Homework 4
Computer Vision CS 4731, Fall 2011
Due Date: Nov. 15, 2011
Total Points: 40

Note 1: Both the analytical problems and the programming assignments are due at the beginning of class on Nov 15, 2011. Please start working on your assignment early and note that there is no credit for late submissions.
Note 2: Read the guidelines for the programming assignments carefully. They are available on CourseWorks at Class Files/Shared Files/Programming Guidelines.
Note 3: It is your responsibility to make sure that the submission includes all the source code and necessary files. You will be graded based on the submission received.

Problem 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)

Problem 2: The reflectance map can be parameterized in various ways. In class (and in Chapter 10 of Horn) 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.

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.

This problem is actually Exercise 10-7 from page 236 of **Robot Vision** by Horn. (4 points)

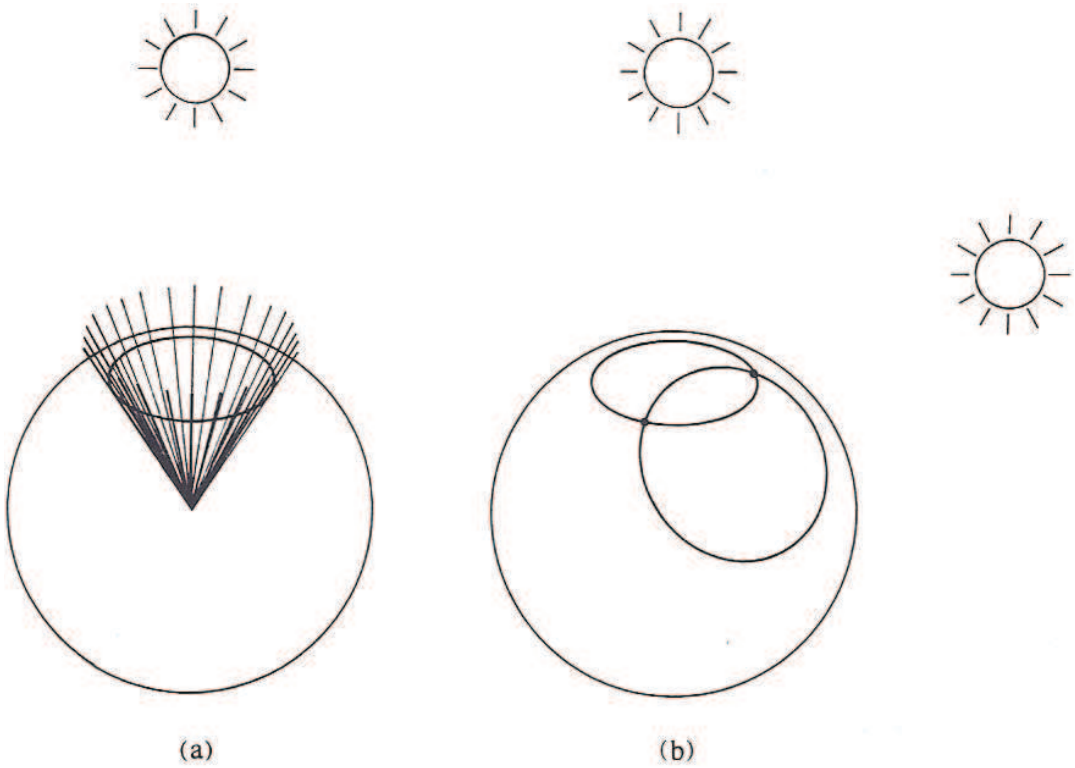

Figure 1: The reflectance map can be plotted on the Gaussian sphere. (a) The contours of constant brightness are particularly simple for a Lambertian surface illuminated by a point source. (b) This makes it easy to prove that there are at most two solutions to the two-source photometric stereo problem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

Programming Assignment

In this programming assignment you are asked 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 normals and albedo for this 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 (available on CourseWorks as hw4data.zip).

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 five 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. 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 **p1** that locates the sphere in an image and computes its center and radius. The program parameters are as follows:

   p1 {*input original image*} {*output parameters file*}

   Assuming an orthographic projection, 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.

   The resulting parameters file is a text file consisting of a single line containing the x-coordinate of the center, the y-coordinate of the center, and the radius of the circle, separated by spaces.

   (2 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 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 **p2** 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 brightest pixel found in the corresponding image. Scale the direction vector so that its length equals this value.

Here are the program parameters:

p2 {*input parameters file*} {*image 1*} {*image 2*} {*image 3*} {*image 4*} {*image 5*} {*output directions file*}

The input parameters file is the one computed in part (a). The image files are the 5 images of the sphere (do not hard code their names in the program!). The resulting directions file is a plain text file that consists of 5 lines. Line i contains the x-, y-, and z-components (separated by a space character) of the vector computed for light source i.

(4 points)

c. Write a program **p3** 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.

Here are the program parameters:

p3 {*image 1*} {*image 2*} {*image 3*} {*image 4*} {*image 5*} {*output mask*}

The image files are the 5 images of an object and the output image is the binary image mask.

(2 points)

d. Now you are ready to compute the surface normals and albedo of the object.

Write a program **p4** that, given 5 images of an object, computes the normals to that object's surface and its albedo.

You may want to use the formulae given in the class lecture notes. Be careful here! Make sure to take into account the different intensities of the light sources.

This program can be divided into three subtasks:

- The first task is to compute the surface normals. 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 normals. (You may also decide to choose more than 3 light sources to compute the surface normals. 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.

  You are now required to express the computed surface normals as a normal map image. Normal maps are images that store normals directly in the RGB values of an image. The mapping is as follows:

  X (-1.0 to +1.0) maps to Red (0-255)

  Y (-1.0 to +1.0) maps to Green (0-255)

  Z (-1.0 to +1.0) maps to Blue (0-255)

  A normal map thumbnail for a sphere is included in hw4data.zip for your reference.

  (8 points)

- The second task is to compute the surface gradients **(p,q)** at each pixel from the corresponding surface normal. The relation between surface normal and surface gradients is given in the lecture notes. A structure to store the gradients is included in the vision utilities.h file. Create a 2D array of gradient struct of the same size as the image. Store the estimated surface gradients at each pixel within this array. Finally save the entire structure array in a single binary file. You will be using this in the final part of the assignment to compute the surface depth.

  (4 points)

- The third task is to compute the albedo for all pixels corresponding to the object. Scale the albedo 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 the pixel's albedo scaled by a constant factor.

(4 points)

The program's parameters are as follows:

p4 {*input directions*} {*image 1*} {*image 2*} {*image 3*} {*image 3*} {*image 4*} {*mask*} {*normal map*} {*albedo map*} {*output gradient*}

Here, the input directions file is the file generated by **p2**. The 5 image files are the files of the object taken with light sources 1, 2, 3, 4 and 5, in this order.

Normal map is a RGB image and is the output of the first sub-task.

Albedo map is a gray scale image and contains the albedo at each pixel. (Result of third subtask)

Finally, output gradient is the name of the binary file storing the gradient values at each pixel. (Result of second subtask)

Note: The above computations may be time consuming when applied on large images. Therefore we suggest that you to first test your code for small thumbnails of the input images. Once you are confident that your program is behaving as it should, you can run the program on the full scale images.

e. Write a program **p5** that computes the surface depth from the estimated surface gradients.

To compute the depth, you can use the path integration algorithm discussed in the class. However, there is one caveat. The reference or the seed point from where we start the integration of gradients should lie on the object and not on the background. Therefore, we cannot start integrating from one of the corners of the image. Accordingly, we need to change the integration scheme. You are free to choose any scheme for the integration path progression.

Figure 2 presents one of the possible integration schemes. The idea is to divide the image into four quadrants with the seed point as the center and then integrate each quadrant separately.

Once you have computed the depth for all pixels corresponding to the object, scale the depth to occupy the range [0,1]. You don't have to compute the depth for the background. Now repeat the computation of depth for different seed points. Finally, compute the average of all the depths to reduce the error due to noise.

Scale the average depth map so as to occupy the range [0,255] and save it as the output image.

The program arguments are as follows:

p5 {*input gradient file*} {*seed point file*} {*binary mask*} {*output depth map*}

The input gradient file is the name of the file in which you saved the surface gradients. It is up to you to load this file properly.

The second argument (seed point file) is the name of a simple text file containing a list of seed points. Each line in this file has the x and y coordinates of a seed point. Your program should compute the depth starting at each seed point. Include the seed files in your submission.

The binary mask is the image got from p3 in order to distinguish between the object and the background.

Scale the average depth to fit [0,255] and save it in the output image.

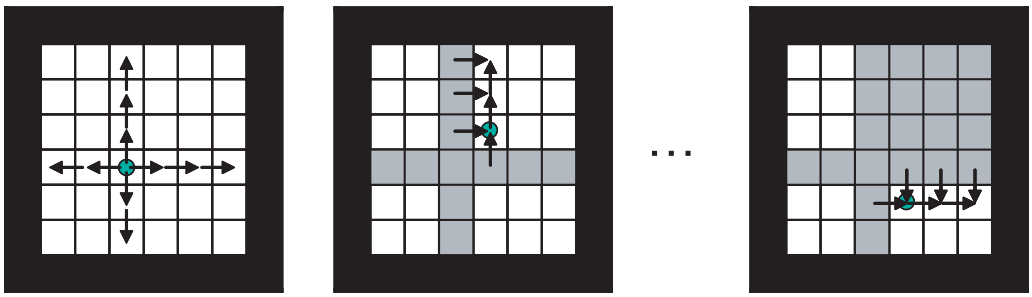

Figure 2: Integration scheme proposed for program p5.

(8 points)

Notes:

- The data for the programming assignment can be downloaded from CourseWorks as hw4data.zip. This zip file contains the test images and a Makefile which contains targets for building the programs (you will need to fill some values, such as the names of your files, thresholds, etc.), targets for testing it out ('make test' to test all programs, 'make test1' to test only program p1, 'make test2' to test only program p2, etc.), and a target for submitting ('make submit').
- You should also download the vision utilities, available from CourseWorks. This contains useful functions for reading and writing images, accessing image data, and drawing lines. Be sure to copy the files **vision_utilities.c** and **vision_utilities.h** to your working directory so that you can compile and use them in your program.