Homework 4
Computer Vision CS 4731, Spring 2014
Due Date: April 15, 2014
Total Points: 28

This homework contains two written assignments and two programming challenges. All submissions are due at the beginning of class on **April 15, 2014**. The written assignments should be turned in as a hard copy at the beginning of class, and the challenges should be submitted according to the instructions in the document **CS4731_Guidelines_for_Programming_Assignments.pdf** before the beginning of class. Note that there is no credit for late submissions. So please start working on the homework early.

Written Assignments

Problem 1: A Lambertian surface is illuminated simultaneously by two distant point sources with equal intensity in the direction s_1 and s_2. Show that for all normals on the surface that are visible to both sources, illumination can be viewed as coming from a single "effective" direction s_3. How is s_3 related to s_1 and s_2? Now, if the two distant sources have unequal intensities I_1 and I_2, respectively, what is the direction and intensity of the "effective" source? **(4 Points)**

Problem 2: The reflectance map can be parameterized in various ways. In class we have concentrated on using the gradient *(p, q)* as a means of specifying surface orientation. In some cases, the Gaussian sphere is more suitable for this purpose. Each point on the Gaussian sphere corresponds to a particular direction, from the center of the sphere to that point. The orientation of a surface patch can be specified by giving the direction of its surface normal. Thus a given surface orientation can be identified with a particular point on the Gaussian sphere. The reflectance map is merely a means of associating brightness with orientation.

a. What are the contours of constant brightness on the Gaussian sphere in the case of a Lambertian surface illuminated by a point source? Hint: See Figure 1a. **(2 Points)**

b. Show that there are at most two surface orientations that give rise to a given pair of brightness values when the photometric stereo method is applied to a Lambertian surface. Assume that two different light sources are used. Hint: See Figure 1b. **(2 Points)**

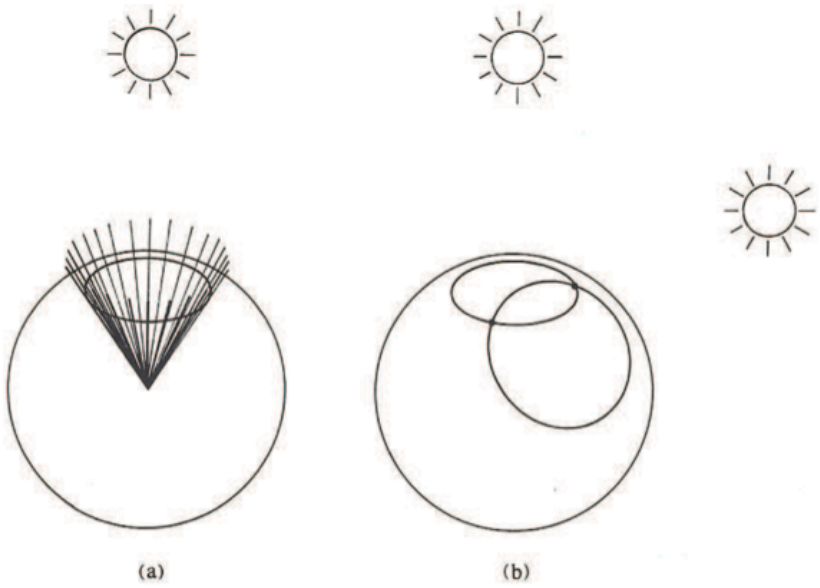

Figure 1: The reflectance map can be plotted on the Gaussian sphere. (a) The contours of constant brightness are particularly simple for a Lambertian surface illuminated by a point source. (b) This makes it easy to prove that there are at most two solutions to the two-source photometric stereo problem in this case.

Programming Assignments

This programming assignment has two challenges (each with its own subset of milestones or unit tests). Instructions and summary corresponding to these are given below. **runHw4.m** will be your main interface for executing and testing your code. Parameters for the different programs or unit tests can also be set in that file.

Before submission, make sure you can run all your programs with the command `runHw4('all')` with no errors.

MATLAB is optimized for operations involving matrices and vectors. Avoid using loops (e.g., for, while) in MATLAB whenever possible—looping can result in long running code. Instead, you should "vectorize [1]**" loops to optimize your code for performance. In many cases, vectorization also results in more compact code (fewer lines to write!). If you are new to MATLAB, refer to these articles** [1] [2] **on techniques to optimize MATLAB code.**

Challenge 1: Your task is to develop a vision system that recovers the shape,

surface normal and reflectance of an object. For this purpose you will use photometric stereo.

You will be given 5 images of an object taken using five different light sources. Your task is to compute the surface normal and albedo for the object. For this purpose, however, you will need to know the directions and intensities of the five light sources. Thus, in the first part of the assignment, you will compute the light sources directions and intensities from 5 images of a sphere and use this information in the second part to recover the orientation and reflectance.

The 11 images, **sphere0…sphere5**, and **vase1…vase5** are provided to you.

Before you begin, pay attention to the following assumptions you can make about the capture settings:

- The surfaces of all objects (including the sphere) are Lambertian. This means there are only diffuse peaks in the reflectance maps (no specular components).
- For the images, assume orthographic projections.
- Image files with the same indices are taken using the same light source. For example, **sphere1** and **vase1** are taken using light source number 1 only.
- The objects maintain the same position, orientation and scale through the different images – the only difference is the light source. For example, the sphere in **sphere0…sphere5** has the same coordinates and the same radius.
- The light sources are not in singular configuration, i.e., the S-matrix that you will compute should not be singular.
- You may **NOT** assume that the light sources are of equal intensities. This means that you need to recover not only the directions of the light sources but also their intensities.
- The background in the image is black (0 pixel value) in all images.

The task is divided into four parts, each corresponding to a program you need to write and submit.

a. First you need to find the locations of the sphere and its radius. For this purpose you will use the image **sphere0**, which is taken using many light sources (so that the entire front hemisphere is visible).

   Write a program `findSphere` that locates the sphere in an image and computes its center and radius.

```
[center, radius] = findSphere(input_img)
```
Assuming an orthographic project, the sphere projects into a circle on the image plane. Find the location of the circle by computing its centroid. Also estimate the area of the circle and from this, compute the radius of the circle. **(1 points)**

b. Now you need to compute the directions and intensities of the light sources. For this purpose you should use the images **sphere1…sphere5**.

Derive a formula to compute the normal vector to the sphere's surface at a given point, knowing the point's coordinates (in the image coordinate frame), and the center and radius of the sphere's projection onto the image plane (again, assume an orthographic projection). This formula should give you the resulting normal vector in a 3-D coordinate system, originating at the sphere's center, having its x-axis and y-axis parallel respectively to the x-axis and the y-axis of the image, and z-axis chosen such as to form an orthonormal right-hand coordinate system. Don't forget to include your formula in your README file.

Write a program `computeLightDirections` that uses this formula, along with the parameters computed in (a), to find the normal to the brightest surface spot on the sphere in each of the 5 images. Assume that this is the direction of the corresponding light source (Why is it safe to assume this? State this in the README).

Finally, for the intensity of the light source, use the magnitude (brightness) of the brightness pixel found in the corresponding image. Scale the direction vector so that its length equals this value.

```
light_dirs_5x3 = computeLightDirections(center, radius,
img_cell)
```

`Center` and `radius` are the resulted computed in (a). `img_cell` contains the 5 images of the sphere. The resulting `light_dirs_5x3` is a 5x3 matrix. Row i of `light_dirs_5x3` contains the x-, y-, and z-components of the vector computed for light source i. **(2 points)**

c. Write a program computeMask to compute a binary foreground mask for the object. A pixel in the mask has a value 1 if it belongs to the object and 0 if it belongs to the background. Distinguishing between the foreground and

background is simple: if a pixel is zero in all 5 images, then it is background. Otherwise, it is foreground.
`mask = computeMask(img_cell)`
The `img_cell` contains the 5 images of an object and mask is the binary image mask. **(1 points)**

d. Write a program `computeNormals` that, given 5 images of an object, computes the normals and albedo to that object's surface.
`[normal, albedo_img] = computeNormals(light_dirs, img_cell, mask)`
You may want to use the formula given in the class lecture notes. Be careful here! Make sure to take into account the different intensities of the light source.

Photometric stereo requires the object to be lit by at least 3 light sources. However, in our case, we have a total of 5 light sources. The lighting has been arranged in such a way that all visible surface points on an object are lit by at least 3 light sources. Therefore, while computing the surface normal at a point, choose the 3 light sources for which the point appears brightest. Be careful – choosing the wrong light sources will result in erroneous surface normal. (You may also decide to choose more than 3 light sources to compute the surface normal. This results in an over-determined linear system and can provide robust estimates. However, such a computation is not mandatory.)

Do not compute the surface normal for the background. You can assume that the surface normal in this region is looking toward the camera. Use the mask generated in the previous program to identify whether a given pixel corresponds to the object or the background.

Scale the albedo up or down to fit in the range 0…1 and show them in the output image. Thus each pixel in the output image should be the pixel's albedo scaled by a constant factor. **(6 points)**

At this point you can use the outputs of your program to reconstruct the surface of the object. `demoSurfaceReconstruction` demonstrates how to use the Frankot-Chellappa algorithm to compute the 3D shape from surface normals. You can use the MATLAB function `surf` to visualize the reconstructed surface. Surface reconstruction is provided as a demo--no submission is required.

Challenge 2: Your task is to develop an application that allows a user to refocus a scene, much similar to the two web applications shown here [3] [4]. We will call it a refocusing app.

In a conventional image, the focus of a scene is fixed at the time of capture. How can a refocusing app refocus a scene? The trick is that, instead of using a single image, a refocusing app uses a focal stack--a sequence of images captured at different focus settings--as an internal representation of a scene. When a user chooses a scene point to be "refocused", the app picks the "best focused" image from the focal stack, and displays the image. How does the app determine the "best focused" image? You have learned the answer in the Depth from Focus/Defocus lecture, and now it is your turn to apply the knowledge to refocus a real world scene.

The task is divided into two parts, each corresponding to a program you need to write and submit.

a. Write a program named `generateIndexMap` that generates an index map from a focal stack. An index map is an image with each pixel corresponding to a scene point. The integer intensity of each pixel indicates the index of the best focused layer associated with the corresponding scene point.

   Index map generation can be carried out in two steps, and the two steps need to be performed with caution to produce a quality result. The first step is to compute a focus measure for every pixel in every image in a focal stack. We will use the modified Laplacian (described in the lecture) as the focus measure. Read the formulation of the focus measure carefully and design your program to optimize for speed. A naïve implementation with unnecessary loops will result in a long running program.

   The second step is to find the layer with the maximal focus measure for each scene point. Here we will not fit a Gaussian to the computed focus measures, as finding the precise depth is not the goal. Instead, we will simply choose the layer with the maximum focus measure as the best focused layer. Be careful, the computed focus measure could be noisy, thus you may need to smooth the data (e.g., with a moving average filter) before selecting the maximal focus measure.

   Before calling `generateIndexMap`, write a program named `loadFocalStack` to load a focal stack into memory. Refer to `challenge2a` for the specifications. MATLAB has a collection of file operation functions,

which can be useful for this task [5]. **(6 Points)**

b. Write a program named `refocusApp` that executes the following tasks in a loop: 1)display an image in the focal stack; 2)ask a user to choose a scene point (with the MATLAB function `ginput`); 3)refocus to the image such that the scene point is focused. The program terminates when the user chooses a point outside of the displayed image frame. Use the index map computed in (a) to facilitate refocusing. To animate the transition effect, simply call the MATLAB function `imshow` consecutively to display a sequence of images. **(4 Points)**

References

[1] MathWorks. Vectorization. [Online]. http://www.mathworks.com/help/matlab/matlab_prog/vectorization.html

[2] MathWorks. Technique for Improving Performance. [Online]. http://www.mathworks.com/help/matlab/matlab_prog/techniques-for-improving-performance.html

[3] Lytro Image Gallery. [Online]. https://pictures.lytro.com/

[4] Columbia CAVE. Focal Sweep Photography. [Online]. http://www.focalsweep.com/

[5] MathWorks. File Operations. [Online]. http://www.mathworks.com/help/matlab/managing-files.html