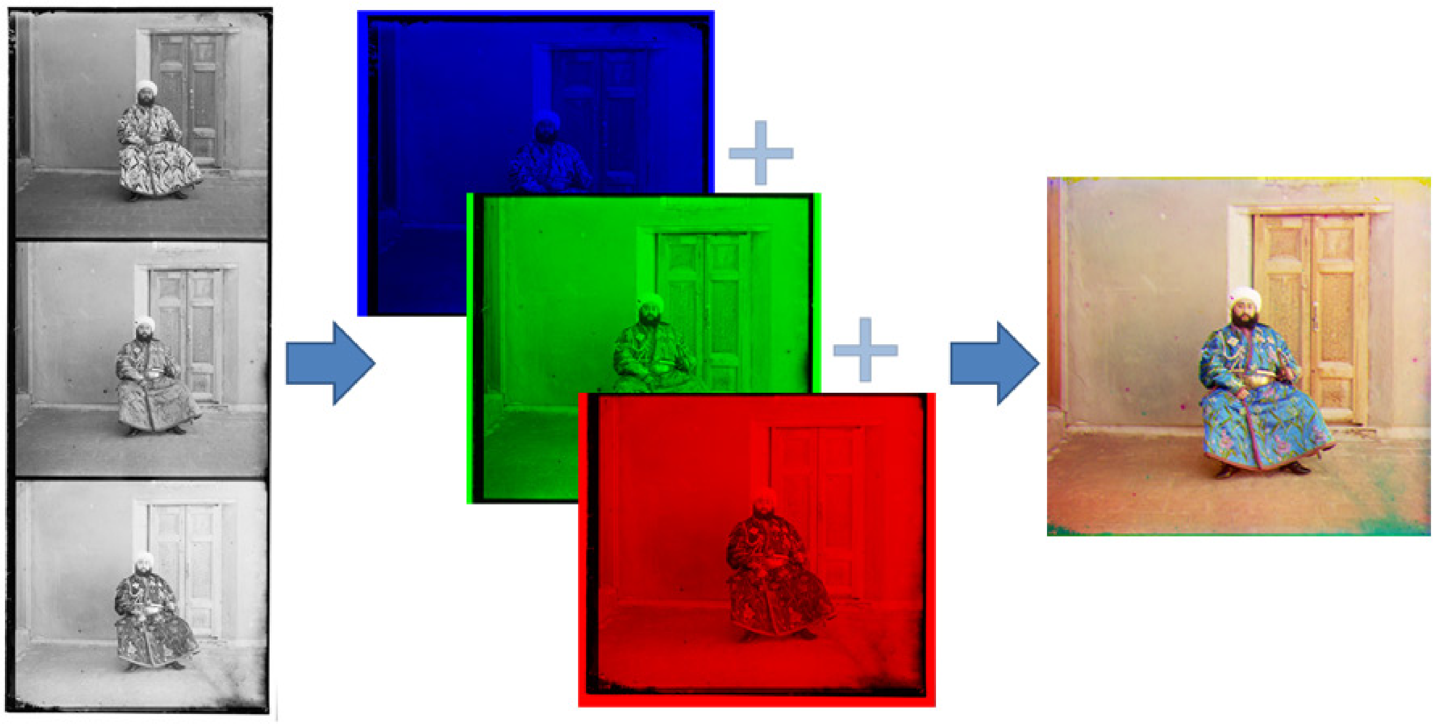


# Project 1: Image Alignment with Pyramids

## Automatic color aligning and compositing of the **Prokudin-Gorskii** photo collection

**Due Date: 11:59pm on Monday, September 17th, 2012**

## Brief

- This handout: `/course/cs129/asgn/proj1/handout/`
- Stencil code: `/course/cs129/asgn/proj1/stencil/`
- Data: `/course/cs129/asgn/proj1/data/`
- Browse through more data: **Library of Congress**
- Handin: `cs129_handin proj1`
- Required files: README, code/, html/, html/index.html

## Background

**Sergei Mikhailovich Prokudin-Gorskii** (1863-1944) was a photographer ahead of his time. He saw color photography as the wave of the future and came up with a simple idea to produce color photos: record three exposures of every scene onto a glass plate using a red, a green, and a blue filter and then project the monochrome pictures with correctly coloured light to reproduce the color image; color printing of photos was very difficult at the time. Due to the fame he got from his color photos, including the only color portrait of **Leo Tolstoy** (a famous Russian author), he won the Tzar's permission and funding to travel across the Russian Empire and document it in 'color' photographs. His RGB glass plate negatives were purchased in 1948 by the Library of Congress. They are now digitized and available **on-line**.

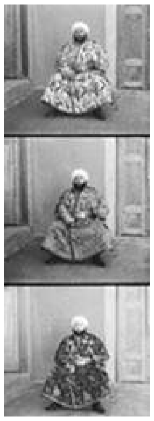
## Requirements

Take the digitized Prokudin-Gorskii glass plate images and automatically produce a color image with as few visual artifacts as possible. You will need to extract the three color channel images and align them so that they form a single RGB color image. The high-resolution images are quite large so you will need to have a fast and efficient aligning algorithm (read: Image Pyramid). You are required to implement a single-scale and multi-scale aligning algorithm that searches over a user-specified window of displacements.

Also, you are required to try your algorithm on other images from the **Prokudin-Gorskii collection**.

## Details

Important notes about the images:



- The images are, from top to bottom, in BGR order.
- Each image has a high and low res image available online, so consider trying your aligning algorithm on both.
- Assume the negatives are evenly divided into 3 plates (ie, each plate is in exactly 1/3 of the negative).
- Assume that a simple x,y translation model is sufficient for proper alignment.

MATLAB stencil code is available in `/course/cs129/asgn/proj1/stencil/`. You're free to do this project in whatever language you want, but the TAs are only offering support in MATLAB.

There are some of the digitized glass plate images (both hi-res and low-res versions) in: `/course/cs129/asgn/proj1/data/`.

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 record the displacement vector that was used to align the parts. Don't get your coordinate order mixed up -- Matlab matrices are accessed (y, x).

The easiest way to align the parts is to exhaustively search over a window of possible displacements (e.g. [-15,15] pixels), score each one using some image matching metric, and take the displacement with the best score. There are several possible metrics to measure how well images match:

- Sum of squared differences: `sum( (image1-image2).^2 )`
- Normalized cross correlation: `dot( image1./||image1||, image2./||image2|| )`

Note that in this particular case, the images to be matched do not actually have the same brightness values (they are different color channels), so other metrics might work better.

Exhaustive search will become prohibitively expensive if the displacement search range or image resolution are too large (which will be the case for high-resolution glass plate scans). To avoid this, you will need to implement a coarse-to-fine search strategy using 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 displacement estimate as you go. Do not use Matlab's `impyramid` function -- write your own function which blurs an image and then subsamples the pixels.

## Write up

For this project, and all other projects, you must do a project report in HTML. In the report you will describe your algorithm and any decisions you made to write your algorithm a particular way. Then you will show and discuss the results of your algorithm. Also discuss any extra credit you did. Feel free to add any other information you feel is relevant.

## Extra Credit

Although the color images resulting from this automatic procedure will often look strikingly real, they are still not nearly as good as the manually restored versions available on the LoC website and from other professional photographers. However, each photograph takes days of painstaking Photoshop work, adjusting the color levels, removing the blemishes, adding contrast, etc. Can you come up with ways to address these problems automatically? Feel free to come up with your own approaches or talk to the Professor or TAs about your ideas. There is no right answer here, just try out things and see what works.

Here are some ideas, but we will give credit for other clever ideas:

- Up to 4 pts: Automatic cropping. Remove white, black or other color borders. Don't just crop a predefined margin off of each side -- actually try to detect the borders or the edge between the border and the image.
- Up to 3 pts: Automatic contrasting. It is usually safe to rescale image intensities such that the darkest pixel is zero (on its darkest color channel) and the brightest pixel is 1 (on its brightest color channel). More drastic or non-linear mappings may improve perceived image quality.

- Up to 5 pts: Automatic white balance. This involves two problems -- 1) estimating the illuminant and 2) manipulating the colors to counteract the illuminant and simulate a neutral illuminant. Step 1 is difficult in general, while step 2 is simple (see the Wikipedia page on **Color Balance** and section 2.3.2 in the Szeliski book). There exist some simple algorithms for step 1, which don't necessarily work well -- assume that the average color or the brightest color is the illuminant and shift those to gray or white.
- Up to 3 pts: Better color mapping. There is no reason to assume (as we have) that the red, green, and blue lenses used by Produkin-Gorskii correspond directly to the R, G, and B channels in RGB color space. Try to find a mapping that produces more realistic colors (and perhaps makes the automatic white balancing less necessary).
- Up to 3 pts: Better features. Instead of aligning based on RGB similarity, try using gradients or edges.
- ~~Up to 5 pts~~: Better transformations. Instead of searching for the best x and y translation, additionally search over small scale changes and rotations. Adding two more dimensions to your search will slow things down, but the same course to fine progression should help alleviate this.
- Up to 4 pts: Aligning and processing data from other sources. In many domains, such as astronomy, image data is still captured one channel at a time. Often the channels don't correspond to visible light, but NASA artists stack these channels together to create false color images. For example, **here is a tutorial** on how to process Hubble Space Telescope imagery yourself. Also, consider images like **this one of a coronal mass ejection** built by combining **ultraviolet images** from the Solar Dynamics Observatory. To get full credit for this, you need to demonstrate that your algorithm found a non-trivial alignment and color correction.

For all extra credit, be sure to demonstrate on your web page cases where your extra credit has improved image quality.

## Graduate Credit

To get graduate credit on this project you need to do at least 10 points worth of extra credit. This may not be the case for all projects.

## Web-Publishing Results

All the results for each project will be put on the course website so that the students can see each other's results. In class we will have presentations of the projects and the students will vote on who got the best results. If you do not want your results published to the web, you can choose to opt out. If you want to opt out, email cs129tas[at]cs.brown.edu saying so.

## Handing in

This is very important as you will lose points if you do not follow instructions. Every time after the first that you do not follow instructions, you will lose 5 points. The folder you hand in must contain the following:

- README - text file containing anything about the project that you want to tell the TAs
- code/ - directory containing all your code for this assignment
- html/ - directory containing all your html report for this assignment (including images). Do not turn in the large .tiff images. Your web page should only display compressed images (e.g. jpg or png or gif if you want to animate something).
- html/index.html - home page for your results

Then run: `cs129_handin proj1`
If it is not in your path, you can run it directly: `/course/cs129/bin/cs129_handin proj1`

## Rubric

- +55 pts: Single-scale implementation
- +35 pts: Multi-scale implementation
- +10 pts: Write up
- +10 pts: Extra credit (up to ten points)
- -5*n pts: Lose 5 points for every time (after the first) you do not follow the instructions for the hand in format

## Final Advice

- A lot of the suggested MATLAB code will be in the Image Processing Toolbox. If you plan to do this outside of the Sun Lab machines, you will need the Toolbox.

- 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 (see `im2double` and `im2uint8`).
- Shifting a matrix is easy to do in MATLAB by using `circshift`.
- You can create the coordinates of the window you are shifting over by using `meshgrid` and turn that into a list of (x,y) pairs.
- The borders of the images will probably hurt your results, try computing your metric on the internal pixels only.
- Output all of your images to jpg, it'll save you a lot of disk space.

## Credits

Project derived from Alexei A. Efros' Computational Photography course, with permission.

## Good Luck!