Homework assignment 5

15-463, 15-663, 15-862 Computational Photography, Fall 2023
Carnegie Mellon University

Due Monday, Nov. 20, at 11:59pm ET

The purpose of this assignment is to explore photometric stereo as a computational imaging technique for recovering the shape of an object. As we discussed in class, photometric stereo, in its most common form, takes as input images of an object under a fixed orthographic camera and multiple illumination directions. By assuming that the object is Lambertian and the illumination directional, per-pixel albedoes, normals, and eventually depth can be recovered using simple linear algebraic operations.

In the first part of the homework, you will use data captured by us to implement two types of photometric stereo: uncalibrated, where we do not know the lighting directions, and calibrated, where we know the lighting directions. By comparing the two, you will learn about the generalized bas-relief ambiguity [2] that exists in the uncalibrated case. In the second part, you will use your own uncalibrated photometric stereo to measure the shape of some objects of your choice (up to a generalized bas-relief ambiguity), by capturing images with your own camera. There are also two bonus parts, where you can implement two popular algorithms for resolving the bas-relief ambiguity.

We strongly encourage you to read the papers by Belhumeur et al. [2] and Yuille and Snow [5], which discuss the generalized bas-relief ambiguity and uncalibrated photometric stereo.

Towards the end of this document, you will find a "Deliverables" section describing what you need to submit. Throughout the writeup, we also mark in red questions you should answer in your submitted report. Lastly, there is a "Hints and Information" section at the end of this document that is likely to help. We strongly recommend that you read that section in full before you start to work on the assignment.

1. Photometric stereo (100 points)

For the first part of the homework assignments, you will use a set of seven images of a face, measured using a near-orthographic camera with fixed viewpoint, and under different illuminations. These images are available as files `./data/input_N.tif` in the homework ZIP archive, where $N = \{1, \ldots, 7\}$. These are RAW files that have been demosaiced and converted to the *linear* sRGB color space.

Initials (5 points). Load the seven images in Python, convert them to XYZ color space, and extract the luminance channel for each of them. Then, stack the seven luminance channels into a matrix $\boldsymbol{I}$ of size $7 \times P$, where P is the number of pixels of each luminance channel.

Uncalibrated photometric stereo (20 points). Our goal is to recover a 3×1 normal vector $\boldsymbol{n}$ and a scalar albedo a at each pixel of the camera. As we did in class, it will be convenient to consider at each pixel the *pseudo-normal* $\boldsymbol{b} = a \cdot \boldsymbol{n}$. We can stack the pseudo-normals for all pixels into a $3 \times P$ matrix $\boldsymbol{B}$, which we call the *pseudo-normal matrix*.

Additionally, we captured each of our seven input images under some directional light, described by a 3×1 unit-norm vector $\boldsymbol{l}_i$. We can stack the seven light vectors into a 3×7 *light matrix* $\boldsymbol{L}$.

Photometric stereo relies on the "n-dot-l" shading model we discussed in class, which is valid under directional light and Lambertian reflectance. Under this model, we can relate the matrices $\boldsymbol{I}$, $\boldsymbol{L}$ and $\boldsymbol{B}$ through a simple matrix product,

$$\boldsymbol{I} = \boldsymbol{L}^T \cdot \boldsymbol{B}. \tag{1}$$

If all of our assumptions are satisfied exactly, the matrix $\boldsymbol{I}$ will have rank equal to exactly 3. In practice, this will not be exactly the case, because of noise and because the n-dot-l shading assumptions are in practice never perfectly accurate. However, we can find the best approximation (in the least-squares sense) by using SVD to recover the best rank-3 decomposition of matrix $\boldsymbol{I}$. From this decomposition, we can recover estimates for the matrices $\boldsymbol{L}$ and $\boldsymbol{B}$. In turn, from $\boldsymbol{B}$ we can use normalization to recover estimates for the $1 \times P$ albedo matrix $\boldsymbol{A}$ and the $3 \times P$ normal matrix N.

Unfortunately, these estimates are not unique. Let's call $\boldsymbol{L}_e$ and $\boldsymbol{B}_e$ the light and pseudo-normal matrices from the above SVD procedure, and let $\boldsymbol{Q}$ be an invertible 3×3 matrix. Then, the matrices $\boldsymbol{L}_Q = \boldsymbol{Q} \cdot \boldsymbol{L}_e$ and $\boldsymbol{B}_Q = \boldsymbol{Q}^{-T} \cdot \boldsymbol{B}_e$ approximate Equation (1) exactly as well as the original estimates $\boldsymbol{L}_e$ and $\boldsymbol{B}_e$.

Use SVD to recover $\boldsymbol{L}_e$ and $\boldsymbol{B}_e$. For this, you should multiply the left and right singular vectors with the square root of the corresponding singular values. Then convert $\boldsymbol{B}_e$ to per-pixel albedoes $\boldsymbol{A}_e$ and normals $\boldsymbol{N}_e$. Reshape $\boldsymbol{A}_e$ and $\boldsymbol{N}_e$ into single-channel and three-channel images with the same width and height as the original image, and visualize the results. (See Hints and Information for how to visualize normal images.) Additionally, select any non-diagonal matrix $\boldsymbol{Q}$, and visualize the albedo $\boldsymbol{A}_Q$ and normals $\boldsymbol{N}_Q$ you compute from the corresponding $\boldsymbol{B}_Q$.

Enforcing integrability (45 points). As we discussed in class, the per-pixel normals $\boldsymbol{n}(x, y)$ can be related, after appropriate normalization, to the x and y derivatives of the *depth image* $z = f(x, y)$ corresponding to the surface of the object we are scanning. Therefore, the true normals should ideally be integrable, as otherwise they would not correspond to a true surface.

Arbitrary invertible transformations $\boldsymbol{Q}$ *do not* preserve integrability of normal fields. We can thus try to resolve the ambiguity in the normals by finding a Q so that the corresponding normal field $\boldsymbol{N}_Q$ is integrable.

Unfortunately, even though enforcing integrability does help remove some ambiguity, it is not sufficient for uniquely determining the true albedoes and normals. Belhumeur et al. [2] showed that there exists a class of matrices of the form

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \mu & \nu & \lambda \end{bmatrix}, \tag{2}$$

such that, for all μ and ν and for all $\lambda \neq 0$, they are invertible and preserve invertibility. That is, if a normal field corresponding to pseudo-normals $\boldsymbol{B}$ is integrable, then the normal field corresponding to the transformed pseudo-normals

$$\boldsymbol{G}^{-T} \cdot \boldsymbol{B} \tag{3}$$

is also integrable! Therefore, by enforcing integrability, we can only hope to recover the per-pixel albedoes and normals up to a matrix of the form $\boldsymbol{G}$. Put another way, enforcing integrability lets us reduce the degrees of freedom we have from nine (the entries of the original $\boldsymbol{Q}$) down to three (the μ, ν, λ of $\boldsymbol{G}$), but not down to zero. Matrices of the form of Equation (2) correspond to the *generalized bas-relief* (GBR) transformation, so-called because when $\mu = 0$ and $\nu = 0$ this describes the transformation used in bas-relief sculptures to create a perception of 3D shape.

Our goal now is to find a matrix $\boldsymbol{Q}$ such that the corresponding normals $\boldsymbol{N}_Q$ are integrable. We will follow the original derivation by Yuille and Snow [5], which solves for the matrix $\boldsymbol{\Delta} \equiv \boldsymbol{Q}^T$ instead of $\boldsymbol{Q}$. To estimate $\boldsymbol{\Delta}$, first let us denote as $\tilde{\boldsymbol{b}} = \boldsymbol{\Delta}^{-1} \cdot \boldsymbol{b}_e$ the transformed pseudo-normal at each pixel. Then, we can write the integrability constraint at each pixel as:

$$\frac{\partial}{\partial x}\begin{pmatrix} \tilde{\boldsymbol{b}}(2) \\ \tilde{\boldsymbol{b}}(3) \end{pmatrix} = \frac{\partial}{\partial y}\begin{pmatrix} \tilde{\boldsymbol{b}}(1) \\ \tilde{\boldsymbol{b}}(3) \end{pmatrix} \tag{4}$$

$$\Rightarrow \tilde{\boldsymbol{b}}(3)\frac{\partial \tilde{\boldsymbol{b}}(2)}{\partial x} - \tilde{\boldsymbol{b}}(2)\frac{\partial \tilde{\boldsymbol{b}}(3)}{\partial x} = \tilde{\boldsymbol{b}}(3)\frac{\partial \tilde{\boldsymbol{b}}(1)}{\partial y} - \tilde{\boldsymbol{b}}(1)\frac{\partial \tilde{\boldsymbol{b}}(3)}{\partial y}. \tag{5}$$

By replacing $\tilde{\boldsymbol{b}} = \boldsymbol{\Delta}^{-1} \cdot \boldsymbol{b}_e$, and after some manipulation, we arrive at the following linear equation:

$$\begin{bmatrix} A_1 & A_2 & A_3 & A_4 & A_5 & A_6 \end{bmatrix} \boldsymbol{x} = 0, \tag{6}$$

where

$$A_1 = \boldsymbol{b}_e(1)\frac{\partial \boldsymbol{b}_e}{\partial x}(2) - \boldsymbol{b}_e(2)\frac{\partial \boldsymbol{b}_e}{\partial x}(1) \tag{7}$$

$$A_2 = \boldsymbol{b}_e(1)\frac{\partial \boldsymbol{b}_e}{\partial x}(3) - \boldsymbol{b}_e(3)\frac{\partial \boldsymbol{b}_e}{\partial x}(1) \tag{8}$$

$$A_3 = \boldsymbol{b}_e(2)\frac{\partial \boldsymbol{b}_e}{\partial x}(3) - \boldsymbol{b}_e(3)\frac{\partial \boldsymbol{b}_e}{\partial x}(2) \tag{9}$$

$$A_4 = -\boldsymbol{b}_e(1)\frac{\partial \boldsymbol{b}_e}{\partial y}(2) + \boldsymbol{b}_e(2)\frac{\partial \boldsymbol{b}_e}{\partial y}(1) \tag{10}$$

$$A_5 = -\boldsymbol{b}_e(1)\frac{\partial \boldsymbol{b}_e}{\partial y}(3) + \boldsymbol{b}_e(3)\frac{\partial \boldsymbol{b}_e}{\partial y}(1) \tag{11}$$

$$A_6 = -\boldsymbol{b}_e(2)\frac{\partial \boldsymbol{b}_e}{\partial y}(3) + \boldsymbol{b}_e(3)\frac{\partial \boldsymbol{b}_e}{\partial y}(2), \tag{12}$$

and $\boldsymbol{x}$ is a 6×1 vector such that,

$$\boldsymbol{\Delta} = \begin{bmatrix} -x(3) & x(6) & 1 \\ x(2) & -x(5) & 0 \\ -x(1) & x(4) & 0 \end{bmatrix}, \tag{13}$$

where the fixed third column corresponds to the three degrees of freedom the GBR affords us. You can find the details of the above derivation in Yuille and Snow [5].

You can now compute matrix $\boldsymbol{\Delta}$ by performing the following steps:

1. Form the three-channel "pseudo-normal" image $\boldsymbol{b}_e$, and compute its x and y spatial derivatives (use `numpy`'s `gradient`). You should apply a small amount of Gaussian blurring to $\boldsymbol{b}_e$ before computing derivatives. Experiment with blur kernels of $\sigma \in [1, 20]$, and report which value you ended up using.

2. Form a *homogeneous* linear system $\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{0}$, by stacking together linear constraints of the form of Equation (6) for all pixels. Note that, in Equations (7)-(12), for the $\boldsymbol{b}_e$ terms, you should use the *unblurred* pseudo-normal images; the Gaussian blurring is only to compute the spatial derivative terms.

3. Use SVD to solve the resulting homogeneous system for $\boldsymbol{x}$, and estimate $\boldsymbol{\Delta}$ from it using Equation (13).

Once you have found $\boldsymbol{\Delta}$, apply it to the pseudo-normal matrix $\boldsymbol{B}_e$, and then visualize the resulting albedo and normals. Figure 1 shows the expected results.

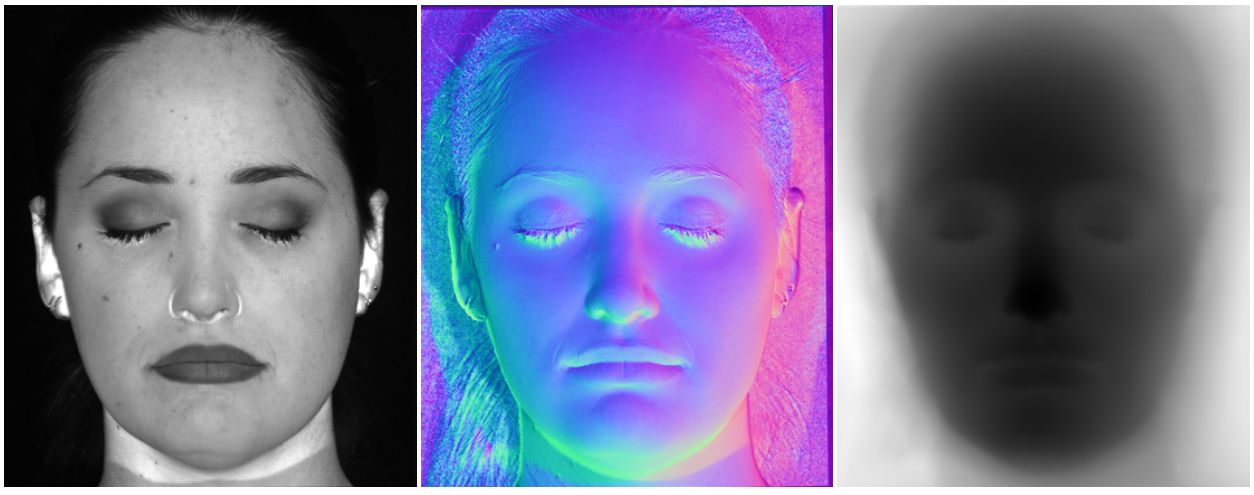

Figure 1: Uncalibrated photometric stereo results, after enforcing integrability. From left to right: estimated albedo ($\times 10$), normals, and depth (normalized to $[0, 1]$).

Normal integration (15 points). Now that you have a normal field, you can use it to compute a surface $Z = f(x, y)$. First, compute from the normals the derivatives $(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})$. Then, integrate the normals to compute the surface $Z = f(x, y)$.

For the integration, you can experiment with the functions `integrate_poisson` and `integrate_frankot` provided in the `./src/cp_hw5.py` in the homework ZIP archive: The first function integrates the derivative vector field by solving the Poisson equation, which is similar to the integration procedure you implemented in Homework Assignment 3. The second function integrates the derivative vector field using a projection method by Frankot and Chellappa [3]. Try both functions, and use the result you like the most.

Visualize the final surface you reconstructed as both a depth image and a 3D surface. (See `matplotlib`'s `plot_surface`.) Figure 1 shows the depth image, and Figure 2 shows two views of the expected surface.

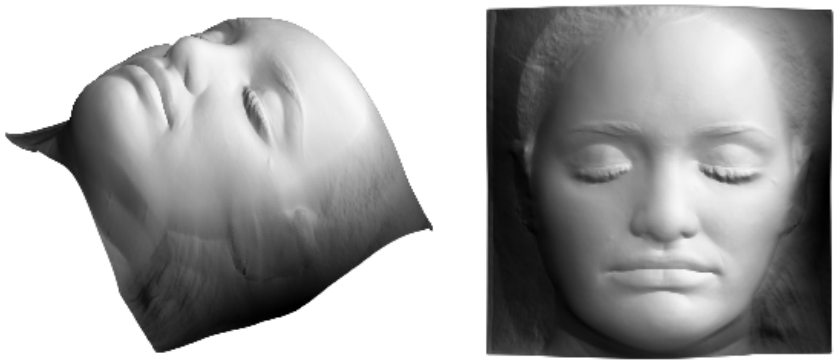

Figure 2: Uncalibrated photometric stereo results. Two views of the recovered face shape.

Additionally, experiment with GBR transformations G for different values μ, ν, λ, until you find one that produces a reasonably undistorted face surface. Report what GBR you end up using, and show the corresponding albedoes, normals, and 3D surfaces.

Calibrated photometric stereo (15 points). You can load the groundtruth light source vectors, in the form of the light matrix $\boldsymbol{L}$, using the function `load_sources` provided in the `./src/cp_hw5.py` in the homework ZIP archive. When the light directions have been calibrated, photometric stereo becomes considerably easier: Given that now both $\boldsymbol{I}$ and $\boldsymbol{L}$ are known, all you have to do is solve the linear system of Equation (1) for the pseudo-normal matrix $\boldsymbol{B}$.

Solve this linear system to recover per-pixel albedoes and normals. Additionally, perform normal integration as before, to recover the 3D surface z. Visualize the recovered albedoes, normals, and surface as before. Figures 3 and 4 show what you should expect to see. How do these results compare to the results of the uncalibrated case?

2. Capture and reconstruct your own shapes (100 points)

You will now perform *uncalibrated* photometric stereo using images you capture with your own camera. For this, you should select two objects you want to scan: First, select an object that approximately satisfies the assumptions of photometric stereo (very diffuse reflectance without much/any glossiness, few interreflections and occlusions). Second, select an object that partially violates the assumptions of photometric stereo (e.g., it has a somewhat glossy reflectance, or it has strong convavities).

For each object, capture at least seven images with a fixed camera and different lighting conditions. Make sure to consult the Hints and Information section for how to best capture these images. Apply uncalibrated photometric stereo to the images you capture, and produce a reconstruction of the albedo, normals, and surfaces for each of the two objects. As your reconstructions will be up to a GBR transformation, you can manually experiment with different GBR transformations until you find the best surface result.

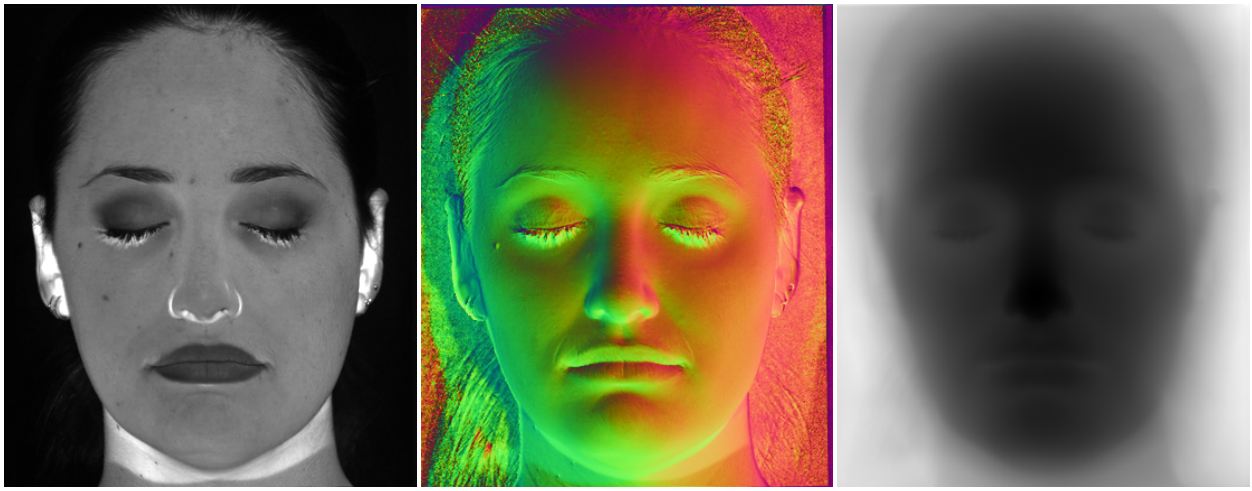

Figure 3: Calibrated photometric stereo results. From left to right: estimated albedo, normals, and depth (normalized to $[0, 1]$).

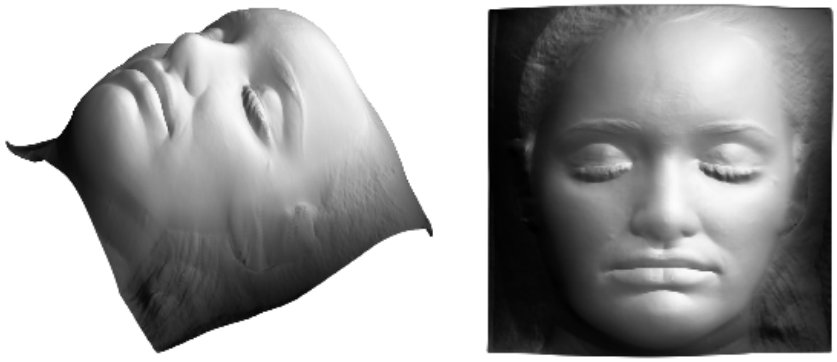

Figure 4: Calibrated photometric stereo results. Two views of the recovered face shape.

For each object, show one of the images you captured, and the albedo, normals, and surface you reconstructed. Additionally, show a rendering of both objects under a new lighting direction of your preference.

3. Bonus: Resolving the GBR ambiguity (100 points)

Following the discovery of the GBR ambiguity, there have been a number of techniques that use different heuristics to try to resolve the ambiguity when performing uncalibrated photometric stereo. Below we mention two that have been particularly successful. For up to 100 points of extra credit (50 points for each method), you can read the corresponding paper, implement their method, and apply it to both the images that came with the homework for Part 1, and the images you captured for Part 2. (You can still get partial credit for incomplete implementations, and for applying the method to just one set of images).

Entropy minimization (50 points). This technique was introduced by Alldrin and Kriegman [1]. The intuition behind it is that many real-world objects have a relatively small number of albedo values (e.g., different parts of the surface painted with a few different colors). Therefore, among all possible GBR

transformations, we should prefer the one that reduces the *variability* of the recovered albedo values. The paper proposes measuring variability using entropy.

Using perspective cameras (50 points). The GBR ambiguity is, in part, a consequence of the fact that we assume an orthographic camera. When the camera we use is perspective, then normals and albedoes can be reconstructed *exactly*, and the surface can be reconstructed up to a *one* degree of freedom (a global scale). This was proven by Papadhimitri and Favaro [4], who also show how one can do the reconstruction in this case.

Deliverables

When submitting your solution, make sure to follow the homework submission guidelines available on the course website (http://graphics.cs.cmu.edu/courses/15-463/assignments/submission_guidelines.pdf). Your submitted solution should include the following:

- A PDF report explaining what you did for each problem, including the various visualizations of albedoes, normals, and surfaces, as well as renderings of images, that are requested throughout problems 1 and 2, as well as answers to all questions asked throughout both problems. The report should include any figures and intermediate results that you think may help. Make sure to include explanations of any issues that you may have run into that prevented you from fully solving the assignment, as this will help us determine partial credit.
- All of your Python code, as well as a `README` file explaining how to use the code.
- If you do Bonus Part 3: Your PDF report should include a detailed description of the experiments you performed for either paper. You should also show and compare the albedoes, normals, and surfaces you reconstructed with each method.
- For the photography competition: Submit an image of one of the surface reconstructions you produced for Problem 2, named as `competition_entry.png`.

Hints and Information

Normal visualization. You may be tempted to visualize the three-channel "images" containing the per-pixel normals directly as an RGB image, e.g., using `matplotlib`'s `imshow`. However, you need to take into account that the coordinates of these normals will have values in the range $[-1, 1]$. Therefore, before displaying them as an RGB image, you should first transform them to the range $[0, 1]$. The standard way to do this is to map each normal $\boldsymbol{n}$ to $(\boldsymbol{n} + 1)/2$. This is how we produced all the images of normals in the write-up and you should do the same for your own results.

Gaussian blurring. The results you obtain in the uncalibrated photometric stereo part will depend *very significantly* on the σ value you use for the Gaussian blurring. Make sure to experiment with a few values.

Handling color. When Gaussian blurring the 3-channel pseudo-normal image $\boldsymbol{b}_e$, you need to make sure that each channel is blurred separately. If you pass the 3-channel image directly to `scipy`'s `gaussian_filter`, it will end up blurring with a 3D kernel, and thus blur across the three channels. This is incorrect. What you should do instead is have a for loop that processes each channel of be one at a time.

Normal integration. When converting normals to heightfield derivatives for integration, you need to divide by each normal's z coordinate. This creates numerical problems when the z coordinate is (close to) zero; you can avoid these by adding a small ϵ to the z coordinate before dividing.

Enforcing integrability. In uncalibrated photometric stereo, after enforcing integrability, integrating the resulting normals may produce a surface that looks like it is "inside-out". You can deal with this by applying

the following GBR transform matrix to your pseudo-normals *after enforcing integrability*:

$$\boldsymbol{G}_F = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}. \tag{14}$$

Make sure to recompute the normals and reintegrate the surface afterwards.

Memory management. The matrix of the homogeneous linear system in Equation (6) is quite large, and Python will likely run out of memory when trying to compute its singular value decomposition. To deal with this, you should use the option `full_matrices=False` when you use `numpy.linalg.svd` (i.e., `numpy.linalg.svd(I, full_matrices=False)`).

Negative values. When computing the output radiance from a field of surface normals $\boldsymbol{n}(x, y)$ and albedoes $a(x, y)$ illuminated from direction $\boldsymbol{l}$, clip negative values using $L = \max(0, a\boldsymbol{n} \cdot \boldsymbol{l})$.

Visualizing surfaces. The following code demonstrates one way to display a surface in Python.

```
import matplotlib.pyplot as plt
from matplotlib.colors import LightSource
from mpl_toolkits.mplot3d import Axes3D

# Z is an HxW array of surface depths
H, W = Z.shape
x, y = np.meshgrid(np.arange(0,W), np.arange(0,H))

# set 3D figure
fig = plt.figure()
ax = fig.gca(projection='3d')

# add a light and shade to the axis for visual effect
# (use the '-' sign since our Z-axis points down)
ls = LightSource()
color_shade = ls.shade(-Z, plt.cm.gray)

# display a surface
# (control surface resolution using rstride and cstride)
surf = ax.plot_surface(x, y, -Z, facecolors=color_shade, rstride=4, cstride=4)

# turn off axis
plt.axis('off')
plt.show()
```

Heterogeneous photometric stereo. In calibrated photometric stereo, when solving the heterogeneous linear system of Equation (1), do *not* flatten the pseudo-normal matrix $\boldsymbol{B}$, as the resulting system will not fit in memory. Instead, recover $\boldsymbol{B}$ directly, by solving a system of the form $A_{7\times 3} \cdot x_{3\times N} = b_{7\times N}$, where N is the number of pixels.

Capturing your own data. The quality of your results in part 2 will critically depend on how well the measurements you take match the imaging conditions assumed by photometric stereo, namely linearity of measurements, orthographic projection, and directional light. Below are some tips that can help maximize your chances of success.

You should set your camera to fully manual mode, as it is important that exposure time, aperture, focus, and so on do not change as you capture different images. You should rotate the zoom ring on your lens so

that you are using the maximum possible focal length of the lens. Once you have done that, you should focus your camera so that the object is sufficiently far away from the front of the lens (at least one-two meters). This is necessary in order for your camera to be approximately orthographic. After focusing, set the aperture to the largest aperture you can have while making sure that all of your object remains within the depth of field. Finally, set the shutter speed so that the object is well-exposed—no saturated pixels and no very black pixels. As we explain below, you should *not* increase exposure by increasing ISO. Given that you will need to take multiple images without any motion, you should make sure to mount your camera on a tripod and that it is tethered to your laptop.

Remember that you need linear measurements. Therefore, you should make sure to set the camera to produce RAW images. It should also help to set its white balancing option to manual, and its output color space to sRGB. After you have captured your images, you can use `dcraw` to convert them to demosaiced TIFF, in the same way as you did in Homework Assignment 2.

For the lighting, you should use a small light that is far away from the object—at least as far away as the camera. The flash from your mobile phone, or a bright but small desk light would be good light sources. Make sure to capture the object from a sufficiently large set of angles, including lighting from left, right, top, bottom, and front. See the set of images provided for Part 1 to get a sense of the degree of variability you should have. Finally, you should capture your images in a dark room, where there is no (or at least, very little) ambient light.

Lastly, you will find that the quality of your results will be highly sensitive to the amount of noise in your images. This is because of the gradient-of-pseudo-normals operation you apply while enforcing integrability. To reduce the amount of noise, you should make sure to set the camera ISO to a low value (ideally 100). It can also help if, for each lighting direction, you capture multiple images and average them, then use the averages for photometric stereo. Finally, if all else fails, you can process your images with a small box filter (kernel size 2 or 4) to reduce noise.

Credits

A lot of inspiration for this assignment, as well as parts of the write-up and the data for Part 1, came from the computer vision course offered by Todd Zickler at Harvard.

References

[1] N. G. Alldrin, S. P. Mallick, and D. J. Kriegman. Resolving the generalized bas-relief ambiguity by entropy minimization. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–7. IEEE, 2007.

[2] P. N. Belhumeur, D. J. Kriegman, and A. L. Yuille. The bas-relief ambiguity. *International journal of computer vision*, 35(1):33–44, 1999.

[3] R. Frankot and R. Chellappa. A method for enforcing integrability in shape from shading algorithms. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (4):439–451, 1988.

[4] T. Papadhimitri and P. Favaro. A new perspective on uncalibrated photometric stereo. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1474–1481, 2013.

[5] A. Yuille and D. Snow. Shape and albedo from multiple images using integrability. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 158–164. IEEE, 1997.