Carnegie Mellon University

16-720: Computer Vision

Homework 5: Photometric Stereo

- Due date: 12/05/2024
- **Gradescope submission.** You will need to submit both (1) a `YourAndrewName.pdf` and (2) a `YourAndrewName.zip` file containing your code, either as standalone python (`.py` or colab notebooks `.ipynb`). Remember you *must* use Gradescope's functionality to mark pages in your PDF that answer individual questions, and please make sure *all* text is legible and large enough to read; if not, you will risk loosing points!
- **Suggestions for creating a PDF.** We suggest you create a PDF by `printing` your colab notebook from your browser. However, you are responsible for making sure all your code and results are visible and not cut off. Long lines of code can print poorly; we suggest you add a backslash to break a single long line into multiple lines. You also may wish to look at this video for alternate pathways to convert notebooks to PDFs: https://youtu.be/-Ti9Mm21uVc?si=bo4kHfp2BoJvPpZI. In some cases, you may wish to download images or screengrabs and explicitly append them to your PDF using online tools such as https://combinepdf.com/, or insert empty cells to create adequate spacing so that you don't need to zoom out too much to prevent content from being clipped. You may also find it useful to look at this post: https://askubuntu.com/questions/2799/how-to-merge-several-pdf-files.
- For each question, make sure to include all explanations, code, and results related to that question. When re-using functions from previous questions, you do not need to show them again. In some questions, you might be asked to explain something **and** write a function **and** show the result, make sure to include all in your pdf submission to Gradescope.

The starter code can be found at the course gdrive folder (you will need your andrew account to access): https://drive.google.com/drive/folders/18PCONvpfIQbFNcB_6Dm0DmAyJd8gXCQ3?usp=drive_link

We recommend editing and running your code in Google Colab, although you are welcome to use your local machine instead.

Please remember to list your collaborators in your Colab file.

1 Calibrated photometric stereo (75 points)

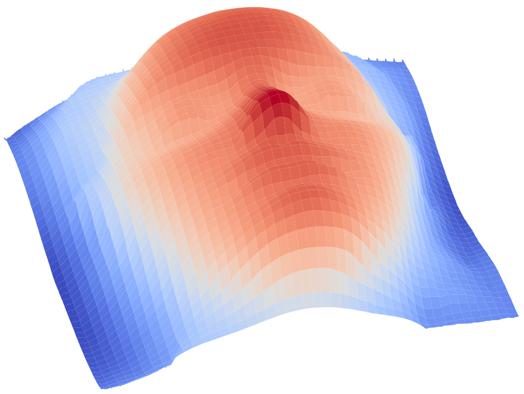
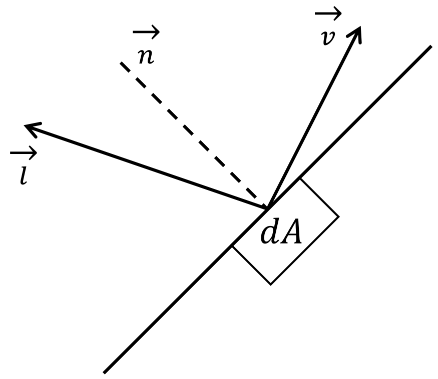


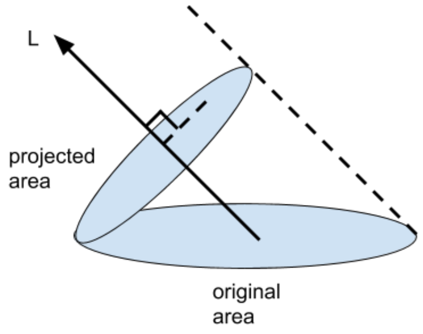
Figure 1: 3D reconstruction result from calibrated photometric stereo

The skeleton code is provided in the Colab file in multiple functions.

Image formation. We will now look at image formation under our specific assumptions. We will understand and implement the process that occurs in a scene when we take a picture with an orthographic camera.



(a) Geometry of photometric stereo



(b) Projected area

Figure 2: Geometry of photometric stereo

(a, 5 points) Understanding n-dot-l lighting. In your Colab file, explain the geometry of the n-dot-l lighting model from Fig. 2a. Where does the dot product come from? Where does projected area (Fig. 2b) come into the equation? Why does the viewing direction not matter?

(b, 10 points) Rendering n-dot-l lighting. Consider a uniform fully reflective Lambertian sphere with its center at the origin and a radius of 0.75 cm (Fig 3). An orthographic camera located at (0, 0, 10) cm looks towards the negative z-axis with its sensor axes aligned and centered on the x- and y-axes. The pixels on the camera are squares 7 μm in size, and the resolution of the camera is 3840 $\times$ 2160 pixels. Simulate the appearance of the sphere under the n-dot-l model with directional light sources with incoming lighting directions $(1, 1, 1)/\sqrt{3}$, $(1, -1, 1)/\sqrt{3}$ and $(-1, -1, 1)/\sqrt{3}$ in the function `renderNDotLSphere` (individual images for all three lighting directions). Note that your rendering isn't required to be absolutely radiometrically accurate: we need only evaluate the n-dot-l model.

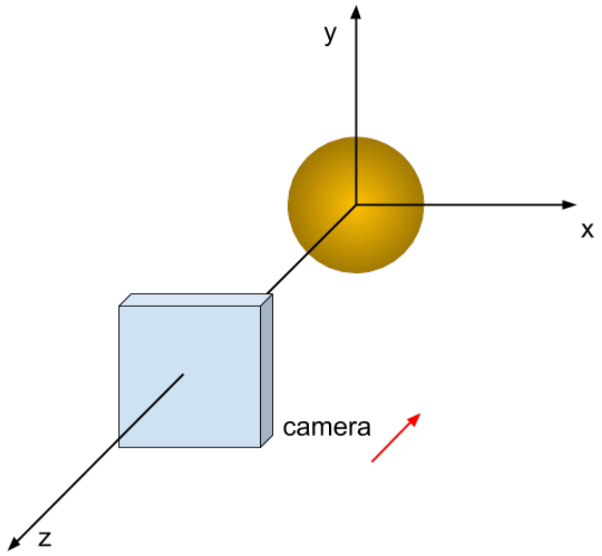


Figure 3: Setup for Problem 1b

Inverting the image formation model. Armed with this knowledge, we will now invert the image formation process given the lighting directions. Seven images of a face lit from different directions are given to us, along with the ground-truth directions of light sources. These images are taken at lighting directions such that there is not a lot of occlusion or normals being directed opposite to the lighting, so we will ignore these effects. The images are in the `/content/data/` directory, named as `input_n.tif` for the n^{th} image. The source directions are given in the file `/content/data/sources.npy`.

The data is loaded using the `loadData` utils function provided to you. It converts the RGB images into the XYZ color space and extracts the luminance channel. It vectorizes these luminance images and stacks them in a $7 \times P$ matrix, where P is the number of pixels in each image. This is the matrix $\mathbf{I}$, which is given to us by the camera. Finally, it loads the source file and converts it to a 3×7 matrix L.

(c, 10 points) Initials. Recall that in general, we also need to consider the reflectance, or albedo, of the surface we're reconstructing. We find it convenient to group the normals and albedos (both of which are a property of only the surface) into a pseudonormal $\mathbf{b} = a \cdot \mathbf{n}$, where a is the scalar albedo. We then have the relation $\mathbf{I} = \mathbf{L}^T\mathbf{B}$, where the $3 \times P$ matrix $\mathbf{B}$ is the set of pseudonormals in the images. With this model, explain why the rank of $\mathbf{I}$ should be 3. Perform a singular value decomposition of $\mathbf{I}$ computed by `loadData` in the previous question, and report the singular values in your Colab file. Do the singular values agree with the rank-3 requirement? Show the singular values and explain.

(d, 20 points) Estimating pseudonormals. Since we have more measurements (7 per pixel) than variables (3 per pixel), we will estimate the pseudonormals in a least-squares sense. Note that there is a linear relation between $\mathbf{I}$ and $\mathbf{B}$ through $\mathbf{L}$: therefore, we can write a linear system of the form $\mathbf{Ax} = \mathbf{y}$ out of the relation $\mathbf{I} = \mathbf{L}^T\mathbf{B}$ and solve it to get $\mathbf{B}$. Solve this linear system in the function `estimatePseudonormalsCalibrated`. In your Colab file, mention how you constructed the matrix $\mathbf{A}$ and the vector $\mathbf{y}$.

Note that here matrices you end up creating might be huge and might not fit in your computer's memory. In that case, to prevent your computer from freezing or crashing completely, make sure to use the `sparse` module from `scipy`. You might also want to use the sparse linear system solver and kronecker product in the same module.

(e, 10 points) Albedos and normals. Estimate per-pixel albedos and normals from matrix $\mathbf{B}$ in `estimateAlbedosNormals`. Note that the albedos are the magnitudes of the pseudonormals by definition. Calculate the albedos, reshape them into the original size of the images and display the resulting image using the utils function `displayAlbedosNormals`. Comment on any unusual or unnatural features you may find in the albedo image, and on why they might be happening. Make sure to display in the `gray` colormap.

The per-pixel normals can be viewed as an RGB image. Reshape the estimated normals into an image with 3 channels and display it in the function `displayAlbedoNormals`. Note that the components of these normals will have values in the range $[-1, 1]$. You will need to rescale them so that they lie in $[0, 1]$ to display them properly as RGB images. Do the normals match your expectation of the curvature of the face? Make sure to display in the `rainbow` colormap. Your results should look like that in Fig. 4.

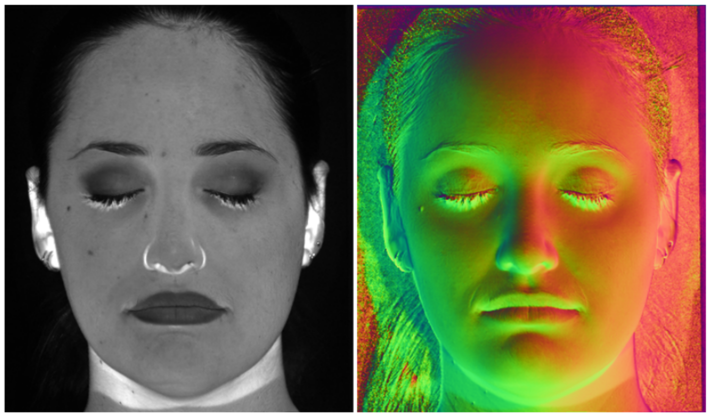


Figure 4: Calibrated photometric stereo results. Estimated albedo ($\times 10$) and normals.

(f, 5 points) Normals and depth. We will now estimate the actual shape of the face from the normals. Represent the shape of the face as a 3D depth map given by a function $z = f(x, y)$. Let the normal at the point (x, y) be $\mathbf{n} = (n_1, n_2, n_3)$. Explain, in your Colab file, how $\mathbf{n}$ is related to the *partial derivatives* of f at (x, y): $f_x = \partial f(x, y)/\partial x = -n_1/n_3$ and $f_y = \partial f(x, y)/\partial y = -n_2/n_3$. Derive f_x and f_y in terms of $\{n_1, n_2, n_3\}$ without skipping steps.

Normal integration. Given that the normals represent the derivatives of the depth map, we will integrate the normals obtained in (f). We will use a special case of the Frankot-Chellappa algorithm for normal integration, as given in [1]. You can read the paper for the general version of the normal integration algorithm.

(g, 5 points) Understanding integrability of gradients. Consider the 2D, discrete function g on space given by the matrix below. Find its x and y gradient, given that gradients are calculated as $g_x(x_i, y_j) = g(x_{i+1}, y_j) - g(x_i, y_j)$ for all i, j (and similarly for y). Let us define (0, 0) as the top left, with x going in the horizontal direction and y in the vertical.

$$g = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \tag{1}$$

Note that we can reconstruct the entire of g given the values at its boundaries using g_x and g_y. Given that $g(0, 0) = 1$, perform these two procedures:

1. Use g_x to construct the first row of g, then use g_y to construct the rest of g;

2. Use g_y to construct the first column of g, then use g_x to construct the rest of g.

Are these the same?

Note that these were two ways to reconstruct g from its gradients. Given arbitrary g_x and g_y, these two procedures will not give the same answer, and therefore this pair of gradients does not correspond to a true surface. Integrability implies that the value of g estimated in both these ways (or any other way you can think of) is the same. How can we modify the gradients you calculated above to make g_x and g_y non-integrable? Why may the gradients estimated in the way of (g) be non-integrable? Note all this down in your Colab file.

The Frankot-Chellappa algorithm for estimating shapes from their gradient first projects the (possibly non-integrable) gradients obtained in (g) above onto the set of all integrable gradients. The resulting (integrable) projection can then be used to solve for the depth map. The function `integrateFrankot` in utils implements this algorithm given the two surface gradients.

(h, 10 points) Shape estimation. Write a function `estimateShape` to apply the Frankot-Chellappa algorithm to your estimated normals. Once you have the function $f(x, y)$, plot it as a surface in the function `plotSurface`. The result we expect of you is shown in Fig. 1.

2 Uncalibrated photometric stereo (50 points)

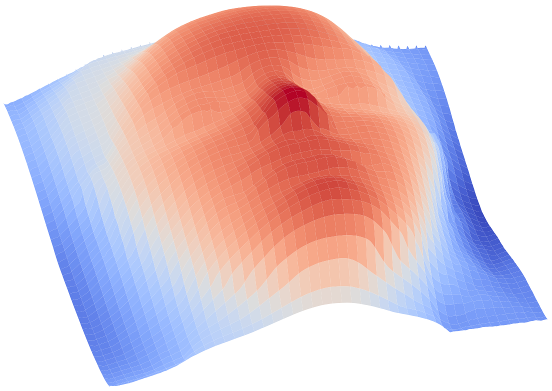


Figure 5: 3D reconstruction result from uncalibrated photometric stereo

We will now put aside the lighting direction and estimate shape directly from the images of the face. As before, we load the lights into $\mathbf{L}_0$ and images in the matrix $\mathbf{I}$. We will not use the matrix $\mathbf{L}_0$ for anything except as a comparison against our estimate in this part. All code for this question goes in your Colab file.

(a, 10 points) Uncalibrated normal estimation. Recall the relation $\mathbf{I} = \mathbf{L}^T\mathbf{B}$. Here, we know neither $\mathbf{L}$ nor $\mathbf{B}$. Therefore, this is a matrix factorization problem with the constraint that with the estimated $\hat{\mathbf{L}}$ and $\hat{\mathbf{B}}$, the rank of $\hat{\mathbf{I}} = \hat{\mathbf{L}}^T\hat{\mathbf{B}}$ be 3 (as you answered in a previous question), and the estimated $\hat{\mathbf{I}}$ and $\hat{\mathbf{L}}$ have appropriate dimensions.

It is well-known that the best rank-k approximation to a $m \times n$ matrix $\mathbf{M}$, where $k \leq \min\{m, n\}$ is calculated as the following: perform a singular value decomposition SVD $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, set all singular values except the top k from $\mathbf{\Sigma}$ to 0 to get the matrix $\hat{\mathbf{\Sigma}}$, and reconstitute $\hat{\mathbf{M}} = \mathbf{U}\hat{\mathbf{\Sigma}}\mathbf{V}^T$. Explain in your Colab file how this can be used to construct a factorization of the form detailed above following the required constraints.

(b, 10 points) Calculation and visualization. With your method, estimate the pseudonormals $\hat{\mathbf{B}}$ in `estimatePseudonormalsUncalibrated`, visualize the resultant albedos and normals in the `gray` and `rainbow` colormaps respectively. A sample result has been shown in Fig. 6.

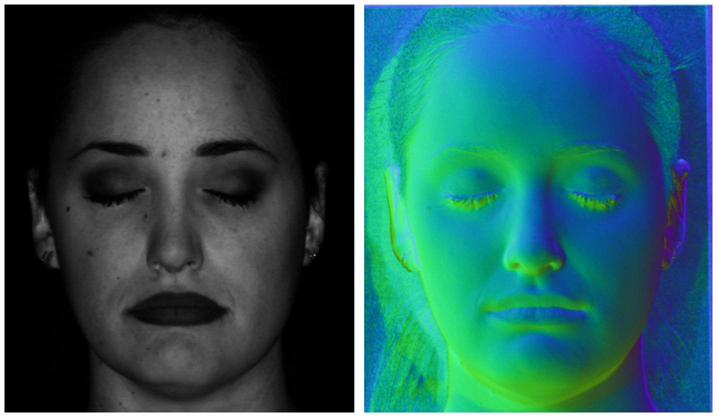


Figure 6: Uncalibrated photometric stereo. Estimated albedo and normals.

(c, 5 points) Comparing to ground truth lighting. In your Colab file, compare the $\hat{\mathbf{L}}$ estimated by the factorization above to the ground truth lighting directions given in Q1. Are they similar? Unless a special choice of factorization is made, they will be different. Describe a simple change to the procedure in (a) that changes the $\hat{\mathbf{L}}$ and $\hat{\mathbf{B}}$, but keeps the images rendered using them the same using only the matrices you calculated during the singular value decomposition (U/S/V). (No need to code anything for this part, just describe the change in your Colab file)

(d, 5 points) Reconstructing the shape, attempt 1. Use the given implementation of the Frankot-Chellappa algorithm from the previous question to reconstruct a 3D depth map and visualize it as a surface in the 'coolwarm' colormap as in the previous question. Does this look like a face?

Enforcing integrability explicitly. The ambiguity you demonstrated in (c) can be (partially) resolved by explicitly imposing integrability on the pseudonormals recovered from the factorization. We will follow the approach in [2] to transform our estimated pseudonormals into a set of pseudonormals that are integrable. The function `enforceIntegrability` in utils implements this process.

(e, 5 points) Reconstructing the shape, attempt 2. Input your pseudonormals into the `enforceIntegrability` function, use the output pseudonormals to estimate the shape with the Frankot-Chellappa algorithm and plot a surface as in the previous questions (results shown in Fig. 5). Does this surface look like the one output by calibrated photometric stereo?

Hint After integration, your surface may look 'inside-out'. You can deal with this by applying the following GBR (generalized bas-relief) transform matrix to your pseudo-normals.

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

Check below for what $\mathbf{G}$ is and once this transformation is applied, make sure to recompute the normals and reintegrate the surface afterwards.

The generalized bas-relief ambiguity. Unfortunately, the procedure in (e) resolves the ambiguity only up to a matrix of the form

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

where $\lambda > 0$, as shown in [3]. This means that for any μ, ν and $\lambda > 0$, if $\mathbf{B}$ is a set of integrable pseudonormals producing a given appearance, $\mathbf{G}^{-T}\mathbf{B}$ is another set of integrable pseudonormals producing the same appearance. The ambiguity introduced in uncalibrated photometric stereo by matrices of this kind is called the *generalized bas-relief ambiguity* (French, translates to 'low-relief', pronounced 'bah ree-leef').

(f, 5 points) Why low relief? Vary the parameters μ, ν and λ in the bas-relief transformation and visualize the corresponding surfaces. Include at least **six** (two with each parameter varied) of the significant ones in your Colab file. Looking at these, what is your guess for why the bas-relief ambiguity is so named? In your Colab file, describe how the three parameters affect the surface.

(g, 5 points) Flattest surface possible. With the bas-relief ambiguity, in Eq. 2, how would you design a transformation that makes the estimated surface as flat as possible?

(h, 5 points) More measurements. We solved the problem with 7 pictures of the face. Will acquiring more pictures from more lighting directions help resolve the ambiguity?

References

[1] R. T. Frankot and R. Chellappa. A method for enforcing integrability in shape from shading algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(4):439–451, July 1988.

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

[3] P. N. Belhumeur, D. J. Kriegman, and A. L. Yuille. The bas-relief ambiguity. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1060–1066, June 1997.