Problem Set 4

Computer Vision
Spring 2022
Due Date (Thursday) 4/7/22
Total Points: 26

Prob. 1: A Lambertian surface is illuminated simultaneously by two distant point sources with equal intensity in the directions $\mathbf{s}_1$ and $\mathbf{s}_2$. 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 $\mathbf{s}_3$. How is $\mathbf{s}_3$ related to $\mathbf{s}_1$ and $\mathbf{s}_2$? 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)

Prob. 2: (22 points)

Programming Assignment
In this programming assignment you are asked to develop a vision system that recovers the orientation and reflectance of an object's surface. For this purpose you will use photometric stereo. You will be given 3 images of an object taken using three different light sources. Your task is to compute the surface normals and albedo for this object. For this purpose, however, you will need to know the directions and intensities of the 3 light sources. Thus, in the first part of the assignment, you will compute the light sources directions and intensities from 4 images of a sphere and use this information in the second part to recover the orientation and reflectance.
The 7 greyscale images, **sphere0** ... **sphere3** and **object1** ... **object3**, that can be found on the home page of the course. Your program will be tested also with additional test images we have taken.

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

- The surface of all objects (including the sphere) is Lambertian. This means there is only a diffuse peak in the reflectance map (no specular component).
- For the images, assume orthographic projections.
- Image files with the same indices are taken using the same light source. For example, **sphere1** and **object1** are taken using light source number 1 only. The image **sphere0** is taken using ambient illumination.
- 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** ... **sphere3** 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 task is divided into four parts, each corresponding to a program you need to write and submit.

Each program must accept arguments in the format specified as
program_name {*1st argument*} {*2nd argument*} ... {*Nth argument*}.

(a)**[3 pts]** First you need to find the location 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 **s1** that locates the sphere in an image and computes its center and radius. The program parameters are as follows:
s1 {*input original image*} {*input threshold value*} {*output parameters file*}.

Assuming an orthographic projection, the sphere projects into a circle on the image plane. You need to threshold the greyscale image to obtain a binary one. Make sure you choose a good threshold, so that the circle in the resulting image looks clean. Then you can find the location of the circle by computing its centroid. As for the radius, you can average the differences between the leftmost and the rightmost and the uppermost and the lowermost points of the binary circle to obtain the diameter. Then divide by two to obtain the radius. The resulting parameters file is a text file consisting of a single line containing the xcoordinate of the center, the y-coordinate of the center, and the radius of the circle, separated by a space.

(b)**[5 pts]** Now you need to compute the directions and intensities of the light sources. For this purpose you should use the images **sphere1** ... **sphere3**.
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 coordinates of the center and the radius of the sphere's projection onto the image plane (again, assume an orthonormal 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 **s2** 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 3 images. Assume that this is the direction of the corresponding light source (why is it safe to assume this?). Finally, for the intensity of the light source, use the magnitude (brightness) of the brightest pixel found in the corresponding image. Scale the direction vector so that its length equals this value.

Here are the program parameters:

s2 {*input parameters file*} {*image 1*} {*image 2*} {*image 3*} {*output directions file*}.

The input parameters file is the one computed in part (a). The image files are the 3 images of the sphere (do not hardcode their names in the program!). The resulting directions file is a plain text file that consists of 3 lines. Line *i* contains the x-, y-, and z-components (separated by a space character) of the vector computed for light source *i*.

(c)**[7 pts]** Now you are ready to compute the surface normals of the object.
Write a program **s3** that given 3 images of an object, computes the normals to that object's surface. You may want to use the formulas in the class lecture notes. Be careful here! Make sure to take into account the different intensities of the light sources.

While computing the normals, do not do this for all the pixels in the image. This would be useless and computationally expensive. Use the following strategy, instead:

- Assume that a pixel (x, y) is visible from all 3 light sources if its brightness in all 3 images is greater than a certain threshold. Do not compute for pixels that are not visible in all images. The threshold is a parameter that will be supplied as a command line argument.
- Do not compute normals for each pixel of the object. Compute for only a grid of pixels, i.e. compute the normals every N pixels along x and y axes. The value of N will be supplied as another command line argument, step.

The program's parameters are as follows:

s3 {*input directions*} {*image 1*} {*image 2*} {*image 3*} {*step*} {*threshold*} {*output*}.

Here, the input directions file is the file generated by **s2**. The 3 image files are the files of the object taken with light sources 1, 2, and 3, in this order. The step and threshold parameters were described above.

The output is image 1 with the normals shown as a "needle map". Show the gridpoints in black (use value 0). You may want to draw a small circle (of radius 1 pixel) around each gridpoint to make it easily visible.

From each grid point draw a line (a "needle") that is the projection of the normal vector at this point onto the image plane (again, use an orthographic projection). To make sure that this line is visible, scale the vector 10 times, after you normalize it (if it is normalized, the maximum projection will be 1 pixel, which is not very informative). Draw the needles in white color (255).

(d)**[7 pts]** Write a program **s4** that computes the surface albedo. The program arguments ara as follows:

s4 {*input directions*} {*image 1*} {*image 2*} {*image 3*} {*threshold*} {*output*}.

Here, the input arguments are the same as in (c), except for the step parameter, which is not needed. The output is an albedo map. Compute the albedos for all pixels visible from all 3 light sources, scale them up or down to fit in the range 0...255 and show them in the output image. Thus each pixel in the output image should be this pixel's albedo scaled by a constant factor.

As usual, you should submit a **README** file specifying what threshold values, parameters, formulas, etc. you used.