# Assignment 1

*Computer Vision 2015*
*University of Bern*

## Photometric Stereo (Due on 27/10/2015)

The purpose of this assignment(credit to Prof. Fleet[1]) is to construct a depth map 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 (courtesy of Dan Goldman and Steven Seitz[2]). For each of the 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 illumination conditions. In addition to the 12 images there is a mask image for each object that specifies which pixels should be used (value 1) and which should be discarded (value 0). Finally, in addition to the test objects there are images of a chrome sphere. With these you can estimate the light source directions. Download the handout code *A1handout.zip* from ILIAS. The main script for the solution is *solntemplate.m*. This file and others in the *zip* file are used in the following questions:

**1. Calibration (35 points)** Before we can extract surface normals from images, we have to calibrate our acquisition setup. We need to specify the transforma-

[1] `http://www.cs.toronto.edu/~fleet/courses/2503/fall11/Handouts/a1.pdf`
[2] `http://www.cs.washington.edu/education/courses/csep576/05wi\newline/projects/project3/project3.htm`

tion 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 $I_\nu(\mathbf{x})$, at a pixel $\mathbf{x}$ in each of the colour channels, $\nu = R, G, B$, is given by

$$I_\nu(\mathbf{x}) = a_\nu(\mathbf{x})(\mathbf{n} \cdot \mathbf{L}) \tag{1}$$

Here $a_\nu(\mathbf{x})$ for $\nu = R, G, B$ is the albedo of the surface at pixel $\mathbf{x}$ (within the range $[0, 255]$), and $\mathbf{n}(\mathbf{x})$ is the surface normal. Also $\mathbf{L}$ represents both the direction and intensity of the light source. The only remaining task in calibrating 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, $\mathbf{X} = (X, Y, Z)^T$ , to the locations of image pixels, $\mathbf{x} = (x, y)$, which reduces to $\mathbf{x} = (x, y)^T = (X, Y)^T$.

**Your Task:** Given this setup, complete the M-function $fitChromeSphere.m$ so that it estimates the light source directions from the images of the chrome sphere. In your Matlab code avoid looping over image pixels. You can test your code by running the script file *solntemplate.m* which 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.)

**Matlab scripts to use:** *solntemplate.m*, *getLightDir.m*, $fitChromeSphere.m$

**2. Computing Surface Normals and Grey Albedo (30 points)** 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 $\mathbf{n}(\mathbf{x})$ and albedo $a(\mathbf{x})$ at some scene point $\mathbf{p}$ which projects to image pixel $\mathbf{x}$ is given by

$$I(\mathbf{x}) = a(\mathbf{x})(\mathbf{n} \cdot \mathbf{L}) \tag{2}$$

This is a monochrome version of equation (1), with $I(\mathbf{x})$ denoting the image brightness at pixel $\mathbf{x}$.
**Your Task:** Given the lighting directions computed in question 1, estimate (via least-squares) the unknowns $a(\mathbf{x})$ and $\mathbf{n}(\mathbf{x})$. 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. After you computed the normal and albedo for a monochrome (or grayscale) image, you can now use that normal to compute the albedo for all three colour channels.
**Your Task:** Estimate the albedo for each colour channel.
**Matlab scripts to use:** *solntemplate.m*, *fitReflectance.m*

**3. Surface Fitting (35 points)** 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 $\mathbf{S}(X, Y) = (X, Y, Z(X, Y))^T$ , in world coordinates. We can then write two surface tangent vectors as

$$\mathbf{t}_X = \frac{\partial \mathbf{S}}{\partial X} = (1, 0, Z_X)^T \tag{3}$$

$$\mathbf{t}_Y = \frac{\partial \mathbf{S}}{\partial Y} = (0, 1, Z_Y)^T \tag{4}$$

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 $\mathbf{t}_X \times \mathbf{t}_Y$ , is given by

$$\mathbf{n}(X, Y) = \frac{(Z_X, Z_Y, -1)^T}{||(Z_X, Z_Y, -1)||} \tag{5}$$

We already estimated these normals at every pixel in question 2 above. To formulate our constraints on depth, we write the depth derivatives, $\frac{\partial Z}{\partial (X,Y)} = (Z_X, Z_Y)$ as $\frac{\partial Z}{\partial (X,Y)} = \frac{\partial Z}{\partial (x,y)}$ where we use first-order forward differences to approximate the depth derivatives, yielding

$$Z_X \approx Z(x+1, y) - Z(x, y) \tag{6}$$

$$Z_Y \approx Z(x, y+1) - Z(x, y) \tag{7}$$

We then substitute these two approximations into equations (3) and (4) 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.,

$$\mathbf{t}_X(x,y)\cdot\mathbf{n}(x,y)=0 \tag{8}$$

$$\mathbf{t}_Y(x,y)\cdot\mathbf{n}(x,y)=0 \tag{9}$$

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

$$A\mathbf{z}=\mathbf{v} \tag{10}$$

where $z$ 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 $v$ 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 $\mathbf{v}$ you can use the backslash Matlab operation $A\backslash\mathbf{v}$ to find the pseudo-inverse solution for $\mathbf{z}$. 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 $\mathbf{Z}(\mathbf{x})$. Describe, in no more than a few paragraphs, your assessment of when the technique works well, and when there are failures. When the technique fails to produce nice results, please explain as best as you can what the likely causes of the problems are.
**Matlab scripts to use:** *solntemplate.m*, *getDepthFromNormals.m*

**What to hand in**:

1. Your personally written and commented **Matlab code** (zip file). The code should run without errors or warnings and should not crash, otherwise there will be a penalty at the final assignment mark. The mark does not depend on performance, however if the code takes too long to run (e.g. because of many *for* loops) a penalty will apply again.

2. A **PDF report** (see template for details).

For any questions feel free to post on the ILIAS Forum, contact the Teaching Assistants (Attila Szabó or Qiyang Hu) in class or fix an appointment at szabo@inf.unibe.ch or hu@inf.unibe.ch.