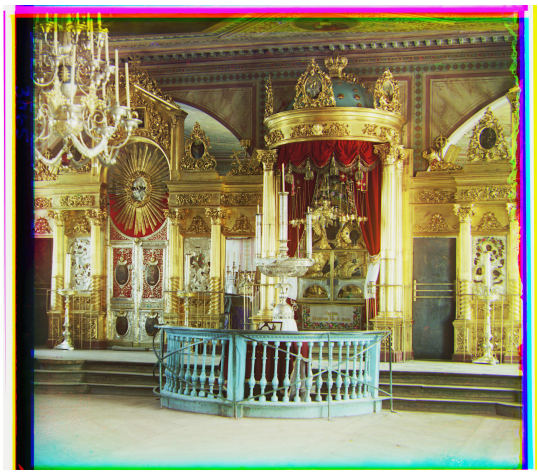
harvesters.tif

Before displacement

After displacing G and R channels by
G: (18, 60)
R: (10, 124)

icon.tif

Before displacement



After displacing G and R channels by
G: (16, 40)
R: (22, 88)

lady.tif

Before displacement

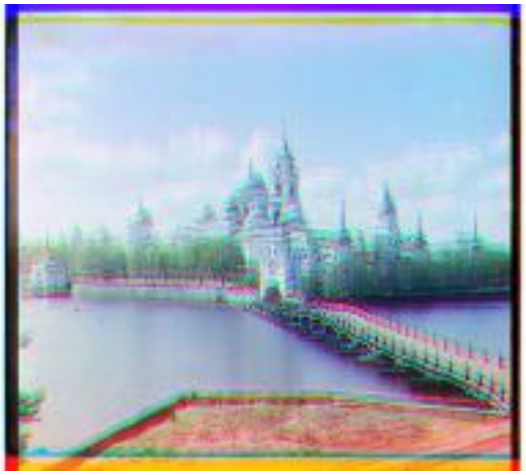
After displacing G and R channels by
G: (10, 56)
R: (12, 120)

melons.tif

Before displacement

After displacing G and R channels by
G: (10, 80)
R: (12, 176)

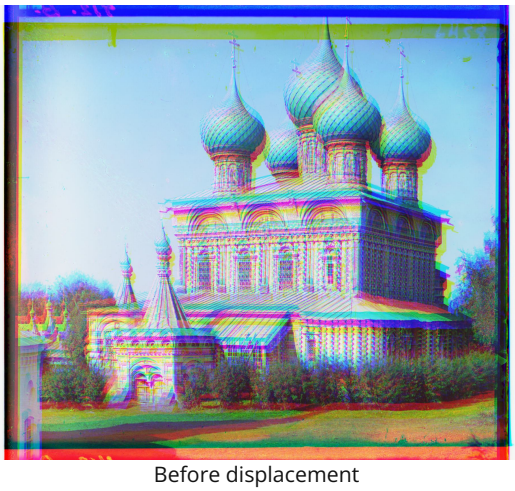
monastery.jpg



Before displacement

After displacing G and R channels by
G: (0, -4)
R: (2, 2)

onion_church.tif



Before displacement

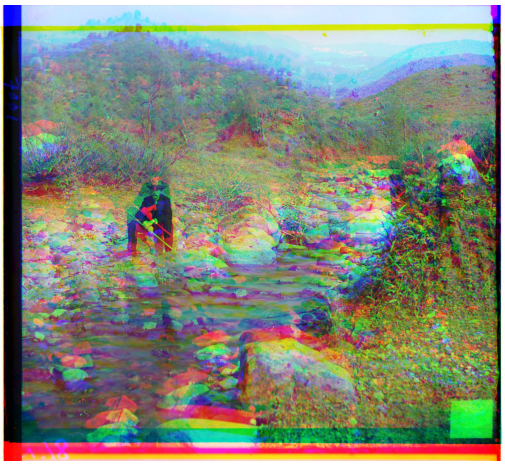
After displacing G and R channels by
G: (26, 52)
R: (36, 108)

sculpture.tif

Before displacement

After displacing G and R channels by
G: (-10, 32)
R: (-26, 140)

self_portrait.tif



Before displacement

After displacing G and R channels by
G: (30, 80)
R: (36, 174)

three_generations.tif

Before displacement

After displacing G and R channels by
G: (0, 60)
R: (0, 116)

tobolsk.jpg

Before displacement

After displacing G and R channels by
G: (2, 2)
R: (2, 6)

train.tif

Before displacement

After displacing G and R channels by
G: (0, 42)
R: (28, 84)

Result on extra images

The following is the result of my algorithm on some other images downloaded from the Prokudin-Gorskii collection.

lake.tif

Before displacement

After displacing G and R channels by
G: (6, -24)

R: (8, -36)

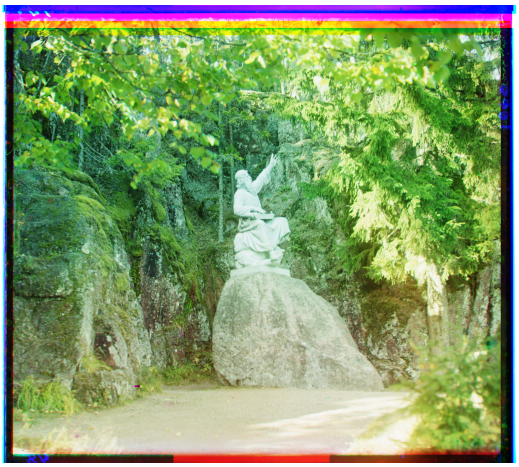
shell.tif

Before displacement

After displacing G and R channels by
G: (4, 26)
R: (8, 110)

rock.tif

Before displacement



After displacing G and R channels by
G: (2, 44)
R: (2, 164)

Bells and Whistles

Channel Normalization

I implemented channel normalization in an attempt to make similaraties in the input channel and compare channel to be more apparent. Here's an example of a channel before and after normalization.

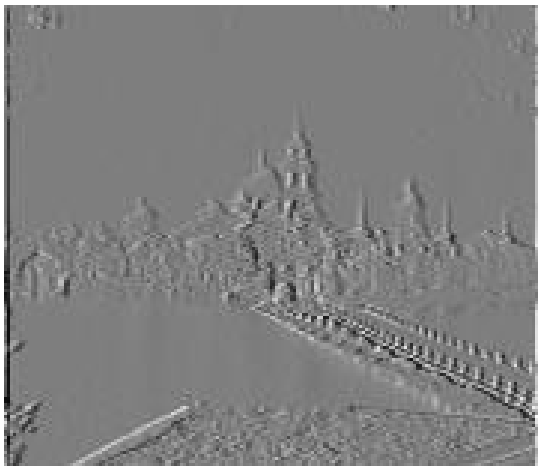

Before normalize


After normalize

Edge Detection

I implemented Edge Detection by referencing a lecture on the topic I found at https://www.cse.psu.edu/~rtc12/CSE486/lecture02.pdf. This approach involves taking two partial derivatives in the x and y direction, then using the magnitude of those partial derivative channels edge detect! I found that it worked best when I ignored the y direction. Results are as follows:


Input channel


Ix = Partial derivative wrt x


Iy = Skipping partial derivative wrt y


Magnitude of gradient (Ix^2 + Iy^2)

After normalization

Result after applying to G and R

Bells and Whistles Results

Alltogether, my extra work paid off!

Before

After

https://inst.eecs.berkeley.edu/~cs194-26/fa22/upload/files/proj1/cs194-26-ahn/