CSCE 689 - Computational Photography

Programming Assignment 1

Deadline: Feb. 8th

1 Background

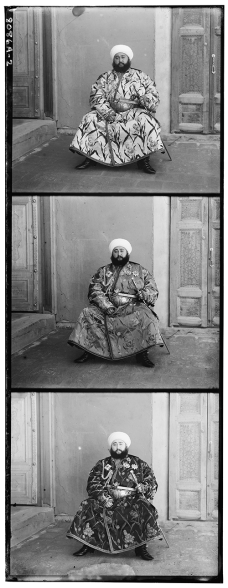
Sergei Mikhailovich Prokudin-Gorskii (1863-1944) was a man well ahead of his time. Convinced, as early as 1907, that color photography was the wave of the future, he won Tzar's special permission to travel across the vast Russian Empire and take color photographs of everything he saw including the only color portrait of Leo Tolstoy. And he really photographed everything: people, buildings, landscapes, railroads, bridges... thousands of color pictures! His idea was simple: record three exposures of every scene onto a glass plate using a red, a green, and a blue filter. Never mind that there was no way to print color photographs until much later – he envisioned special projectors to be installed in "multimedia" classrooms all across Russia where the children would be able to learn about their vast country. Alas, his plans never materialized: he left Russia in 1918, right after the revolution, never to return again. Luckily, his RGB glass plate negatives, capturing the last years of the Russian Empire, survived and were purchased in 1948 by the Library of Congress. The LoC has recently digitized the negatives and made them available on-line.

2 Overview

The goal of this assignment is to take the digitized Prokudin-Gorskii glass plate images and, using image processing techniques, automatically produce a color image with as few visual artifacts as possible. In order to do this, you will need to extract the three color channel images, place them on top of each other, and align them so that they form a single RGB color image.

We will assume that a simple (x, y) translation model is sufficient for proper alignment. However, the full-size glass plate images are very large, so your alignment procedure will need to be relatively fast and efficient. Starter code (in MATLAB) along with the images can be downloaded from here.

3 Details



The digitized glass plate images are in BGR order from top to bottom. Your program will take a glass plate image as input and produce a single color image as output. The program should divide the image into three equal parts and align the second and the third parts (G and R) to the first (B). For each image, you will need to print the (x, y) displacement vector that was used to align the parts.

The easiest way to align the parts is to exhaustively search over a window of possible displacements (say [-15,15] pixels), score each one using some image matching metric, and take the displacement with the best score. There is a number of possible metrics that one could use to score how well the images match, two of which can be find below:

- Sum of squared differences (SSD): `sum((I1-I2).^2)`
- Normalized cross-correlation (NCC): `dot(I1./||I1||, I2./||I2||)`, where `I1` and `I2` are zero-mean images.

Note that in the case of the Emir of Bukhara (show on right), the images to be matched do not actually have the same brightness values (they are different color channels), so you might have to use a cleverer metric, or different features than the raw pixels.

Exhaustive search will become prohibitively expensive if the pixel displacement is too large (which will be the case for high-resolution glass plate scans). In this case, you will need to implement a faster

search procedure such as an image pyramid. An image pyramid represents the image at multiple scales (usually scaled by a factor of 2) and the processing is done sequentially starting from the coarsest scale (smallest image) and going down the pyramid, updating your estimate as you go. Do not use MATLAB's `impyramid` function but you can use `imresize`.

Your job will be to implement an algorithm that, given a 3-channel image, produces a color image as output. Implement a simple single-scale version first, using for loops, searching over a user-specified window of displacements. You should pick one of the smaller .jpg images to test this version of the code. Next, add a coarse-to-fine pyramid speedup to handle large images like the .tiff ones.

4 Hints

- You need to implement almost everything from scratch (except the functions for reading, writing, resizing, shifting and displaying images: e.g. `imresize`, `cirshift`.). In particular, you are not allowed to use high level functions for constructing pyramids, computing image matching scores, etc.
- The average running time is expected to be less than 1 minute per image. If it takes hours for your program to finish, you should further optimize the code.
- A lot of the suggested MATLAB code will be in the Image Processing Toolbox.
- Avoid too many `for` loops by vectorizing/parallelizing your code (See more details for MATLAB).
- For all projects, don't get bogged down tweaking input parameters. Most, but not all images will line up using the same parameters. Your final results should be the product of a fixed set of parameters (if you have free parameters). Don't worry if one or two of the handout images don't align properly using the simpler metrics suggested here.
- The input images can be in jpg (uint8) or tiff format (uint16), remember to convert all the formats to the same scale (`im2double`).
- Shifting a matrix is easy to do in MATLAB by using `circshift`.
- Compute your metric on the internal pixels only, to avoid using the invalid border pixels.
- Output all of your images to jpg, it'll save you a lot of disk space.

5 Deliverables

Your entire project should be in a folder and submitted in the zip format (called "`firstname_lastname.zip`") through e-campus. Inside the folder, you should have the followings:

- A folder named "Code" containing all the codes for this assignment. Please include a README file to explain what each file does.
- A folder named "Results" containing the generated RGB images for each of the input glass plate images.
- A write up describing the algorithm used to perform the assignment. Please discuss any problem you faced when implementing the assignment or any decisions you had to make. You should also show all the results and for each report the optimum translation for the R and G channels as well as the timing. *Make sure you write your name on top of the report.*

6 Acknowlegements

This project is derived from Alexei A. Efros Computational Photography course with permission.