CSC2503—Foundations of Computer Vision, Fall 2016

Assignment 1: Photometric Stereo

Due: 2:00pm, Wednesday, October 12

The purpose of this assignment is to construct a depth field from a series of images of an object from the same view point, but under different illumination conditions. We assume orthographic projection of an object with Lambertian reflectance, illuminated by several distant point light sources. This problem is widely known as *Photometric Stereo*.

Your goal is to 1) recover the light source directions, 2) estimate the surface normals at each pixel, 3) estimate the albedo of the object at each pixel, and 4) estimate the depth (or height) at each pixel. You will be given a collection of test images.[1] For each of six objects you will receive 12 images. For a single object all the images are taken from the same camera position and viewing direction. They are, however, taken under different point light sources; this is the key. In addition to the 12 images there is a mask image for each object that specifies those pixels to which the object projects in the images. Finally, in addition to the test objects there are images taken of a chrome sphere (from which you will estimate eight of the light source directions).

Get the handout code A1handout.zip from the course dropbox. A template for the solution is `solntemplate.m`. This file and others in the zip file are used in the following questions.

What to hand in. Write a short report in PDF format addressing each of the questions below (LaTeX- or Word-formatted reports are preferred but hand-written reports that have been scanned to PDF along with a small sample of result images is acceptable). The report should be self-contained in that the marker should not have to run your Matlab code to be convinced your code is correct. In your report you can assume that the marker knows the context of the questions, so do not spend time repeating material in the hand-out or in class notes. Pack your PDF report and the completed Matlab files into a zip file called `A1.zip` and submit it via the `submit` command on a CS Teaching Lab computer. *You need to have a Teaching Lab account to do this so make sure you get your account set up well before the due date.* Accounts for all registered students have already been set up (the username is your UTORid).

1. **Calibration.** Before we can extract surface normals from images, we have to calibrate our capture setup. We need to specify the transformation from the scene to image coordinates. We also need to determine the power and direction of the various illuminants, as well as the camera response function.

   For this assignment, the camera response function has already been radiometrically calibrated. In particular, the gamma correction has been undone, and any vignetting has been corrected for. What this means is that you can treat pixel values as (proportional to) scene radiance. So for a Lambertian surface we have the image brightness $I_\nu(\vec{x})$, at a pixel $\vec{x}$ in each of the colour channels, $\nu = R, G, B$, is given by

   $$I_\nu(\vec{x}) \;=\; a_\nu(\vec{x}) \lfloor \vec{n}(\vec{x}) \cdot \vec{L} \rfloor. \tag{1}$$

   Here $a_\nu(\vec{x})$ for $\nu = R, G, B$ is the albedo of the surface at pixel $\vec{x}$ (within the range $[0, 255]$), and $\vec{n}(\vec{x})$ is the surface normal. Also $\vec{L}$ represents both the direction and intensity of the light source.

[1] Many thanks to Steve Seitz at University of Washington for providing the calibrated images.

We assume the strength of the different light sources has been balanced, and take $||\vec{L}|| = 1$.

The only remaining task in calibratng the capture setup is therefore to estimate the light source directions. The method we suggest is based on images of a shiny chrome sphere, in the same location as all the other objects, illuminated with the same point sources as the other objects. Since we know the object is spherical, we can determine the normal at any given point on its surface, and therefore we can also compute the reflection direction for any spot on the surface.

Toward this end we will assume a particularly simple projection model. First, for simplicity, let's assume that the world coordinate frame and the camera coordinate frame are coincident, so that the extrinsic mapping is the identity. And let's assume a right-handed coordinate system (Y points down, X points right, and the optical axis coincides with positive Z-axis). Finally let's assume an orthographic projection. Accordingly, the direction of the camera from any point on the objects of interest in the scene is $(0, 0, -1)$.

With these assumptions we can specify the mapping from points in the world, $\vec{X} = (X, Y, Z)^T$, to the locations of image pixels, $\vec{x} = (x, y)$, comprising the intrinsic and extrinsic transformations, as follows:

$$\vec{x} \;=\; \begin{bmatrix} s_1 & 0 & o_1 \\ 0 & s_2 & o_2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} \vec{X} \\ 1 \end{pmatrix} \tag{2}$$

where s_1 and s_2 are determined by the pixel dimensions, and $(o_1, o_2)^T$ is a simple coordinate shift of the origin (which we can take to be zero). Here we can use square pixels of unit size, so $s_1 = s_2 = 1$. With these choices the orthographic projection reduces to $\vec{x} = (x, y)^T = (X, Y)^T$.

Your Task. Given this setup, complete the M-function `fitChromeSphere.m` so that it estimates the light source directions from the eight images of the chrome sphere. (These are the same light source directions for the first 8 images for each different object.) In your Matlab code avoid looping over image pixels. You can test your code by running the script file `solntemplate.m` (up to the first `continue` statement). This script calls `getLightDir.m`, which in turn calls `fitChromeSphere.m`. In your report briefly describe the algorithm you used here and your reasons for choosing it. (If you have trouble with this question, default light source directions are provided in `getLightDir.m`. These are incorrect, but will let you begin doing the other parts of this assignment.)

2. **Computing Surface Normals and Grey Albedo.** Let's begin with a grey-level image I (perhaps just the average of the three colour channels, R, G, and B). Then, given the model assumptions made above of diffuse objects and distant point light sources, the relationship between the surface normal $\vec{n}(\vec{x})$ and albedo $a(\vec{x})$ at some scene point $\vec{p}$ which projects to image pixel $\vec{x}$ is given by

$$I(\vec{x}) \;=\; a(\vec{x})\; \vec{n}(\vec{x}) \cdot \vec{L}. \tag{3}$$

This is a monochrome version of equation (1), with $I(\vec{x})$ denoting the image brightness at pixel $\vec{x}$. Also, for convenience, we have dropped the rectification of the $\vec{n} \cdot \vec{L}$ term. Essentially we are assuming that the light source is positioned such that no visible piece of the surface is oriented away from the light source. This assumption may not hold exactly, but it will simplify our estimation algorithms. Since our images are already balanced as described above, we assume the

incoming irradiance from each light is 1, that is $||\vec{L}|| = 1$. Finally, it is also assumed that $\vec{n}(\vec{x})$ is a unit vector.

Given the lighting directions computed in question 1, equation (3) provides one constraint on three unknowns, namely, the albedo $a(\vec{x})$ and two independent components of the direction of $\vec{n}(\vec{x})$. With three images, taken under linearly independent lighting directions, we should be able to uniquely constrain $a(\vec{x})$ and $\vec{n}(\vec{x})$. With more than three images we may not be able to find an exact solution, but we can formulate a least-squares estimator for the unknowns $a(\vec{x})$ and $\vec{n}(\vec{x})$.

Initially, let the unknowns at each pixel be written as a single 3-vector, for example, let $\vec{g}(\vec{x}) = a(\vec{x})\vec{n}(\vec{x})$. We therefore wish to estimate the vector $\vec{g}(\vec{x})$ at pixel $\vec{x}$. Toward this end, our objective function for pixel $\vec{x}$, given greyscale measurements $I_j(\vec{x})$ in image j, is

$$E(\vec{x}) \;=\; \sum_{j=1}^{8} (I_j(\vec{x}) - \vec{g}(\vec{x}) \cdot \vec{L}_j)^2 \tag{4}$$

Here j indexes just the first eight images, that is, the ones for which we obtained estimates of the light source directions $\vec{L}_j$ in question 1 above. The objective $E(\vec{x})$ is then minimized with respect to $\vec{g}(\vec{x})$.

Your Task. Derive the solution by differentiating $E(\vec{x})$ with repect to the elements of $\vec{g}$ (for a fixed pixel $\vec{x}$) and then setting the derivatives to zero. Show your work here. Finally, given $\vec{g}(\vec{x})$, it follows that the grey-level albedo $a(\vec{x})$ is just the magnitude of $\vec{g}(\vec{x})$ and $\vec{n}(\vec{x})$ is of course the direction of $\vec{g}(\vec{x})$. Implement your solution in the M-file `fitReflectance.m`, which has been started in the handout code. In your Matlab code, avoid looping over image pixels. When you complete this M-file, run the script file `solntemplate.m` up to the `continue` statement to look at your result. The script file will display figures of your estimated grey albedo, the components of the recovered normals, and the image reconstructions along with their errors. Comment on the quality of the (grey-level) image reconstructions. What sort of modelling errors are most apparent?

3. **Computing RGB Albedo and Relighting.** Above we computed the normal and albedo for a monochrome (or grayscale) image. We can now use that normal to compute the albedo for all three colour channels. You can formulate this as a least-squares estimator for the albedo $a_\nu(\vec{x})$ for pixel $\vec{x}$ and color channel ν, minimizing

$$E_\nu(\vec{x}) \;=\; \sum_{j=1}^{8} (\, I_\nu(\vec{x}) - a_\nu(\vec{x})\, \vec{n}(\vec{x}) \cdot \vec{L}_j \,)^2 \tag{5}$$

with ν ranging over R, G, and B, where all quantities are known except of course for $a_\nu(\vec{x})$.

Your Task. To minimize equation (5), differentiate $E_\nu(\vec{x})$ with respect to $a_\nu(\vec{x})$, set the derivative to zero and solve. Show your work here. Complete the corresponding section of the script file `solntemplate.m`. When this is completed the code should display your estimated albedo (now in colour, check this on the cat, owl, or rock image sets), and will synthesize new images from novel light source positions. Comment on the quality of the newly synthesized images. (E.g., do they appear realistic?) Supposing that the recovered albedos and surface normals are correct, what are some of the main remaining modelling errors in generating these synthetically light images?

4. **Surface Fitting.** Finally we want to find a surface which has (roughly) these normals. Due to noise in the estimated normals, we cannot simply integrate the surface normals along particular paths to obtain the surface. The trouble is that, due to noise, the value we get will typically be path dependent. Therefore, rather than attempting to compute a unique surface that is exactly consistent with all the normals, we will use a simple least-squares estimator that is approximately consistent with the normals.

   To formulate the problem, let's first express the surface as $\vec{S}(X,Y) = (X, Y, Z(X,Y))^T$, in world coordinates. We can then write two surface tangent vectors as

$$\vec{\tau}_X = \frac{\partial \vec{S}}{\partial X} = (1,\, 0,\, Z_X)^T \tag{6}$$

$$\vec{\tau}_Y = \frac{\partial \vec{S}}{\partial Y} = (0,\, 1,\, Z_Y)^T \tag{7}$$

   where $Z_X = \frac{\partial Z}{\partial X}$ and $Z_Y = \frac{\partial Z}{\partial Y}$. And therefore the surface normal, a unit vector in the direction of the cross product $\vec{\tau}_X \times \vec{\tau}_Y$, is given by

$$\vec{n}(X,Y) = \frac{(Z_X, Z_Y, -1)^T}{||(Z_X, Z_Y, -1)||} \tag{8}$$

   We already estimated this normal $\vec{n}(\vec{x})$ at every pixel in question 2 above.

   To formulate our constraints on depth, we exploit equation (2), and write the depth derivatives, $\frac{\partial Z}{\partial (X,Y)} = (Z_X, Z_Y)$, as

$$\frac{\partial Z}{\partial (X,Y)} = \frac{\partial Z}{\partial (x,y)} \frac{\partial (x,y)}{\partial (X,Y)} = \frac{\partial Z}{\partial (x,y)} \begin{bmatrix} s_1 & 0 \\ 0 & s_2 \end{bmatrix} = \left(s_1 \frac{\partial Z}{\partial x},\, s_2 \frac{\partial Z}{\partial y} \right) \tag{9}$$

   Here we denote the pixel $\vec{x}$ as the 2-vector $(x,y)^T$. Next, we use first-order forward differences to approximate the depth derivatives, yielding

$$Z_X \approx s_1 \left(Z(x+1, y) - Z(x,y) \right) \tag{10}$$

$$Z_Y \approx s_2 \left(Z(x, y+1) - Z(x,y) \right) \tag{11}$$

   We then substitute these two approximations into equations (6) and (7) to obtain approximate tangent vectors. Ignoring image noise and errors in the numerical differentiation, we know that these tangent vectors are perpendicular to the surface normal; i.e.,

$$\vec{\tau}_X(x,y) \cdot \vec{n}(x,y) = 0 \tag{12}$$

$$\vec{\tau}_Y(x,y) \cdot \vec{n}(x,y) = 0 \tag{13}$$

   These equations constrain the unknown depths Z using the known normal vectors. However, given any solution $Z(x,y)$ it follows that $Z(x,y) + c$ is also a solution for any constant c. To constrain this remaining degree of freedom, set $Z(x_0, y_0) = 0$ at one pixel (x_0, y_0).

   With some algebraic manipulation, one can rewrite the collection of constraints for pixels within the object mask as

$$A\,\vec{z} = \vec{v}\,, \tag{14}$$

where $\vec{z}$ is a vector comprising the values in the *depth map* $Z(x, y)$, the entries of the matrix A include the third component of the normal at each pixel along diagonals and some off-diagonals, and the vector $\vec{v}$ comprises the first and second elements of the normal vector at each pixel. The matrix equation has NM columns and just less than $2NM$ rows. It's very large. Fortunately, most of the entries are zero, and so you can exploit the use of sparse matrix representations and solvers in Matlab. Given A and $\vec{v}$ you can use the backslash Matlab operation $A\backslash\vec{v}$ to find the pseudo-inverse solution for $\vec{z}$. As this is a large sparse system Matlab will use an iterative conjugate gradient solver.

Your Task. Complete the M-file `getDepthFromNormals.m` so that it uses the above algorithm to estimate the depths $Z(\vec{x})$. The handout code should then display your estimate. Describe, in no more than a few paragraphs, your assessment of where the technique works well, and where there are failures. Where the technique fails to produce nice results, please explain as best you can what are the likely causes of the problems. Can you formulate ways to fix these problems?

5. **BONUS [for 2 extra marks]: Estimate Light Source Direction Given Scene Properties.** The image data set for each object includes four additional images taken from the same viewpoint as before, but with unknown light source directions $\vec{L}$. These are the last four images in each data set, and for these we do not have corresponding images of the chrome sphere.

   Your Task. Formulate a simple minimization problem to estimate both the direction and magnitude of $\vec{L}$ for each of these four additional images, given your results for $a(\vec{x})$ and $\vec{n}(\vec{x})$ from question 2 above. Your write up should clearly describe this minimization problem, and the algorithm you used to solve it. Implement the algorithm in the appropriate part of `solntemplate.m`. Your recovered light sources should be stored as columns in the 3×4 array `Lrec`.

   In addition, suppose one of the novel images was actually illuminated by two separate, distant, point light sources of reduced intensity. If you ignore self-shadowing and the overall brightness, could you estimate both light source directions from such an image? If so, explain how. If not, what could you estimate about the two light source directions? Explain. (This is a theoretical question, you do not have to program anything for this last part. Although, if you are curious, you can try this by computing the average of any two images for a particular data set. This will provide a very good approximation of the image of the object under both illuminants, each with reduced intensity.)