Mini-project 1

CMPSCI 670, Fall 2019, UMass Amherst

Instructor: Subhransu Maji

TAs: Aruni Roy Chowdhury, Archan Ray

Guidelines

Submission. Submit a *single pdf document* via Gradescope that includes your solutions, figures and code. The latex source file for the homework is provided which you can modify to produce your report. You are welcome to use other typesetting software as long as the final output is a pdf. For readability you may attach the code printouts at the end of the solutions within the same pdf. Note that we will not run your code. Similarly figures should be included in a manner which makes it easy to compare various approaches. Poorly written or formatted reports will make it harder for the TA to evaluate it and may lead to a deduction of credit.

Late policy. You could have 24 hours late submission with a 50% mark down. Late submission beyond 24 hours will not be given *any* credits.

Plagiarism. We might reuse problem set questions from previous years, covered by papers and webpages, we expect the students not to copy, refer to, or look at the solutions in preparing their answers. We expect students to want to learn and not google for answers. See the Universities' guidelines on academic honesty (`https://www.umass.edu/honesty`).

Collaboration. The homework must be done individually, except where otherwise noted in the assignments. 'Individually' means each student must hand in their own answers, and each student must write their own code in the programming part of the assignment. It is acceptable, however, for students to collaborate in figuring out answers and helping each other solve the problems. We will be assuming that you will be taking the responsibility to make sure you personally understand the solution to any work arising from such a collaboration.

Using other programming languages. We made the starter code available in Python and Matlab. You are free to use other languages such as C, Java, Octave or Julia with the caveat that we won't be able to answer or debug language-questions.

Python requirements. We will be using Python 2.7. The Python starter code requires `scipy`, `numpy` (at least v1.12), and `scikit-image`. If you are not familiar with installing those libraries through some package manager (like `pip`), the easiest way of using them is installing Anaconda.

1 Aligning Prokudin-Gorskii images [25 points]

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.[1]

Figure 1: Example image from the Prokudin-Gorskii collection. On the left are the three images captured individually. On the right is a reconstructed color photograph. Note the colors are in B, G, R order from the top to bottom (and not R, G, B)!

Your goal is to take photographs of each plate and generate a color image by aligning them (Figure 1). The easiest way to align the plates 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. The simplest one is just the L2 norm also known as the Sum of Squared Differences (SSD) distance which is simply `sum(sum((image1-image2).^2))` where the sum is taken over the pixel values. Another is normalized cross-correlation (NCC), which is simply a dot product between two normalized vectors: `(image1./||image1||` and `image2./||image2||)`. Note that in the case of the Emir of Bukhara (Figure 1), the images to be matched do not actually have the same brightness values (they are different color channels), so the distance between them will not be zero even after alignment.

1.a Code

Before you get started make sure you know the basics of programming in MATLAB/Python. Then download the `p1.zip` and `p1.pdf` from the moodle. Move them to your homework directory and extract the files (e.g. by typing `unzip p1.zip`). The codes are available in the directory `p1/code`. The latex sources are also available as `p1/latex`.

[1] This description and assignment is courtesy Alyosha Efros @ UC Berkeley

Before you start aligning the Prokudin-Gorskii images (in `data/prokudin-gorskii`), you will test your code on synthetic images which have been randomly shifted. Your code should correctly discover the inverse of the shift. Take a look at this code and see how the test examples are generated. Run `evalToyAlignment` on the Matlab command prompt inside the code directory. This should produce the following output. Note the actual 'gt shift' might be different since it is randomly generated.

```
    Evaluating alignment ..
    1 balloon.jpeg
   gt shift: ( 1,11) ( 4, 5)
 pred shift: ( 0, 0) ( 0, 0)
    2 cat.jpg
   gt shift: (-5, 5) (-6,-2)
 pred shift: ( 0, 0) ( 0, 0)
     ...
```

The code loads a set of images, randomly shifts the color channels and provides them as input to `alignChannels`. **Your goal is to implement this function.** A correct implementation should obtain the shifts that are the negation of the ground-truth shifts producing the following output:

```
    Evaluating alignment ..
    1 balloon.jpeg
   gt shift: ( 13, 11)  ( 4, 5)
 pred shift: (-13,-11)  (-4,-5)
    2 cat.jpg
   gt shift: (-5, 5) (-6,-2)
 pred shift: ( 5,-5) ( 6, 2)
     ...
```

Once you are done with that, run `evalProkudinAlignment`. This will call your function to align images from the Prokudin-Gorskii collection stored in `data/prokudin-gorskii` directory. The output is saved to the `outDir`.

1.b What to submit?

To get full credit for this part you have to

- Include your implementation of `alignChannels`
- Verify that the `evalAlignment` correctly recovers the color image and shifts for the toy example and include the output results.
- Show the aligned color images from the output of `alignProkudin` and the computed shift vectors for each image.

1.c Some tips

- Look at functions `circshift()` and `padarray()` to deal with shifting images in MATLAB. In Python, look for `np.roll` and `np.pad`.
- The "verbatim" package provides ways to include pasted code, or code from a file, in its raw format. For example the code in a file `alignChannels` can be included in Latex by typing:

  `\verbatiminput{../solution/alignChannels.m}}.`

- Shifting images can cause ugly boundary artifacts. You might find it useful to crop the image after the alignment.

2 Photometric stereo [50 points]

In this part you will implement a basic shape-from-shading algorithm as described in Lecture 5. This is also described in the "shape from shading" section (Sec 2.2) in Forsyth and Ponce book (pdf link for this section). The input to the algorithm is a set of photographs taken with known lighting directions and the output of the algorithm is the albedo (paint), normal directions, and the height map (Figure 2).

If you haven't done so already, download the `p1.zip` and `p1.pdf` files from Moodle. The `data/photometricStereo` directory consists of 64 images each of four subjects from the Yale Face database. The light source directions are encoded in the file names.

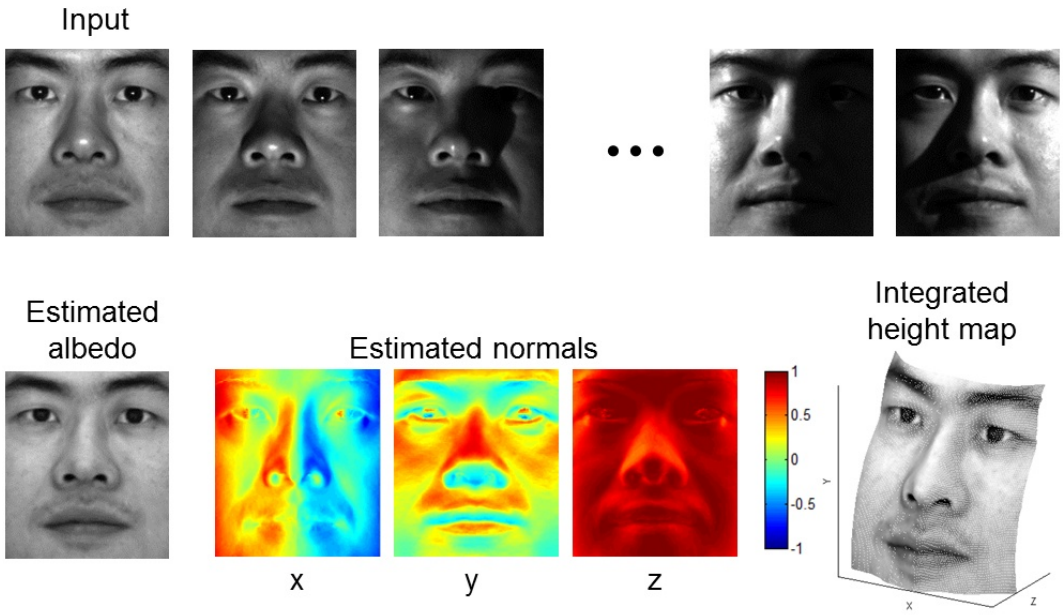


Figure 2: Top row: Input images. Bottom row: Estimated albedo, normals, and depth map

Below is an outline of the steps you need to implement the functions. The entry code for this part is `evalPhotometricStereo`. You will add code to files `prepareData`, `photometricStereo` and `getSurface` as described below. The remaining files are for processing the input data and displaying the output.

1. **Read images and light sources.** For each subject (sub-directory in the `data` folder), read in the images and light source directions. This is accomplished by the function `loadFaceImages` (which, in turn, calls `getpgmraw` to read the PGM files). The function `loadFaceImages()` returns the images for the 64 light source directions, an ambient image (i.e., image taken with all the light sources turned off), and the light directions for each image.

2. **[5 points] Preprocess the data.** We need the images under the illumination of only the point light source. Fortunately, due to linearity of light we can do this by subtracting the ambient image from all the images. Also, set any negative values to zero and rescale the resulting intensities to between 0 and 1. Implement this in the `prepareData` file.

3. **[20 points] Estimate the albedo and normals.** Implement the function `photometricStereo`, which takes as input the stack of images and the light source directions, and returns an albedo image and normal directions. The normal directions can be encoded as a three dimensional array of size $h \times w \times 3$ where the third dimension corresponds to the x-, y-, and z-components of the normal of each pixel.

   To solve for the albedo and the normals, you will need to set up a linear system of equations as described in Lecture 2. To get the least-squares solution of a linear system, use MATLAB's backslash operator. That is, the solution to `Ax = b` is given by `x = A\b`. In Python, you can use `scipy.linalg.lstsq`.

For this homework some useful MATLAB functions are `reshape`, `cat` and `bsxfun`. You may also need to use element-wise operations. For example, for two equal-sized matrices $\mathbf{X}$ and $\mathbf{Y}$, the operation $\mathbf{X}.*\mathbf{Y}$ multiplies corresponding elements, and $\mathbf{X}.\hat{}2$ squares every element. In Python, element-wise operations correspond to the standard operators and matrix multiplication is done through the `np.array` member function `dot`.

4. **[20 points] Compute the surface height map by integration.** The method is shown in Lecture 2, except that instead of continuous integration of the partial derivatives over a path, you will simply be summing their discrete values. Your code implementing the integration should go in the `getSurface` file. As stated in the slide, to get the best results, you should compute integrals over multiple paths and average the results. You should implement the following variants of integration:
   - Integrating first the rows, then the columns. That is, your path first goes along the same row as the pixel along the top, and then goes vertically down to the pixel. It is possible to implement this without nested loops using the `cumsum()` function.
   - Integrating first along the columns, then the rows.
   - Average of the first two options.
   - Average of multiple random paths. For this, it is fine to use nested loops. You should determine the number of paths experimentally.
5. Display the results using `displayOutput` and `plotSurfaceNormals` functions.

Hint: You can start by setting the `subjectName='debug'` which creates images from a toy scene with known geometry and albedo. You can debug your code against this before you try the faces.

2.a What to submit

To get full credit for this part you have to

- Include your implementation of `prepareData`, `photometricStereo`, and `getSurface`. Also, include the visualization of albedo image, estimated surface normals, and recovered surface of four subjects. It is sufficient to simply show the output of your best method. For the 3D screenshots, be sure to choose a viewpoint that makes the structure as clear as possible (and/or feel free to include screenshots from multiple viewpoints)
- **[5 points]** Discuss the differences between the different integration methods you have implemented. Specifically, you should choose one subject, display the outputs for all of a-d (be sure to choose viewpoints that make the differences especially visible), and discuss which method produces the best results and why. Also, discuss how the Yale Face data violate the assumptions of the shape-from-shading method covered in the slides.

3 Report Writing and Presentation [10 points]

Please follow the guidelines for writing a good report. Graders will penalize reports that are poorly written or fail to present the results in a reasonable manner.

4 Extensions for extra credit

You can get up to 10% extra credit for this homework by substantially improving the baseline algorithm. Some of these may even be research questions that can be explored in the final project. On this assignment, there are not too many opportunities for "easy" extra credit. This said, here are some ideas for exploration:

- Generate synthetic input data using a 3D model and a graphics renderer and run your method on this data. Do you get better results than on the face data? How close do you get to the ground truth (i.e., the true surface shape and albedo)?

- Investigate more advanced methods for shape from shading or surface reconstruction from normal fields.
- Try to detect and/or correct misalignment problems in the initial images and see if you can improve the solution.
- Using your initial solution, try to detect areas of the original images that do not meet the assumptions of the method (shadows, specularities, etc.). Then try to recompute the solution without that data and see if you can improve the quality of the solution.

If you complete any work for extra credit, be sure to clearly mark that work in your report, explain it and include the code.

5 My Solutions

1 Aligning Prokudin-Gorskii Images

1.1 The Sum of Squared Differences (SSD)

We have pictures in three kinds of colors channels: red, green and blue. To align them by using the Sum of Squared Differences (SSD) method. I used the blue channel image as the reference to shift the green channel and the red channel image. Calculated the SSD between blue green channel images and bule red channel images. I got the shifts in x y axes when the errors are minimal. By moving the images to the opposite direction of shifts, the blue channel image and green channel image can be moved to the place where the errors with the red channel images are minimal.

Here are the details I implemented:

- Defined a function in python.
- Made sure that images are in the dimensions of 1*3.
- In the range of [-15, 15], used the if loop to find the minimal SSD between the blue and green channel images, as well as blue and red channel images respectively.
- Recorded the shifts as our predicted shifts when the SSD was minimal.
- Moved the original images to the inverse direction of the predicted shifts. Built new red and green channel images by using predicted shifts.
- Returned the predicted shifts of the red and green channel images.

Table 1 shows the shift results of the alignment images in Prokudin-Gorskii by using the method above:

Table 1: Shift Results For Prokudin-Gorskii – SSD

Image	Shifts	
	G	B
00125v.jpg	(-4, 1)	(-10, 2)
00153v.jpg	(-13, -2)	(-11, -3)
00398v.jpg	(0, 1)	(-8, 2)
00149v.jpg	(-5, 0)	(-9, -1)
00351v.jpg	(-9, 0)	(-13, 1)
01112v.jpg	(-8, -1)	(-8, -3)

(a) 00125v-SSD (b) 00153v-SSD (c) 00398v-SSD

(d) 00149v-SSD (e) 00351v-SSD (f) 01112v-SSD

Figure 3: Output of The Sum of Squared Differences (SSD)

Figure 3 shows the alignment output results of the images in Prokudin-Gorskii:

I also applied this method to the pictures of Toy Example. Table 2 is the results of the image shifts:

Table 2: Shift Results For Toy Example – SSD

Image	gt shift	pred shift
balloon.jpeg	(-9, 8) (-15, 15)	(9, -8) (15, -15)
cat.jpg	(7, -13) (15, -6)	(-7, 13) (-15, 6)
ip.jpg	(-11, 8) (-1, -12)	(11, -8) (1, 12)
puppy.jpg	(6, -13) (-13, -6)	(-6, 13) (13, 6)
squirrel.jpg	(13, 14) (-11, -12)	(-13, -14) (11, 12)
pencils.jpg	(-13, 4) (-3, 13)	(13, -4) (3, -13)
house.png	(4, -14) (11, -11)	(-4, 14) (-11, 11)
light.png	(-10, 2) (-8, -13)	(10, -2) (8, 13)
sails.png	(-7, 14) (-12, -3)	(7, -14) (12, 3)
tree.jpeg	(-8, 11) (9, 0)	(8, -11) (-9, 0)

1.2 **Sobel Filters**

The purpose of this method is to find the edges of the images (where the gradient is large) by calculating the gradients. Then calculate the minimum error of the gradients between images in different color channels, to find the shifts when the error of edges is smallest. Move the images to the inverse direction of the shifts to make the images overlap as much as possible. The implementation details are:
Firstly define the sobel filters equation,

- Define the given kernel in x, y axes;

- Convolve the image with the given kernel (by calling ndimage.filters.convolve)
- Calculate the direction vector G of each edge (by calling np.hypot)
- Calculate the gradient by normalizing the direction vector.
- Calculate theta, the direction of gradients (np.arctan)

Then calculate the minimum error of the gradients among the blue and red channel images, as well as the blue and green channel images.

- The rest parts are similar to the solution in Q1), but the minimum error here is the gradient error.
- Record the shifts when getting the minimum error among gradients.
- Building new images by moving the original images to the inverse direction of the shifts.

Table 3 shows the shift results of the alignment images in Prokudin-Gorskii by using the Sobel Filters method:

Table 3: Shift Results For Prokudin-Gorskii – Sobel Filters

Image	Shifts	
	G	B
00125v.jpg	(5, -1)	(7, -2)
00153v.jpg	(13, 2)	(9, 3)
00398v.jpg	(6, -1)	(15, -2)
00149v.jpg	(5, 0)	(9, 1)
00351v.jpg	(9, 0)	(13, -1)
01112v.jpg	(5, 1)	(8, 2)

Figure 4 shows the alignment output results of the images in Prokudin-Gorskii by using Sobel Filters method.

I also applied this method to the pictures of Toy Example. Table 4 is the results of the image shifts:

Table 4: Shift Results For Toy Example – Sobel Filters

Image	gt shift	pred shift
balloon.jpeg	(12, -11) (10, 9)	(12, -11) (10, 9)
cat.jpg	(-10, 9) (12, -13)	(-10, 9) (12, -13)
ip.jpg	(-10, 14) (13, -13)	(-10, 14) (13, -13)
puppy.jpg	(-9, -6) (-13, -2)	(-9, -6) (-13, -2)
squirrel.jpg	(-7, -13) (10, -13)	(-7, -13) (10, -13)
pencils.jpg	(-5, -3) (-8, 0)	(-5, -3) (-8, 0)
house.png	(12, 1) (4, 0)	(12, 1) (4, 0)
light.png	(-11, -6) (-5, -12)	(-11, -6) (-5, -12)
sails.png	(1, -6) (-11, -3)	(1, -6) (-11, -3)
tree.jpeg	(6, -12) (-7, 9)	(6, -12) (-7, 9)

1.3 **Compare SSD and Sobel Filters**

From the results of the generated aligned images, both SSD and Sobel Filters can perform better under certain conditions. The Sobel Filters performs better in pictures which have clear edges. That is because Sobel Filter aligns the edges at first, then aligns the images. For example, in the image

(a) 00125v-sf (b) 00153v-sf (c) 00398v-sf

(d) 00149v-sf (e) 00351v-sf (f) 01112v-sf

Figure 4: Output of The Sum of Squared Differences (SSD)

"00398v.png", the edges of rails are very clear and the Sobel Filter performs better. But in the image "00153v.png", the man does not have edges as obvious as the railroad track. Therefore, the SSD performs better. We need to use different methods for specific images or requirements when aligning images.

2 Photometric Stereo

2.1 Preprocess the data

The steps that I implemented this method:

```
Defining function
  Traverse image array.
    Subtract the ambient image from all the images.
    Set all negative value to zero (np.clip).
    Scale each image to (0,1) (np.interp).
    Replace the original image with the scaled image.
  Return to the original image array.
```

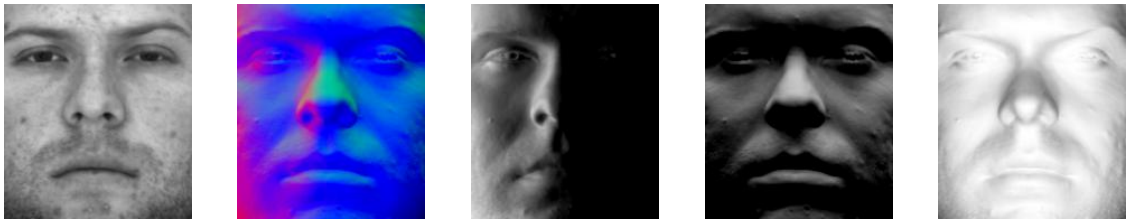
2.2 Estimate the albedo and normals.

According to the instruction of Least Squares in the lecture, I need to build the matrix of all the pixel Ij and source vectors Sj. Then calculate g by using Least Squares.

```
Defining function of photometric Stereo

   Build a new matrix for Albedo Image.
     • Assign the value of the length and width of the image array to the
     Albedo Image matrix. (Albedo Image's size is the same as the image
     array's, i.e. they have same length and width.)
```

```
    • Take 3 as the height of the Albedo Image matrix. Because the normal
    vector is a 3-dimensional vector.

  Calculate pixel matrix I.
   for x in range(imarray.shape[0]):
       for y in range(imarray.shape[1]):
           i = imarray[x, y, :]

  Calculate g by calculating least-squares solution to equation Ax = b.
    g = albedo * normals by using scipy.linalg.lstsq
    Calculate normals by calculating the length of g
      (albedoImage[x, y] = np.linalg.norm(g))
    Calculate albedo by normalizing g
      ( surfaceNormals[x, y, :] = g / np.linalg.norm(g))
return (albedoImage, surfaceNormals)
```

Figure 5678 9shows the visualization of albedo image, estimated surface normals, and recovered surface of four subjects



(a) albedo (b) normalscolor (c) normalsx (d) normalsy (e) normalsz

Figure 5: Output of the albedo and normals – yaleB01

(a) albedo (b) normalscolor (c) normalsx (d) normalsy (e) normalsz

Figure 6: Output of the albedo and normals – yaleB02

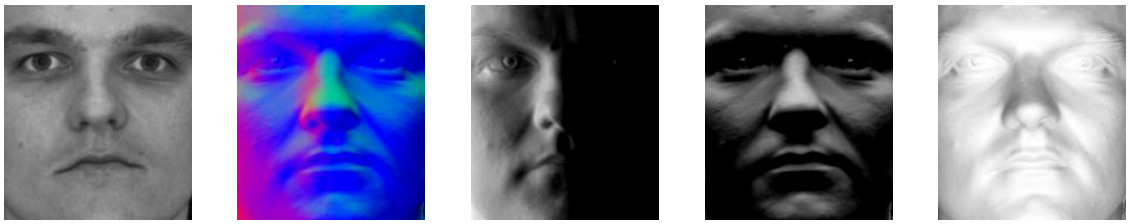
Figure 9 shows the visualization of surface of four subjects

2.3 **Compute the surface height map by integration.**

```
Defining getSurface function (surfaceNormals, method)

  Initialization height height_map
  Calculate the lateral slope fx
   fx = surfaceNormals[:, :, 0] / surfaceNormals[:, :, 2]
```

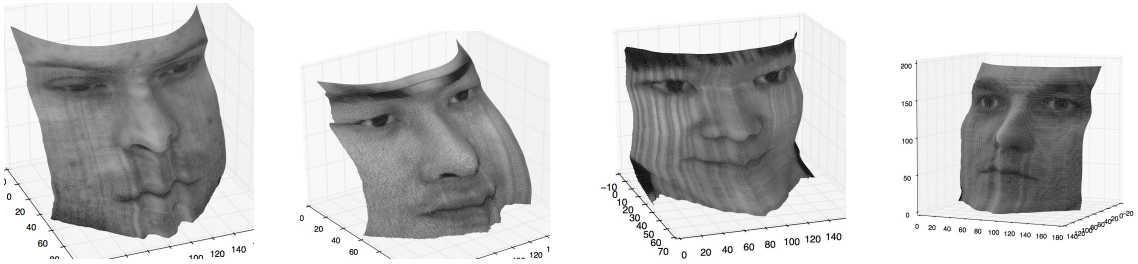
(a) albedo (b) normalscolor (c) normalsx (d) normalsy (e) normalsz

Figure 7: Output of the albedo and normals – yaleB05



(a) albedo (b) normalscolor (c) normalsx (d) normalsy (e) normalsz

Figure 8: Output of the albedo and normals – yaleB07



(a) yaleB01 (b) yaleB02 (c) yaleB05 (d) yaleB07

Figure 9: Output of the surface

```
  Calculate the longitudinal slope fy
    (fy = surfaceNormals[:, :, 1] / surfaceNormals[:, :, 2]

Then accumulate the differences of heights. Find the changes regulation
of height, and calculate the next height.
  if method == 'column':
    sum over rows for each column
      (height_map = height_map + np.cumsum(fx, axis=0)
    sum over columns for each row
      (height_map = height_map + np.cumsum(fy, axis=1))

  if method == 'row':
    sum over rows for each column
      (height_map = height_map + np.cumsum(fy, axis=0)
    sum over columns for each row
      (height_map = height_map + np.cumsum(fx, axis=1) )

   if method == 'average':
```

```
        height_map = height_map + 0.5 * np.cumsum(surfaceNormals[:, :, 0], axis=1)
        height_map = height_map + 0.5 * np.cumsum(surfaceNormals[:, :, 1], axis=0)

    return height_map
```

6 Code

1. Aligning Prokudin-Gorskii Images

1.1 The Sum of Squared Differences (SSD)

```
import numpy as np


def alignChannels(img, max_shift):
    # find the edge of each channel, shift g- and r-channel until find the least
        square error

    assert (img.shape[2] == 3)
    b_channel = img[:, :, 0]
    g_channel = img[:, :, 1]
    r_channel = img[:, :, 2]

    # find shift of g channel
    g_channel_sum_error = sum(sum(np.square(b_channel - g_channel)))
    g_channel_pred_shift = np.array([0, 0])
    for shift_i in range(-max_shift[0], max_shift[1] + 1):
        for shift_j in range(-max_shift[0], max_shift[1] + 1):
            error_sum = sum(sum(np.square(b_channel - np.roll(g_channel, [shift_i,
                shift_j], axis=[0, 1]))))
            if error_sum < g_channel_sum_error:
                g_channel_sum_error = error_sum
                g_channel_pred_shift = np.array([shift_i, shift_j])

    # find shift of r channel
    r_channel_sum_error = sum(sum(np.square(b_channel - r_channel)))
    r_channel_pred_shift = np.array([0, 0])
    for shift_i in range(-max_shift[0], max_shift[1] + 1):
        for shift_j in range(-max_shift[0], max_shift[1] + 1):
            error_sum = sum(
                sum(np.square(b_channel - np.roll(r_channel, [shift_i, shift_j],
                    axis=[0, 1]))))
            if error_sum < r_channel_sum_error:
                r_channel_sum_error = error_sum
                r_channel_pred_shift = np.array([shift_i, shift_j])

    # build img
    img[:, :, 1] = np.roll(img[:, :, 1], [g_channel_pred_shift[0],
        g_channel_pred_shift[1]], axis=[0, 1])
    img[:, :, 2] = np.roll(img[:, :, 2], [r_channel_pred_shift[0],
        r_channel_pred_shift[1]], axis=[0, 1])

    return img, np.stack((g_channel_pred_shift, r_channel_pred_shift), axis=0)
    #raise NotImplementedError("You should implement this.")
```

1.2 Sobel Filters

```
import numpy as np
from scipy import ndimage

def sobel_filters(img):
   Kx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], np.float32)
   Ky = np.array([[1, 2, 1],
               [0, 0, 0],
               [-1, -2, -1]], np.float32)

   Ix = ndimage.filters.convolve(img, Kx)
   Iy = ndimage.filters.convolve(img, Ky)

   G = np.hypot(Ix, Iy)
   G = G / G.max() * 255 # normalize the filtered img
   theta = np.arctan2(Iy, Ix)

   return (G, theta)

def alignChannels(img, max_shift):
   # find the edge of each channel, shift g- and r-channel until find the least
      square error

   assert (img.shape[2] == 3)
   b_channel = img[:, :, 0]
   g_channel = img[:, :, 1]
   r_channel = img[:, :, 2]

   b_channel_filtered = sobel_filters(b_channel)[0]
   g_channel_filtered = sobel_filters(g_channel)[0]
   r_channel_filtered = sobel_filters(r_channel)[0]

   # find shift of g channel
   g_channel_sum_error = sum(sum(np.square(b_channel_filtered -
       g_channel_filtered)))
   g_channel_pred_shift = np.array([0, 0])
   for shift_i in range(-max_shift[0], max_shift[1] + 1):
      for shift_j in range(-max_shift[0], max_shift[1] + 1):
         error_sum = sum(sum(np.square(b_channel_filtered -
             np.roll(g_channel_filtered, [shift_i, shift_j], axis=[0, 1]))))
         if error_sum < g_channel_sum_error:
            g_channel_sum_error = error_sum
            g_channel_pred_shift = np.array([-shift_i, -shift_j])

   # find shift of r channel
   r_channel_sum_error = sum(sum(np.square(b_channel_filtered -
       r_channel_filtered)))
   r_channel_pred_shift = np.array([0, 0])
   for shift_i in range(-max_shift[0], max_shift[1] + 1):
      for shift_j in range(-max_shift[0], max_shift[1] + 1):
         error_sum = sum(
            sum(np.square(b_channel_filtered - np.roll(r_channel_filtered,
                [shift_i, shift_j], axis=[0, 1]))))
         if error_sum < r_channel_sum_error:
            r_channel_sum_error = error_sum
            r_channel_pred_shift = np.array([shift_i, shift_j])

   # build img
```

```
    img[:, :, 1] = np.roll(img[:, :, 1], [g_channel_pred_shift[0],
        g_channel_pred_shift[1]], axis=[0, 1])
    img[:, :, 2] = np.roll(img[:, :, 2], [r_channel_pred_shift[0],
        r_channel_pred_shift[1]], axis=[0, 1])

    return img, np.stack((g_channel_pred_shift, r_channel_pred_shift), axis=0)
    #raise NotImplementedError("You should implement this.")
```

2. Photometric Stereo

2.1 prepareData.py

```
import numpy as np

def prepareData(imArray, ambientImage):
    for i in range(0, imArray.shape[2]):
        image_temp = imArray[:, :, i] - ambientImage

        # Set all negative value to zero
        image_temp = np.clip(image_temp, a_min=0, a_max=None)

        # Scale each image
        image_temp = np.interp(image_temp, (image_temp.min(), image_temp.max()),
            (0, +1))

        imArray[:, :, i] = image_temp
    return imArray
    #raise NotImplementedError("You should implement this.")
```

2.2 photometricStereo.py

```
import numpy as np
import scipy
def photometricStereo(imarray, lightdirs):
    assert(imarray.shape[2] == lightdirs.shape[0])

    albedoImage = np.zeros((imarray.shape[0],imarray.shape[1]))
    surfaceNormals = np.zeros((imarray.shape[0],imarray.shape[1], 3))

    for x in range(imarray.shape[0]):
        for y in range(imarray.shape[1]):
            # Compute matrix I
            i = imarray[x, y, :]

            # Calculate g, Compute least-squares solution to equation Ax = b.
            g = scipy.linalg.lstsq(lightdirs, i)[0] # first term is the solution
            albedoImage[x, y] = np.linalg.norm(g) # according to slides
            surfaceNormals[x, y, :] = g / np.linalg.norm(g) # according to slides

    return (albedoImage, surfaceNormals)
```

2.3 getSurface.py

```
import numpy as np
import matplotlib.pyplot as plt

def getSurface(surfaceNormals, method):
```

```
    height_map = np.zeros((surfaceNormals.shape[0], surfaceNormals.shape[1]))
    fx = surfaceNormals[:, :, 0] / surfaceNormals[:, :, 2]
    fy = surfaceNormals[:, :, 1] / surfaceNormals[:, :, 2]
    if method == 'column':

        height_map = height_map + np.cumsum(surfaceNormals[:, :, 1] /
            surfaceNormals[:, :, 2], axis=0) # sum over rows for each column
        height_map = height_map + np.cumsum(surfaceNormals[:, :, 0] /
            surfaceNormals[:, :, 2], axis=1) # sum over columns for each row

        #raise NotImplementedError("You should implement this.")
    if method == 'row':
        height_map = height_map + np.cumsum(surfaceNormals[:, :, 0] /
            surfaceNormals[:, :, 2], axis=1) # sum over rows for each column
        height_map = height_map + np.cumsum(surfaceNormals[:, :, 1] /
            surfaceNormals[:, :, 2], axis=0) # sum over columns for each row
        #raise NotImplementedError("You should implement this.")
    if method == 'average':
        height_map = height_map + 0.5 * np.cumsum(surfaceNormals[:, :, 0], axis=1)
            # sum over rows for each column
        height_map = height_map + 0.5 * np.cumsum(surfaceNormals[:, :, 1], axis=0)
        #raise NotImplementedError("You should implement this.")
    if method == 'random':
        raise NotImplementedError("You should implement this.")

    return height_map
```