Mike Bartoli
Noah Mulfinger
CS181M Project 1
9/27/2015

The goal of this assignment is to take the digitized Prokudin-Gorsky glass plate images and, using image processing techniques, automatically produce a color image with as few visual artifacts as possible. We succeeded in our goal, and found the best alignment tool to be the normalized cross-correlation.

Our first attempt at solving the alignment problem was using a brute force approach. We didn't encounter any difficulties with this, and were successful in implementing this on our first try after whiteboarding out the solution. The comparison metric we use is the sum of squared difference, as outlined in the project description. This approach takes awhile to run on large images -- more than 15 minutes each.

Our second approach towards solving the alignment problem is based on using an image pyramid. The basic idea behind using the image pyramid is to constrain the search space of the brute force alignment. This works pretty well, we see a huge performance boost, with our large images being processed in 4.6 seconds (notice seconds instead of minutes here). We didn't run into any major problems with this approach, although it took some time to work out the algorithm on the whiteboard. We did however notice issues with the women in the chair; specifically, the blur around her face. Notice (look closely, these are jpegs!) the blue and green misaligned (see figures 3 and 4). In retrospect, our results weren't as good as our implementation of normalized cross-correlation, but at the time they looked pretty great.

We added a few bells and whistles to our project: the human touch; automatic contrasting; better alignment; and parallelization. We were successful in all but implementing parallelization. We'll walk through some of our results in each.

The human touch (2 pts), allows a user pick corresponding points in each image to help with initial alignment. We first implemented the human touch in a way that required the user to specify the aligned x,y values. This worked, however it's hard to use since you you can't just click and go. We were later informed about the built-in MATLAB function *ginput*, which allows the user to click the the point in a displayed image to give us the x,y values. We implemented this feature and found the UI to be much easier to interact

with. It works well, try it; the results are substantially better than the previous brute force alignments, which surprised us (see figures 1 and 2).

Automatic contrasting (3 pts), improves the overall contrast of the image. The heavy lifting for this bell and whistle was done by the built-in MATLAB function *imadjust*. We came to this solution pretty quickly, but didn't notice a significant difference in our output. We found that by altering the cutoff value to with the 10th percentile, we get noticeable changes (see figures 5 and 6).

Better alignment (5 pts), attempts to improve the alignment of the images by using better metrics or preprocessing the images to be aligned -- in our case, this is using the normalized cross-correlation metric. Although still nontrivial, MATLAB provides a function, *normxcorr2*, that calculates the normalized cross correlation given a template image and a smaller image. We followed a similar pattern as presented in the online documentation to a working solution in our case. Compared to the brute force alignment, using the normalized cross-correlation metric performs noticeably better in terms of alignment, and about the same in terms of speed (15.3 seconds).

Implementing parallelization (3 pts), since the scoring of each alignment can be done independently of each other, we can can a speed improvement by performing the scorings in parallel. We attempted to implement our solution using MATLAB's built-in library for parallelization. This was the hardest and most time consuming part of project 1 for our team, we probably sunk at least 3 and a half hours into to trying to debug our solution to no avail. We were focusing on switching our outer for-loops in the brute force alignment with *parafor*, however, we kept running into issues with the slicing of the arrays (which we finally overcame). We then ran into trouble with the keep tracking of min and max values, which for some reason, were always going to 1. We were unable to get a working solution for this bell and whistle, but we attached our implementation to the project for consideration.

Figures 1 and 2: Original Unaligned Image vs. Human Touch Aligned Image
Blue (-120, -6), Green (-66, 0)

Figures 3 and 4: Pyramid Align (Blue = -117, 2; Green = -62, 0) vs Normalized Cross-Correlation Align (Blue = -117, -12; Green = -63, -5

)

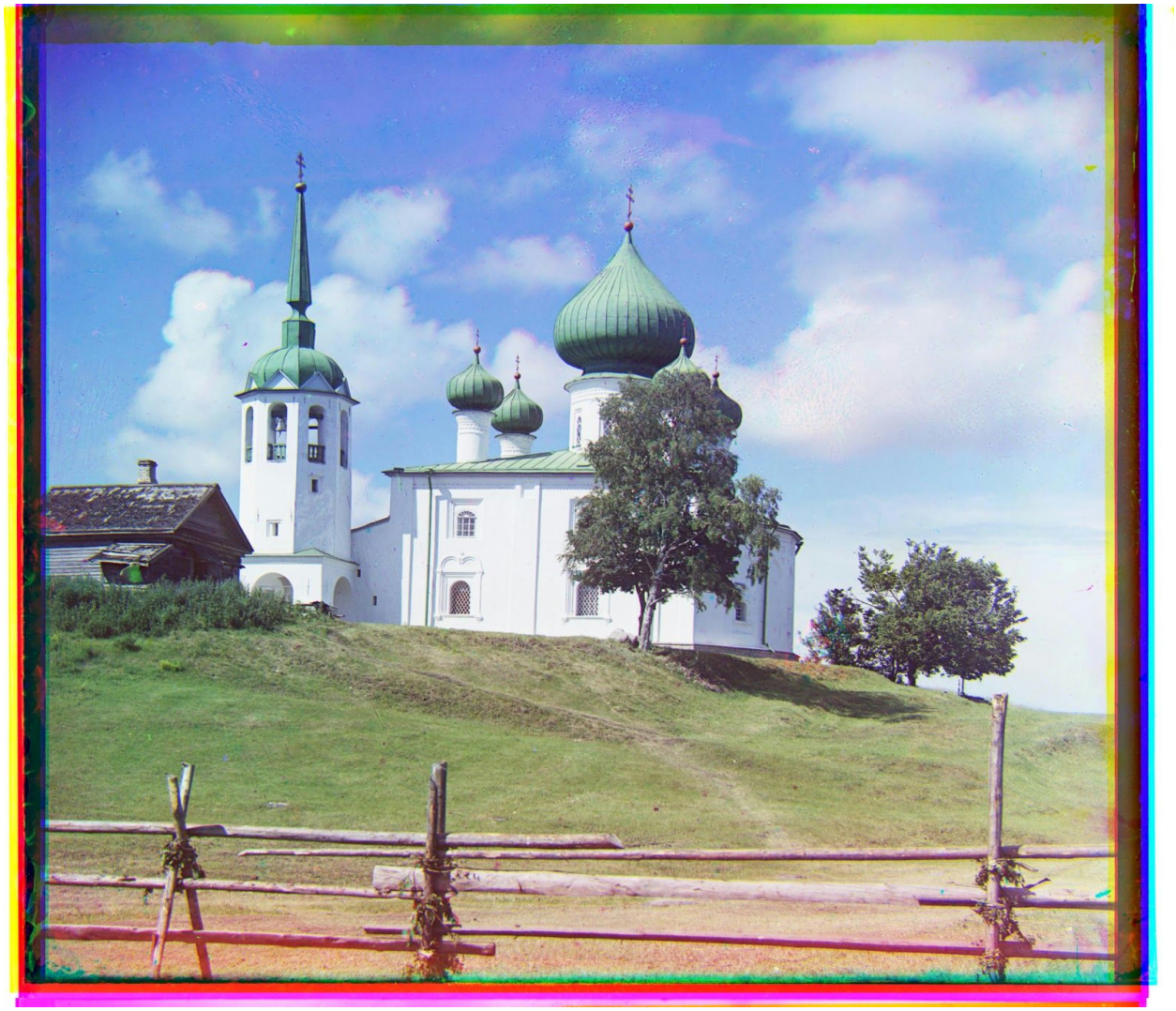
Figures 5 and 6: No Contrast vs. Contrast

Additional Images and their Shift Values Using Normalized Cross-Correlation:

Shifts: Blue=(-62,-15), Green=(-53,-6)

Shifts: Blue=(-86,-38), Green=(-50,-25)

Shifts: Blue=(-107,53), Green=(-57,31)



Shifts: Blue=(-47,-30), Green=(-30,-12)

Shifts: Blue=(-135,12), Green=(-122,5)

Shifts: Blue=(-115,26), Green=(-98,13)

Shifts: Blue=(-53,-22), Green=(-31,-8)

Shifts: Blue=(-56,-31), Green=(-35,-13)

Shifts: Blue=(-42,-16), Green=(-29,-2)

Shifts: Blue=(-143,-54), Green=(-78,-19)

Shifts: Blue=(-107,17), Green=(-59,14)