Due: 8th March 2022 11:55 PM Total Points: **15**

1 Depth from Binocular Stereo

One of the features our brains use to discern scene depth is stereo disparity, i.e. how the images seen by both our eyes differ. Relative to the field-of-view of each of our eyes, objects which are close to us will appear to be in drastically different locations, whereas objects which are far will appear almost identically positioned. This effect occurs whenever two or more cameras are used to image a scene, so long as they are separated by some finite distance from one another. Under the right circumstances this object disparity can be used as a surprisingly robust estimator for scene depth[1].

Your task is therefore: given a stereo image pair, create a third image which contains at every pixel the depth of the scene. The scene depth is directly related to object disparity by geometry, but to make that conversion, you must know the separation between the cameras. Since this separation is unknown, you can present the disparity at every pixel as a proxy for scene depth.

Two images are given, `tsukuba_l.ppm` and `tsukuba_r.ppm`, taken by left and right cameras respectively. The images are already rectified, meaning for each camera each row of pixels lies on exactly one epipolar plane (and the planes are the same for both cameras). That is, if some object feature is imaged to a point on k^{th} row of the left image, the same object feature will be imaged to a point on the k^{th} row of the right image (assuming it is visible in both images). This is convenient, as literally counting the pixel separation between the relative locations of such a point yields its disparity (and therefore its depth). But, for any scene other than point light sources surrounded by darkness, actually finding these corresponding points is non-trivial[2].

One way to find which pixels correspond to which in the images' rows is to consider a small patch centered on each pixel in question. For a wide variety of scenes, it is reasonable to assume that this patch would be "similar" in both the images, despite possibly appearing in different locations. The extent of this similarity yields a quantifiable metric to find the corresponding scene points. Various measures of similarity can be used, such as the pixel-wise sum of squared distances or the patches' normalized cross-correlation.

[1] In fact, this is how many "3D" cinematography cameras work.

[2] And when something is difficult, researchers give it a name, in this case, "The Correspondence Problem."

1.1 Solving Correspondence

5 points

1. Choose a patch size and a similarity metric, and provide a rationale for why you chose what you did.

2. Consider each of the following pixels $\mathbf{p}_{i=1,2,3,4,5,6}(k,m)$ at $(row = k,\ col = m)$ of the left image and their associated patches:

   (a) $\mathbf{p}_1(136, 83)$

   (b) $\mathbf{p}_2(203, 304)$

   (c) $\mathbf{p}_3(182, 119)$

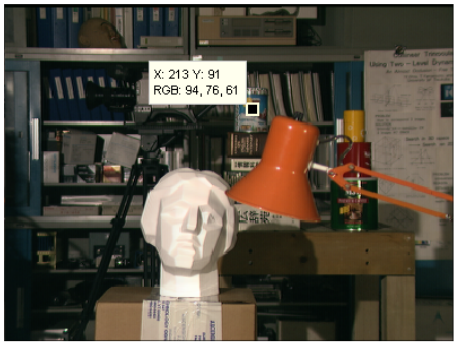
   (d) $\mathbf{p}_4(186, 160)$

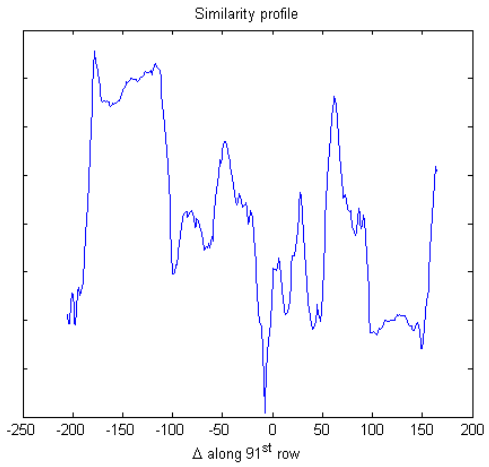
   (e) $\mathbf{p}_5(123, 224)$

   (f) $\mathbf{p}_6(153, 338)$

   For each pixel $\mathbf{p}_i(k,m)$ of left image, calculate and plot the output (profile) of your similarity metric with respect to diparity Δ to all the pixels along the k^{th} row of the right image. Interpret and give reasons for the profiles observed. Does your plot clearly have a reliable extremum? If so, why? If not, why not?

   An example profile is shown below (plotted for pixel $\mathbf{p}(91, 213)$ of left image using arbitrary similarity metric):





   (Where, in the plot, $\Delta = m_2 - m_1$ for similarity measure between pixel $\mathbf{p}(k, m_1)$ of left image and pixel $\mathbf{q}(k, m_2)$ of right image)

3. From your observations of the similarity profiles you just calculated, devise a strategy to estimate the most similar patch. Justify your choices. Include a test to reject patches which have no good matches, and denote such problem-pixels as undefined.

4. Repeat this calculation for all left image pixels. Find the point of maximal similarity in each case using your estimation strategy. The position of this estimate relative to the position of the left image pixel under consideration is the estimated disparity value, Δ_{est}. Accumulate all these disparities (one for each left image pixel) into a matrix and display the matrix as a heatmap. This is the disparity map. Be sure to mark the undefined locations clearly in some way.

5. To fill in the undefined values, use an interpolation technique. There are a variety of possible options, including nearest-neighbor, linear, and cubic[3]. Choose one, describe how it works in your own words, and use it to estimate the unknown disparities. Provide a rationale for why you chose what you did. Display the resulting disparity map.

6. The technique described above attempts to find each pixel correspondence independently. It uses a hard constraint of epipolar geometry and a soft constraint of patch similarity in order to find the correspondence. But, in addition, we can also have other soft constraints like smoothness, uniqueness and ordering of correspondences. These constraints help to relate between pixels on the same row (epipolar line). Briefly explain the three constraints (smoothness, uniqueness and ordering) and mention the cases when they would fail.

1.2 Scanline stereo

5 points

1. Read the included paper[4], "Stereo by intra- and inter-scanline search using dynamic programming," until section 3.2. What are the key points of this portion of the paper? With their approach in mind, how could smoothness, uniqueness and ordering constraints be incorporated to improve the quality of correspondences? Note: although this paper discusses edge correspondences, many of the concepts can be applied to patch correspondences also.

2. Implement dynamic programming to solve the patch correspondence problem. They refer to this as the "intra-scanline search" in the paper.

[3]This is such a common operation that Matlab has many built-in functions which can do this for you. See `griddata, triscatterinterp.`

[4]Ohta, Yuichi, and Takeo Kanade. "Stereo by intra-and inter-scanline search using dynamic programming." Pattern Analysis and Machine Intelligence, IEEE Transactions on 2 (1985): 139-154.

Grad Credits: Photometric Stereo for Lambertian object

5 points

In this section, you will implement a simple version of photometric stereo for Lambertian objects in MATLAB. To aid you, radiometric calibration and discovering lighting angles have been performed for you. That is, image intensities have been made linear and normalized to account for differences in distance between the light source and object and you have been provided with the lighting angle for each view (`lighting.mat`).

You are expected to:

1. Read in 12 images for a given scene
2. Compute surface normal vectors for each point within a specified mask
3. Recover albedo information within the region of interest
4. Find a depth map for the object.

Background

(Notation and algorithm presentation inspired from Forsyth and Ponce.)

The appearance of diffuse objects (in the absence of ambient illumination) in a single image can be expressed as

$$I(x, y) = \kappa_\rho(x, y)\boldsymbol{N}(x, y) \cdot \boldsymbol{L} \tag{1}$$

where $\boldsymbol{I}$ is the intensity of a given pixel (x, y), κ_ρ is the albedo (scalar), $\boldsymbol{L}$ is the lighting direction, and $\boldsymbol{N}$ is a 3D vector of the surface normal in the x,y, and z directions. Each pixel corresponds to a single point on the physical object so we may use the terms pixel location and object point interchangeably.

In our problem, the pixel intensity and lighting direction are known while the albedo and surface normal are unknown. We group the unknowns in a single variable $g(x, y) = \kappa_\rho(x, y)\boldsymbol{N}(x, y)$ and simplify Equation 1

$$I(x, y) = g(x, y) \cdot \boldsymbol{L} \tag{2}$$

Here we have 3 unknowns (2 for the surface normal and 1 for the albedo) and only one measure of pixel intensity. Therefore we require at least 3 images under different lighting conditions to recover the surface normal.

If we have n sources and know the lighting direction for each, we may stack them into the matrix

$$\mathcal{L} = \begin{pmatrix} L_1^T \\ L_2^T \\ \vdots \\ L_n^T \end{pmatrix}. \tag{3}$$

For each image point (x, y), we stack the pixel intensities into a vector

$$\boldsymbol{i}(x, y) = [I_1(x, y), I_2(x, y), \cdots, I_n(x, y)] \tag{4}$$

Substituting equations 3 and 4 into 2 gives us the linear system

$$\boldsymbol{i}(x, y) = \mathcal{L}\boldsymbol{g}(x, y) \tag{5}$$

where $\boldsymbol{g}(x, y)$ can be obobtained by solving a least squares minimization problem. $\boldsymbol{i}(x, y)$ is a 12×1 vector, $\mathcal{L}$ is a 12×3 matrix and $\boldsymbol{g}(x, y)$ is 3×1 vector.

Measuring Albedo and Recovering Normal vectors

Recall that $\boldsymbol{g}(x, y)$ is a unit normal ($\boldsymbol{N}$) scaled by the albedo (κ_ρ), therefore the albedo is nothing more than the L_2 norm of $\boldsymbol{g}(x, y)$. Similarly, the surface normal at (x, y) is simply $\boldsymbol{g}(x, y)$ divided by κ_ρ,

$$\begin{aligned} \kappa_\rho(x, y) &= \|\boldsymbol{g}(x, y)\|_2 \\ \boldsymbol{N}(x, y) &= \frac{1}{\kappa_\rho(x, y)}\boldsymbol{g}(x, y). \end{aligned} \tag{6}$$

Recovering Depth Maps by Direct Integration

The surface normal ($\boldsymbol{N}$) that you recovered in Equation 6 has only two degrees of freedom; the magnitude in the z direction is simply a function of x and y. Also, when recovering a depth map, we only care about the change along the z axis. Thus the surface can be expressed as $(x, y, f(x, y))$ and we can recover depth by integrating the change in the z-normal along the rows and columns.

Let $N_1(x, y)$, $N_2(x, y)$, and $N_3(x, y)$ be the components of N along the x-,y-, and z-axis respectively. The change in the depth at a point (x, y) is then

$$\frac{\partial f}{\partial x} = \frac{N_1(x, y)}{N_3(x, y)} = p(x, y) \text{ and } \frac{\partial f}{\partial y} = \frac{N_2(x, y)}{N_3(x, y)} = q(x, y) \tag{7}$$

Using this representation we can perform a crude integration to recover a depth map (taken from Forsyth and Ponce, page 85):

```
Assume the top left corner has a depth of zero
For each pixel in the left column of the depth map
  depth value = previous depth value + corresponding q value
end
For each row
  For each element of the row except the leftmost
    depth value = previous depth value + corresponding p value
  end
```

This basic integration is susceptible to noisy measurements in p and q and errors are propagated along the row. There are additional integration methods which may be used to return better depth maps and I have provided a function `refineDepthMap.p` that will return such a depth map. The p-code runs exactly like any MATLAB function, but the source has been obfuscated.

Assignment

You must implement a simple photometric stereo algorithm for Lambertian scenes. To increase speed, you need only run computations within a given region provided in the '...mask.png' image for each scene. You are also given the lighting direction matrix $\mathcal{L}$.

In your report you will reconstruct the "Buddha" scene. Give a brief overview of each step and be sure to show (1) the albedo for the red, green, and blue channels, (2) surface normal vectors for the luminance channel, (3) recovered depth map using the direct integration of the surface normal vectors and (4) refined depth map using the provided code. Please note that the albedo is computed for each channel independently and that the surface normal vectors are computed using the grayscale representation of the image.

You must write your own code for this assignment. The only outside code permitted is the provided `refineDepthMap` function.

At the end of the report, be sure to address any shortcomings of your algorithm and offer suggestions for improved performance. Do you trust the depth provided using photometric stereo? **In particular, discuss how shadows can be handled within this framework though you do not have to account for shadows in your code.**

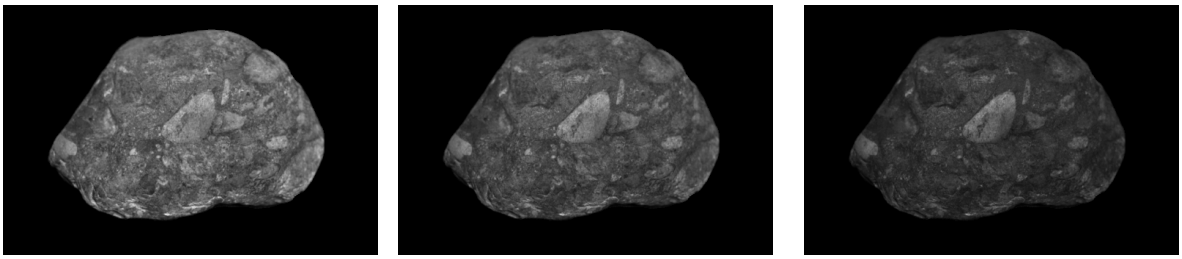
If you leverage the matrix capabilities of MATLAB it is possible to write a code that executes in less than 1 second (excluding depth map refinement). If your code is taking longer than

60 seconds to run (again, excluding depth map refinement) you likely aren't using matrices to their full potential.

The examples on the following pages are for the "Rock" dataset. I encourage you to test your code on the rock dataset and compare your output with the examples. *Submit ONLY the Buddha reconstruction. Other datasets are included for interested students but should not be included in the report.*

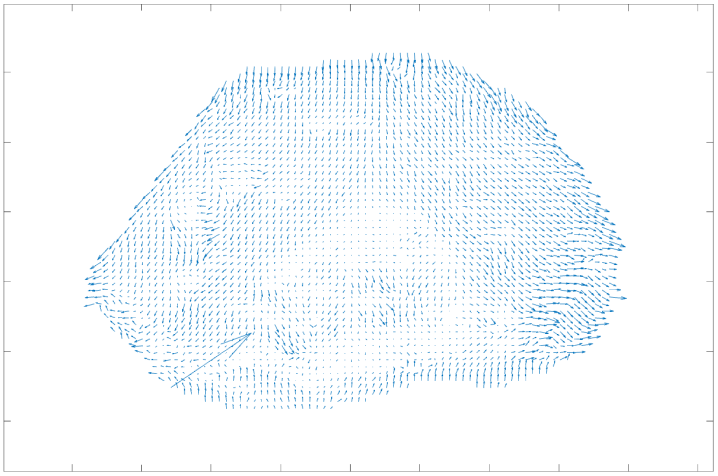
Rock dataset output

Albedo for each color channel:



(a) Red albedo (b) Green albedo (c) Blue albedo

Surface normal vectors computed from grayscale input:
`quiver(1:5:ncols,1:5:nrows,p(1:5:end,1:5:end),q(1:5:end,1:5:end),10)`-be careful about NaNs when finding p and q. You may provide an optional scale argument (shown in red) to make the arrows larger. Be sure to correctly set the axes, using `axis tight ij`. In the above code, the points were downsampled to easily distinguish neighboring vectors.

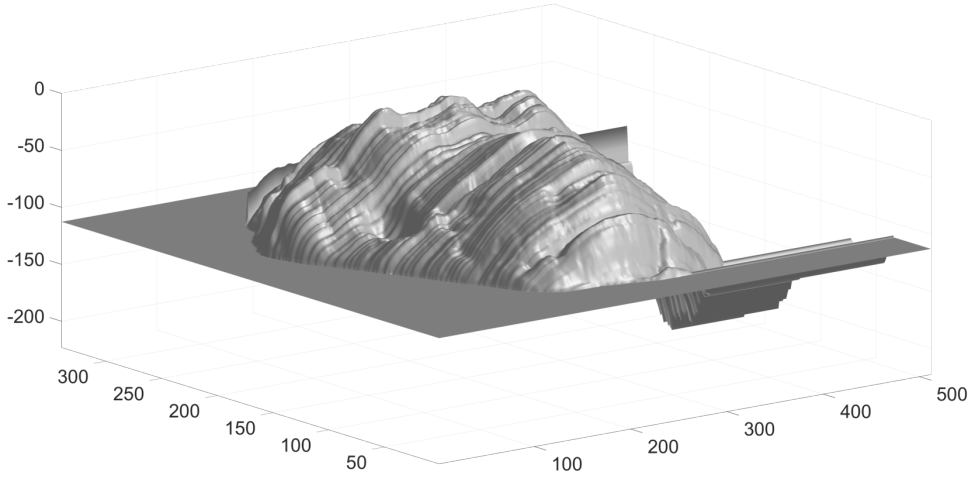


Surface normal vectors computed from grayscale images (exressed as p and q) for the rock dataset

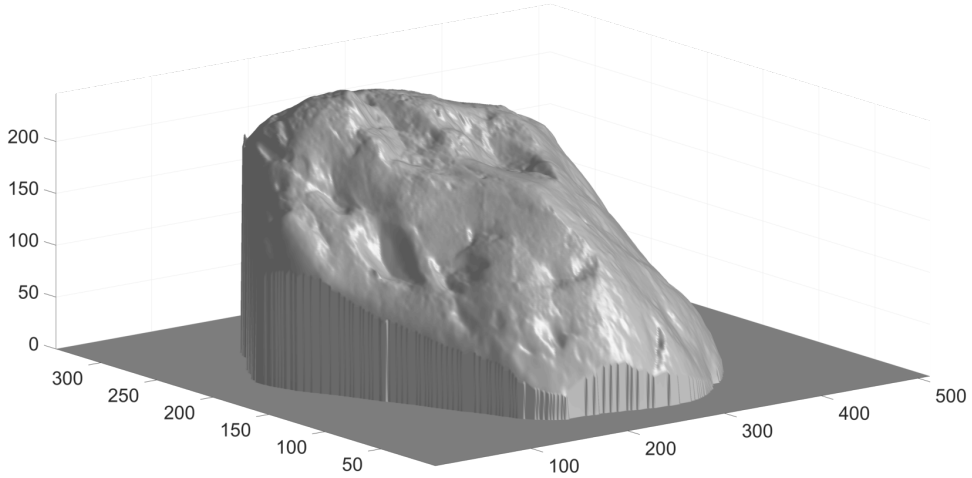
Depth map output: Use the computed surface normal vectors to find a depth map by simple integration and using the provided p-code. (Note: the provided code may take up to a few minutes to run for the rock dataset but it is faster for other images. Also, disregard any warnings about rank deficiency. You may attribute any noticeable artifacts to this.) To plot depth maps I suggest using the following command

```
surfl(±depth); shading interp; colormap gray; axis tight
```

MATLAB does not display 3D plots well, so in some cases it may be better to plot the negative depth to more easily manipulate the plot.



Depth using simple integration (Depth was negated to see this view)



Depth using Depth using provided code: depth = refineDepthMap(N,mask);

Submission Instructions

Every student must submit following 2 files:

- An organized report submitted as a PDF document. The report should describe the implementation, issues (problems encountered, surprises), and an analysis of the test results (interpretation of effects of varying parameters, different image results). Intermediate and final results must be provided.
- A ZIP file containing the necessary codes.

The heading of the PDF file should contain the assignment number and topic.

Late Submission Policy

Assignments are expected to be submitted on the due date. Each student gets a total of 3 late days that can be used however you wish. For examples, all 3 days can be used towards 1 assignment or 1 day late for 3 assignments or other combinations. Late submissions beyond that will be penalized as below:

- One day late will be penalized 25% of the credit.
- Two Days late will be penalized 50%.
- Submissions more than 2 days late will not be considered for credit.

I will be ruthless in enforcing this policy. There will be no exceptions

Collaboration Policy

I encourage collaboration both inside and outside class. You may talk to other students for general ideas and concepts but the programming must be done independently. For mid-term and final examination there will be no collaboration permitted.

Plagiarism

Plagiarism of any form will not be tolerated. You are expected to credit all sources explicitly. If you have any doubts regarding what is and is not plagiarism, talk to me.