

# **CMU Fall24 16820 Homework 5**

Patrick Chen

November 10, 2024

## Q1-a at page 3

Ans:

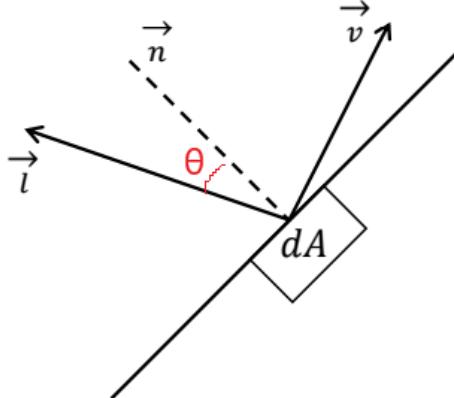
According to Lambertian's cosine law, the intensity of light observed from a Lambertian surface is proportional to the cosine of the angle  $\theta$  between the surface normal,  $\vec{n}$ , and the direction of the incoming light,  $\vec{l}$ .

$$\vec{n} \cdot \vec{l} = |\vec{n}| |\vec{l}| \cos(\theta) \quad (1)$$

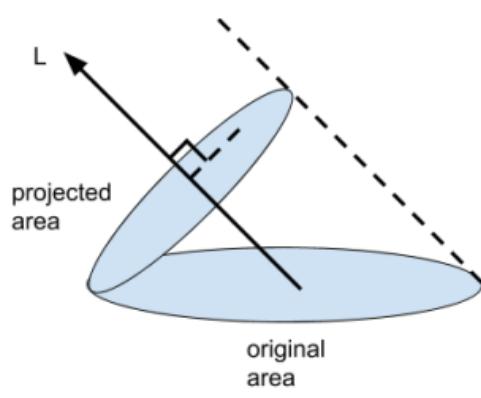
Typically in the context of n-dot-l lighting, both  $\vec{n}$  and  $\vec{l}$  are unit vectors, so we have:

$$\vec{n} \cdot \vec{l} = \cos(\theta) \quad (2)$$

As shown in Fig. 2a as below, the  $\theta$  is the dot product between  $\vec{n}$  and  $\vec{l}$ . The dot product,  $\theta$ , quantifies the amount of incident light that is effectively contributing to the brightness of the surface. As the angle increases, less light is effectively reaching the surface, as the angle decrease, more light hits the surface, reaching maximum incoming light when  $\theta$  is 0, identical to the case that  $\vec{l}$  is aligned with  $\vec{n}$ .



(a) Geometry of photometric stereo



(b) Projected area

Fig. 2a

Fig. 2b

The projected area,  $d\mathbf{A}_{\text{projected}}$ , in Fig. 2b is the original area,  $d\mathbf{A}$ , times  $\cos(\theta)$ :

$$d\mathbf{A}_{\text{projected}} = d\mathbf{A} \cdot \cos(\theta) \quad (3)$$

$$= d\mathbf{A} \cdot \vec{n} \cdot \vec{l} \quad (4)$$

$$\frac{d\mathbf{A}_{\text{projected}}}{d\mathbf{A}} = \cos(\theta) \quad (5)$$

$$= \vec{n} \cdot \vec{l} \quad (6)$$

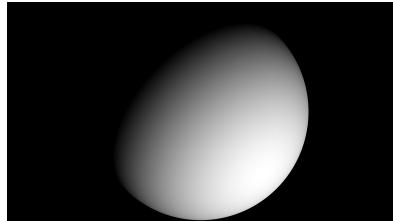
As  $\theta$  becomes larger, the effective area that light directly expose to becomes smaller, and on the opposite, as  $\theta$  becomes smaller, the effective area becomes larger, reaching maximum  $d\mathbf{A}$  as  $\theta = 0$ . Hence, the projected area comes into the equation as part of the Lambertian reflectance model through the  $\cos(\theta)$  factor.

The reason that the viewing direction does not matter is that we assume the surface is Lambertian surface, which reflects light equally in all directions,  $\vec{v}$ . Specifically, the light reflected from each point on the surface has the same intensity value no matter where the observer is located. It also means that the intensity of light that the observer can observe is based solely on the  $\vec{n} \cdot \vec{l} = \cos(\theta)$  term.

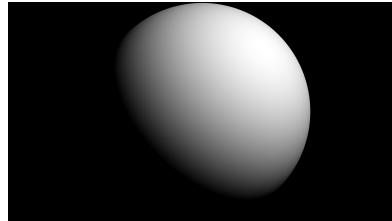
## Q1-b at page 4

Ans:

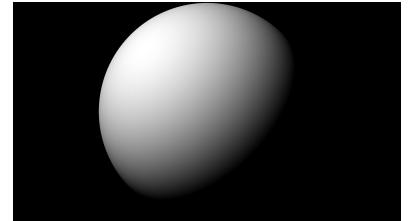
Following Figure 3 - Figure 5 shows the rendered result, and the Figure 6 shows the code snippet in q1.py.



**Figure 3:** Rendered Result with  $(1, 1, 1)/\sqrt{3}$  Light Vector



**Figure 4:** Rendered Result with  $(1, -1, 1)/\sqrt{3}$  Light Vector



**Figure 5:** Rendered Result with  $(-1, -1, 1)/\sqrt{3}$  Light Vector

```

12 def renderNDotLSphere(center, rad, light, pxSize, res):
13     """
14     Question 1 (b)
15
16     Render a hemispherical bowl with a given center and radius. Assume that
17     the hollow end of the bowl faces in the positive z direction, and the
18     camera looks towards the hollow end in the negative z direction. The
19     camera's sensor axes are aligned with the x- and y-axes.
20
21     Parameters
22     -----
23     center : numpy.ndarray
24         The center of the hemispherical bowl in an array of size (3,)
25
26     rad : float
27         The radius of the bowl
28
29     light : numpy.ndarray
30         The direction of incoming light
31
32     pxSize : float
33         Pixel size
34
35     res : numpy.ndarray
36         The resolution of the camera frame
37
38     Returns
39     -----
40     image : numpy.ndarray
41         The rendered image of the hemispherical bowl
42     """
43
44     [X, Y] = np.meshgrid(np.arange(res[0]), np.arange(res[1]))
45     X = (X - res[0] / 2) * pxSize * 1.0e-4
46     Y = (Y - res[1] / 2) * pxSize * 1.0e-4
47     Z = np.sqrt(rad**2 + 0j - X**2 - Y**2)
48     X[np.real(Z) == 0] = 0
49     Y[np.real(Z) == 0] = 0
50     Z = np.real(Z)
51     image = None
52     # Your code here
53     # Identify the visible region on the hemisphere
54     visible = Z > 0 # Only points within the bowl's radius in positive Z
55
56     # Calculate surface normals for the visible points on the hemisphere
57     Nx = X[visible] - center[0]
58     Ny = Y[visible] - center[1]
59     Nz = Z[visible] - center[2]
60     normals = np.stack((Nx, Ny, Nz), axis=-1)
61     normals = normals / np.linalg.norm(normals, axis=-1, keepdims=True)
62
63     # Calculate the dot product (n dot l) for Lambertian lighting
64     ndotl = np.dot(normals, light)
65     ndotl[ndotl < 0] = 0 # Ignore negative values (light coming from behind)
66
67     # Create the final image with lighting applied
68     image = np.zeros((res[1], res[0]))
69     image[visible] = ndotl/np.max(ndotl)*255
70
71
72     return image

```

**Figure 6:** Code Snippet

## Q1-c at page 4

Ans:

Following Figure 7 shows the code snippet of loadData() in q1.py.

```
79 def loadData(path="../data/"):
80     """
81     Question 1 (c)
82
83     Load data from the path given. The images are stored as input_n.tif
84     for n = {1...7}. The source lighting directions are stored in
85     sources.mat.
86
87     Parameters
88     -----
89     path: str
90         Path of the data directory
91
92     Returns
93     -----
94     I : numpy.ndarray
95         The 7 x P matrix of vectorized images
96
97     L : numpy.ndarray
98         The 3 x 7 matrix of lighting directions
99
100    s: tuple
101        Image shape
102
103    """
104
105    I = None
106    L = None
107    s = None
108    # Your code here
109    I = []
110
111    # Load images and convert to luminance channel
112    for n in range(1, 8):
113        image = io.imread(f"{path}/input_{n}.tif").astype(np.uint16)
114        assert image.dtype == np.uint16
115
116        xyz_image = rgb2xyz(image)
117        luminance = xyz_image[:, :, 1] # Extract Y channel (luminance)
118        #luminance = (luminance*65535).astype(np.uint16)
119        if s is None:
120            s = luminance.shape
121        I.append(luminance.flatten())
122    I = np.array(I)
123
124    # Load the sources file
125    L = np.load(f"{path}/sources.npy").T
126
127    return I, L, s
```

Figure 7: Code Snippet

## Q1-d at page 4

Ans:

$L^T$  has the shape (7, 3), representing the transposed light directions. If L has rank of 3, it means 3 of the light directions in L can span the whole 3-dimensional space. Also, B has the shape of (3, P), representing the set of pseudonormals in the images. If B has rank 3, it means 3 of the pseudonormals in B could span the whole 3-dimensional space. If both L and B have rank of 3, then  $I = L^T B$  would also have rank 3, meaning that the singular value of I should agree with rank 3.

However, we can see from below Figure 8, the singular value shows that I has rank of 7, not agreeing with rank-3 requirement. It is because that if the light directions in L or the pseudonormals in B are not linearly independent, due to the noise or measurement imprecision of light on camera intensity, then the rank of L and B might not be exactly 3, resulting in the disagreement here. The Figure 9 shows the code snippet in q1.py.

```
Singular Values of I: [79.36348099 13.16260675 9.22148403 2.414729 1.61659626 1.26289066  
0.89368302]
```

Figure 8: Singular Values of I

```
293     # Part 1(d)  
294     # Your code here  
295     u_vec, singular, vt_vec = np.linalg.svd(I, full_matrices=False)  
296     print("Singular Values of I:", singular)  
297
```

Figure 9: Code Snippet of (d)

## Q1-e at page 4

Ans:

The way I construct A and B is shown in the following derivation of estimated B:

$$\mathbf{I} = \mathbf{L}^T \mathbf{B} \quad (7)$$

$$\text{Set } \mathbf{A} = \mathbf{L}^T, \mathbf{x} = \mathbf{B}, \mathbf{y} = \mathbf{I} \quad (8)$$

$$\rightarrow \mathbf{Ax} = \mathbf{y} \quad (9)$$

$$\rightarrow \mathbf{x} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{y} \quad (10)$$

$$\rightarrow \mathbf{B} = (\mathbf{L} \mathbf{L}^T)^{-1} \mathbf{L} \mathbf{I} \quad (11)$$

Following Figure 10 shows the code snippet of (e) in q1.py.

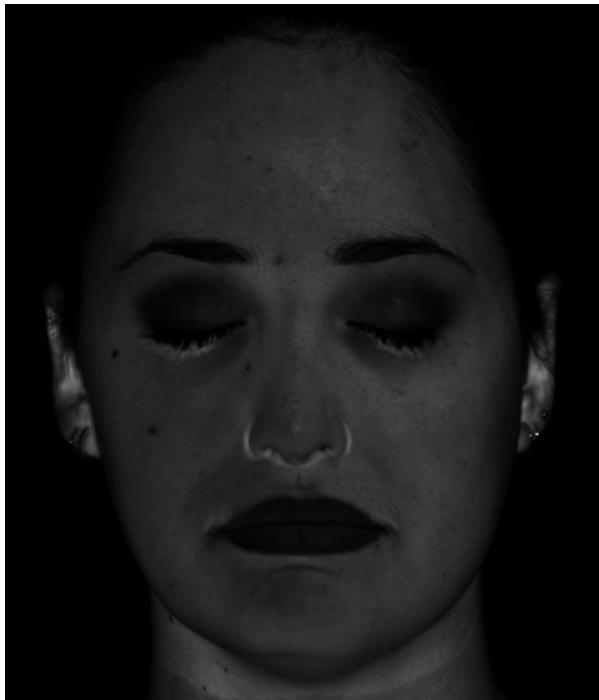
```
130 def estimatePseudonormalsCalibrated(I, L):
131     """
132     Question 1 (e)
133
134     In calibrated photometric stereo, estimate pseudonormals from the
135     light direction and image matrices
136
137     Parameters
138     -----
139     I : numpy.ndarray
140         The 7 x P array of vectorized images
141
142     L : numpy.ndarray
143         The 3 x 7 array of lighting directions
144
145     Returns
146     -----
147     B : numpy.ndarray
148         The 3 x P matrix of pesudonormals
149     """
150
151     # Your code here
152     B = np.linalg.inv(L@L.T)@(L@I)
153     return B
```

Figure 10: Code Snippet of (e)

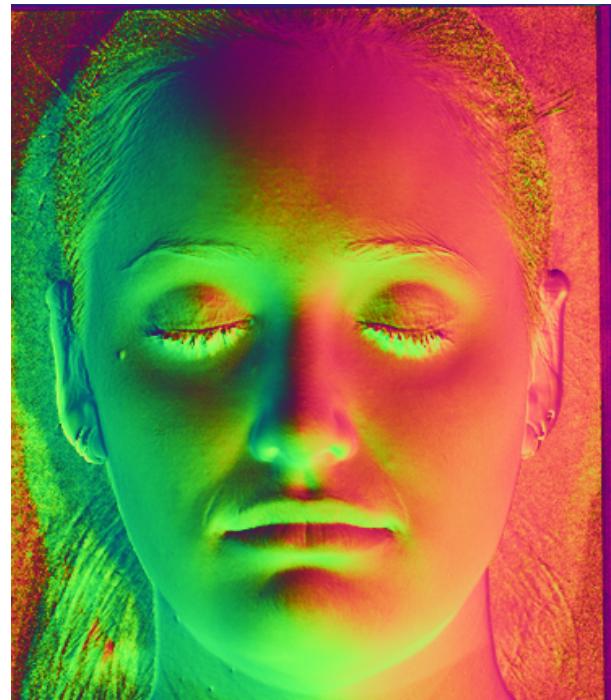
## **Q1-f at page 5**

Ans:

The Figure 11 and Figure 12 show the result of estimated albedo and normals image respectively. The albedo image shows the reflectance of the face, and normal image shows the normal directions of the face. The forehead and cheek has smooth color changing, meaning there exist smooth change of normal direction among these areas, while the area like nose and lips show sharp change of color like boundaries, indicating the normal has sharp change among these areas. As of the albedo image, the forehead, cheek, nose, and ears show high intensity, indicating these areas have high reflectance property, aligning with my expectation because the skin of these area are usually shiny and oily. The Figure 13 and Figure 14 show the code snippets in q1.py.



**Figure 11:** Estimated albedo



**Figure 12:** Estimated normals

```

156 def estimateAlbedosNormals(B):
157     """
158     Question 1 (e)
159
160     From the estimated pseudonormals, estimate the albedos and normals
161
162     Parameters
163     -----
164     B : numpy.ndarray
165         The 3 x P matrix of estimated pseudonormals
166
167     Returns
168     -----
169     albedos : numpy.ndarray
170         The vector of albedos
171
172     normals : numpy.ndarray
173         The 3 x P matrix of normals
174     """
175
176     eps = 1e-6
177
178     albedos = np.linalg.norm(B, axis=0) # (P,)
179     normals = B/(albedos+eps) #eps: avoid dividing by zero
180     # Your code here
181     return albedos, normals

```

**Figure 13:** Code Snippet of  
estimateAlbedosNormals()

```

183 def displayAlbedosNormals(albedos, normals, s):
184     """
185     Question 1 (f, g)
186
187     From the estimated pseudonormals, display the albedo and normal maps
188
189     Please make sure to use the `coolwarm` colormap for the albedo image
190     and the `rainbow` colormap for the normals.
191
192     Parameters
193     -----
194     albedos : numpy.ndarray
195         The vector of albedos
196
197     normals : numpy.ndarray
198         The 3 x P matrix of normals
199
200     s : tuple
201         Image shape
202
203     Returns
204     -----
205     albedoIm : numpy.ndarray
206         Albedo image of shape s
207
208     normalIm : numpy.ndarray
209         Normals reshaped as an s x 3 image
210
211     """
212
213     albedoIm = albedos.reshape(s)
214     normalIm = normals.T.reshape((s[0], s[1], 3)) # (3, P) -> (P, 3) -> (h, w, 3)
215     normalIm = (normalIm + 1.0)/2.0 # from value range [-1, 1] to [0, 1]
216     # Your code here
217     return albedoIm, normalIm

```

**Figure 14:** Code Snippet of  
displayAlbedosNormals()

## Q1-g at page 5

Ans:

The surface normal at point  $(x, y)$  is  $\mathbf{n} = (n_1, n_2, n_3)$ , and 3D depth map function at this point is  $z = f(x, y)$ . We can have the tangent along  $x$  direction as:  $(1, 0, \frac{\partial f(x,y)}{\partial x})$ , and tangent along  $y$  direction as  $(0, 1, \frac{\partial f(x,y)}{\partial y})$ . Then the tangent  $\mathbf{x}$  has the following relationship with normal  $\mathbf{n}$  vector:

$$\mathbf{tangent}_x \cdot \mathbf{n} = (1, 0, \frac{\partial f(x,y)}{\partial x}) \cdot (n_1, n_2, n_3) = 0 \quad (12)$$

$$\rightarrow n_1 + n_3 \frac{\partial f(x,y)}{\partial x} = 0 \quad (13)$$

$$\rightarrow -\frac{n_1}{n_3} = \frac{\partial f(x,y)}{\partial x} = f_x \quad (14)$$

We can do the similar thing on the tangent  $\mathbf{y}$  vector with normal  $\mathbf{n}$  vector:

$$\mathbf{tangent}_y \cdot \mathbf{n} = (0, 1, \frac{\partial f(x,y)}{\partial y}) \cdot (n_1, n_2, n_3) = 0 \quad (15)$$

$$\rightarrow n_2 + n_3 \frac{\partial f(x,y)}{\partial y} = 0 \quad (16)$$

$$\rightarrow -\frac{n_2}{n_3} = \frac{\partial f(x,y)}{\partial y} = f_y \quad (17)$$

As one can see, the equation holds as the statement in Q1-g in hw5.pdf.

## Q1-h at page 6

Ans:

We have the following two gradient calculations:

$$g_x(x_i, y_j) = g(x_{i+1}, y_j) - g(x_i, y_j) \quad (18)$$

$$g_y(x_i, y_j) = g(x_i, y_{j+1}) - g(x_i, y_j) \quad (19)$$

When using equation (18), the gradient of x,  $g_x$ , is:

$$g_x = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

When using equation (19), the gradient of y,  $g_y$ , is:

$$g_y = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \end{bmatrix}$$

Method 1. Given  $g(0, 0) = 1$ , use  $g_x$  to construct the first row of  $g$ , then use  $g_y$  to construct the rest of  $g$ :

Step1. Use  $g_x$  to construct the first row:  $g(0, j) = g(0, j-1) + g_x(0, j-1)$ ,  $1 \leq j \leq 3$ , the order is from left to right.

$$g = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Step2. Use  $g_y$  for the rest:  $g(i, j) = g(i-1, j) + g_y(i-1, j)$ ,  $1 \leq i \leq 3$ ,  $0 \leq j \leq 3$ , the order is from top to down. The reconstructed  $g$  is:

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

Method 2. Given  $g(0, 0) = 1$ , use  $g_y$  to construct the first column of  $g$ , then use  $g_x$  to construct the rest of  $g$ :

Step1. Use  $g_y$  to construct the first column:  $g(i, 0) = g(i-1, 0) + g_y(i-1, 0)$ ,  $1 \leq i \leq 3$ , the order is from top to down.

$$g = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 \\ 9 & 0 & 0 & 0 \\ 13 & 0 & 0 & 0 \end{bmatrix}$$

Step2. Use  $g_x$  for the rest:  $g(i, j) = g(i, j-1) + g_x(i, j-1)$ ,  $0 \leq i \leq 3$ ,  $1 \leq j \leq 3$ , the order is from left to right. The reconstructed  $g$  is:

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

As we can see above, both procedures produce the same reconstruction of  $g$ .

The integrability implies that:

$$\frac{\partial g_x}{\partial y} = \frac{\partial g_y}{\partial x} \quad (20)$$

So, we can modify the  $g_x$  and  $g_y$  as below to make them non-integrable:

$$g_x = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 30 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$g_y = \begin{bmatrix} 4 & 4 & 7 & 4 \\ 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \end{bmatrix}$$

Then the reconstructed  $g$  using Method 1 above would be:

$$g = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 10 & 8 \\ 9 & 10 & 14 & 12 \\ 13 & 14 & 18 & 16 \end{bmatrix}$$

The reconstructed  $g$  using Method 2 above would be:

$$g = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 10 & 8 \\ 9 & 10 & 40 & 41 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

As we can see above, the reconstructed  $g$  are different in these two procedures, thus the  $g_x$  and  $g_y$  are now non-integrable. The reasons that the gradients estimated in the way of  $(g)$  may be non-integrable include the measurement noise in  $g_x$  or  $g_y$ , wrong assumptions on surface reflectance, or the existence of non-smooth surface.

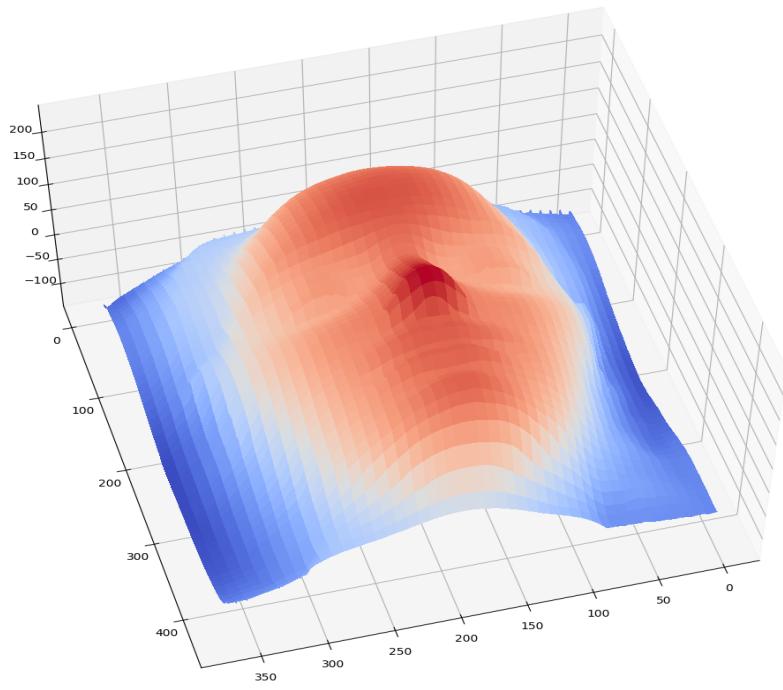
## Q1-i at page 6

Ans:

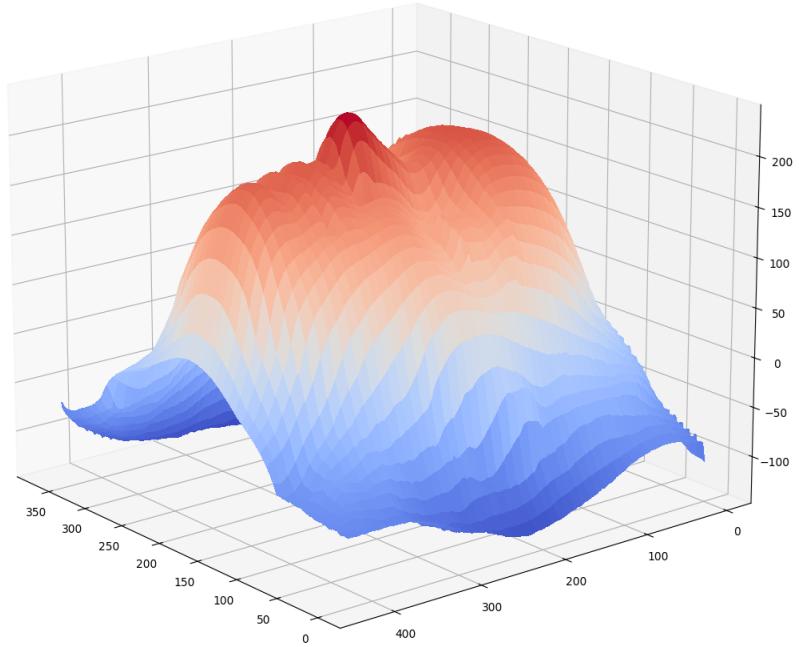
The following Figure 16 - Figure 18 show the results fo estimated shape in three different viewpoints. The code snippet is shown in Figure 15.

```
220 def estimateShape(normals, s):
221     """
222     Question 1 (j)
223
224     Integrate the estimated normals to get an estimate of the depth map
225     of the surface.
226
227     Parameters
228     -----
229     normals : numpy.ndarray
230         The 3 x P matrix of normals
231
232     s : tuple
233         Image shape
234
235     Returns
236     -----
237     surface: numpy.ndarray
238         The image, of size s, of estimated depths at each point
239
240     """
241
242     surface = None
243     # Your code here
244     # Reshape normals into the image shape
245     n1 = normals[0, :].reshape(s) # X-component of normals, (h, w, 1)
246     n2 = normals[1, :].reshape(s) # Y-component of normals, (h, w, 1)
247     n3 = normals[2, :].reshape(s) # Z-component of normals, (h, w, 1)
248
249     # Compute gradients f_x and f_y
250     fx = -n1 / n3
251     fy = -n2 / n3
252
253     # Use utils.integrateFrankot for integration
254     surface = integrateFrankot(fx, fy)
255     return surface
```

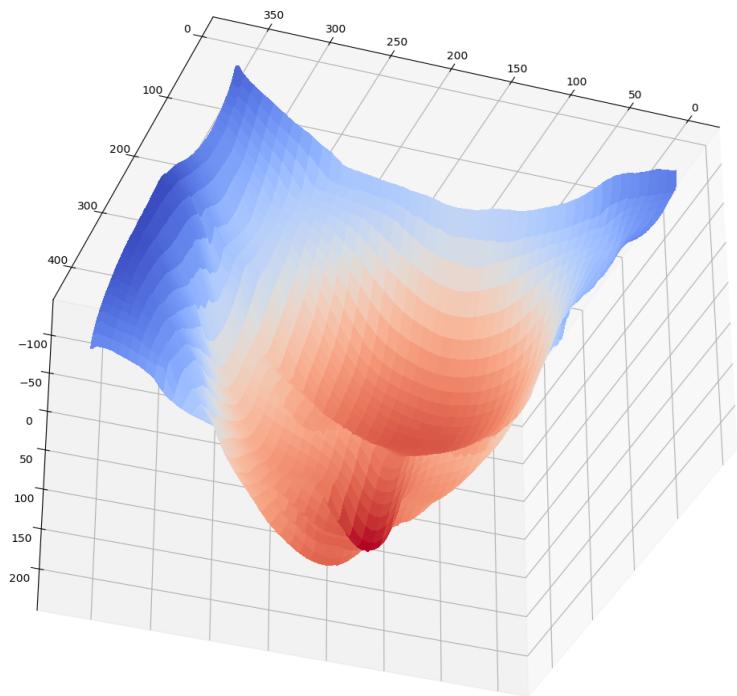
Figure 15: Code Snippet



**Figure 16:** Result of Shape Estimation in View 1



**Figure 17:** Result of Shape Estimation in View 2



**Figure 18:** Result of Shape Estimation in View 3

## Q2-a at page 7

Ans:

Since  $\mathbf{I} = \mathbf{L}^T \mathbf{B}$ , and we do not know  $\mathbf{L}^T$  and  $\mathbf{B}$ , we can directly do SVD on  $\mathbf{I}$ . This gives:

$$\mathbf{I} = \mathbf{U} \Sigma \mathbf{V}^T \quad (21)$$

Since  $\mathbf{I}$  is a  $7 \times P$  matrix, so the  $\mathbf{U}$  would be a  $7 \times 7$  matrix,  $\Sigma$  would be a  $7 \times P$  diagonal matrix, and  $\mathbf{V}^T$  would be a  $P \times P$  matrix, where  $P$  is the number of pixels.

To approximate the rank 3 of  $\mathbf{I}$ , we first set all singular values except the top 3 from  $\Sigma$  to 0, and we get the diagonal  $3 \times 3$  matrix  $\hat{\Sigma}$ . Secondly, we choose the first 3 columns of  $\mathbf{U}$  to get  $7 \times 3$  matrix:  $\hat{\mathbf{U}}$ , choose the first 3 columns of  $\mathbf{V}$  to get a  $P \times 3$  matrix:  $\hat{\mathbf{V}}$ . Thus, we can get  $\hat{\mathbf{I}}$  as below:

$$\hat{\mathbf{I}} = \hat{\mathbf{U}} \hat{\Sigma} \hat{\mathbf{V}}^T \quad (22)$$

Where  $\hat{\mathbf{I}}$  is the reconstructed rank 3 matrix with shape  $7 \times P$ . Accordingly, we can use the result of equation (22) to approximate  $\hat{\mathbf{L}}^T$  and  $\hat{\mathbf{B}}$ :

$$\hat{\mathbf{I}} = \hat{\mathbf{U}} \hat{\Sigma}^{1/2} \hat{\Sigma}^{1/2} \hat{\mathbf{V}}^T \quad (23)$$

$$= \hat{\mathbf{L}}^T \hat{\mathbf{B}} \quad (24)$$

Where  $\hat{\mathbf{L}}^T = \hat{\mathbf{U}} \hat{\Sigma}^{1/2}$  with shape  $7 \times 3$ , and  $\hat{\mathbf{B}} = \hat{\Sigma}^{1/2} \hat{\mathbf{V}}^T$  with shape  $3 \times P$ . So this is how we can use SVD to approximate  $\hat{\mathbf{L}}^T$  and  $\hat{\mathbf{B}}$  and construct a factorization of  $\hat{\mathbf{I}}$  following the required constraints of rank 3.

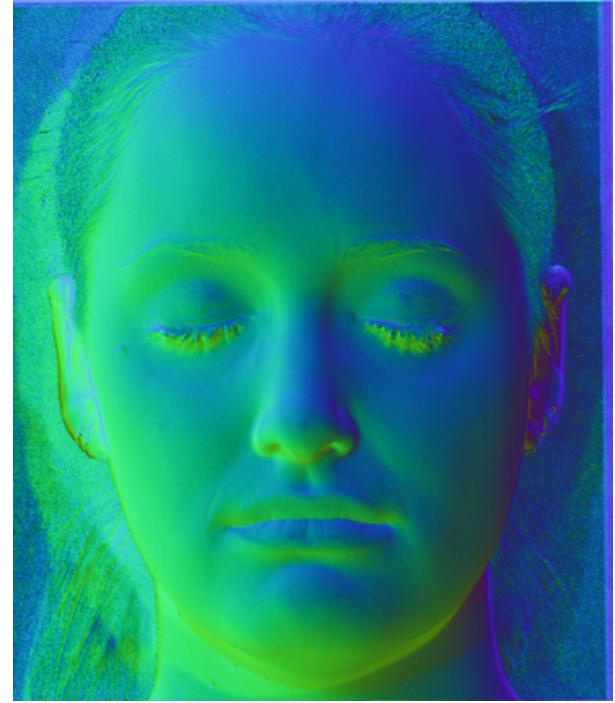
## Q2-b at page 7

Ans:

Below Figure 19 and Figure 20 shows the results generated by the method mentioned in Q2-a. Figure 21 and Figure 22 show the code snippets of this question in q2.py.



**Figure 19:** Estimated albedo



**Figure 20:** Estimated normals

```

19 def estimatePseudonormalsUncalibrated(I):
20     """
21     Question 2 (b)
22
23     Estimate pseudonormals without the help of light source directions.
24
25     Parameters
26     -----
27     I : numpy.ndarray
28         The 7 x P matrix of loaded images
29
30     Returns
31     -----
32     B : numpy.ndarray
33         The 3 x P matrix of pseudonormals
34
35     L : numpy.ndarray
36         The 3 x 7 array of lighting directions
37
38     """
39
40     U, Sigma, Vt = np.linalg.svd(I, full_matrices=False)
41     Sigma_hat = np.diag(Sigma[:3]) # shape(3, 3)
42     U_hat = U[:, :3] #shape(7, 3)
43     V_hat = Vt[:3, :] #shape(3, P)
44
45     L_trans = U_hat @ np.sqrt(Sigma_hat) #shape(7, 3)
46     B = np.sqrt(Sigma_hat) @ V_hat #shape(3, P)
47
48     # Your code here
49     return B, L_trans.T

```

**Figure 21:** Code Snippet 1

```

92 if __name__ == "__main__":
93     I, s = loadData("../data/")
94     formatted_print('I', I)
95
96     ...
97     # test code
98     print(f"l = {L}")
99     print(f"I[0].dtype = {I[0].dtype}")
100
101 num_images = I.shape[0]
102 for i in range(num_images):
103     image = I[i].reshape(s) # Reshape to original shape
104     plt.imshow(image, cmap='gray')
105     plt.title(f"Image {i+1}")
106     plt.axis('off')
107     plt.show()
108
109
110 # Part 2 (b)
111 # Your code here
112 B_hat, L_hat = estimatePseudonormalsUncalibrated(I)
113 albedos, normals = estimateAlbedosNormals(B_hat)
114 albedoIm, normalIm = displayAlbedosNormals(albedos, normals, s)
115 plt.imsave("2b-a.png", albedoIm, cmap="gray")
116 plt.imsave("2b-b.png", normalIm, cmap="rainbow")
117 formatted_print('L_hat', L_hat)

```

**Figure 22:** Code Snippet 2

## Q2-c at page 8

Ans:

Below Figure 23 shows the original  $\mathbf{L}$  and the reconstructed  $\hat{\mathbf{L}} = \mathbf{L\_hat}$ . As we can see,  $\mathbf{L}$  and  $\hat{\mathbf{L}}$  are not similar, but quite different. There are many ways to change the procedure in (a) to change  $\hat{\mathbf{L}}$  and  $\hat{\mathbf{B}}$ , but keeps the rendered image using them the same. For example, we can set  $\hat{\mathbf{L}}^T = \hat{\mathbf{U}}(\frac{1}{2}\hat{\Sigma}^{1/2})$ ,  $\hat{\mathbf{B}} = (2\hat{\Sigma}^{1/2})\hat{\mathbf{V}}^T$ , and  $\hat{\mathbf{L}}^T\hat{\mathbf{B}}$ , the rendered image, is still the same.

```
● (hw5_env) root@docker-desktop:~/CMU_16820_Advanced_Computer_Vision/HW5_my_work/src# python3 ./q2.py
L =
[[-0.1418 0.1215 -0.0690 0.0670 -0.1627 0.0000 0.1478]
 [-0.1804 -0.2026 -0.0345 -0.0402 0.1220 0.1194 0.1209]
 [-0.9267 -0.9717 -0.8380 -0.9772 -0.9790 -0.9648 -0.9713]]

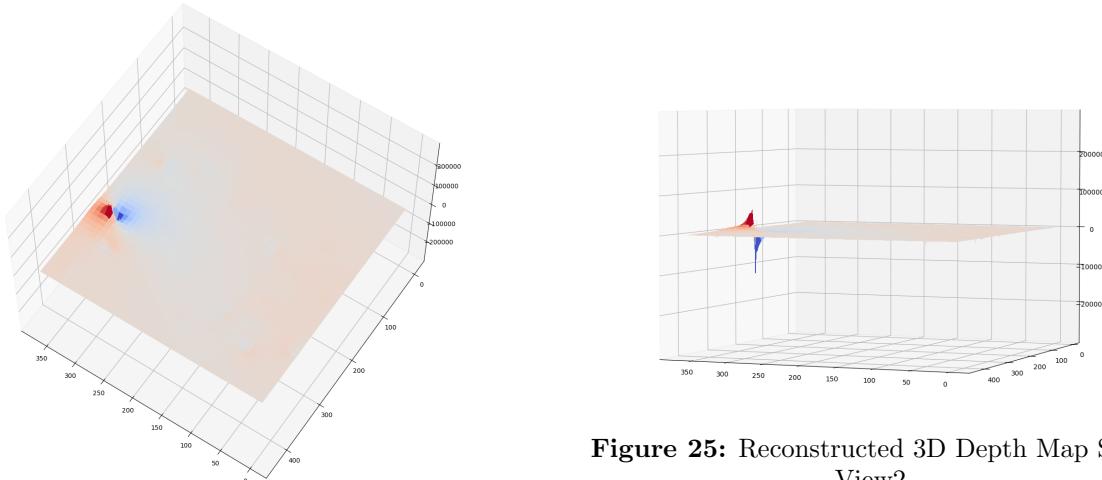
L_hat =
[[-2.9927 -3.8700 -2.4080 -3.7450 -3.5914 -3.3867 -3.3525]
 [0.9478 -2.3171 0.4991 -0.6260 2.3257 0.4661 -0.7927]
 [1.8793 1.0146 0.4294 -0.0173 -0.3108 -0.9127 -1.8830]]
```

Figure 23: Comparison of  $\mathbf{L}$  and  $\hat{\mathbf{L}}$

## Q2-d at page 8

Ans:

Figure 24 and Figure 25 show different views of the reconstructed 3D depth map shape using Franot-Chellappa algorithm. It does not look like a face.



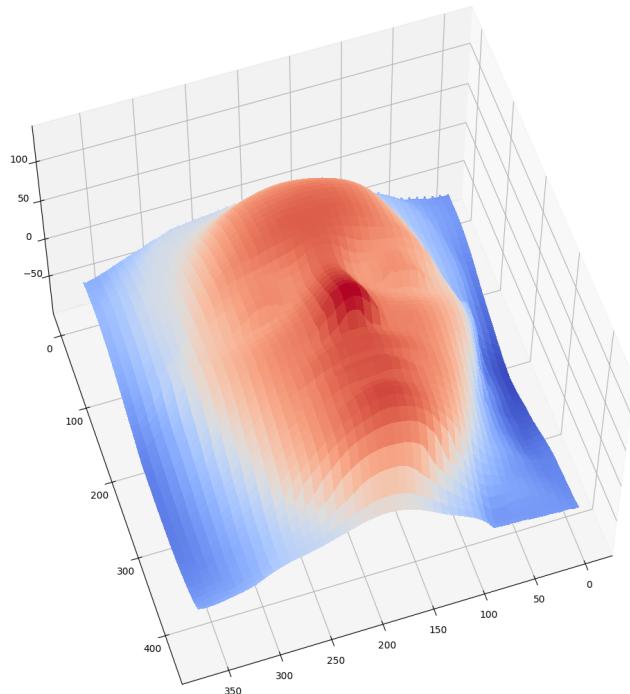
**Figure 25:** Reconstructed 3D Depth Map Shape  
View2

**Figure 24:** Reconstructed 3D Depth Map Shape  
View1

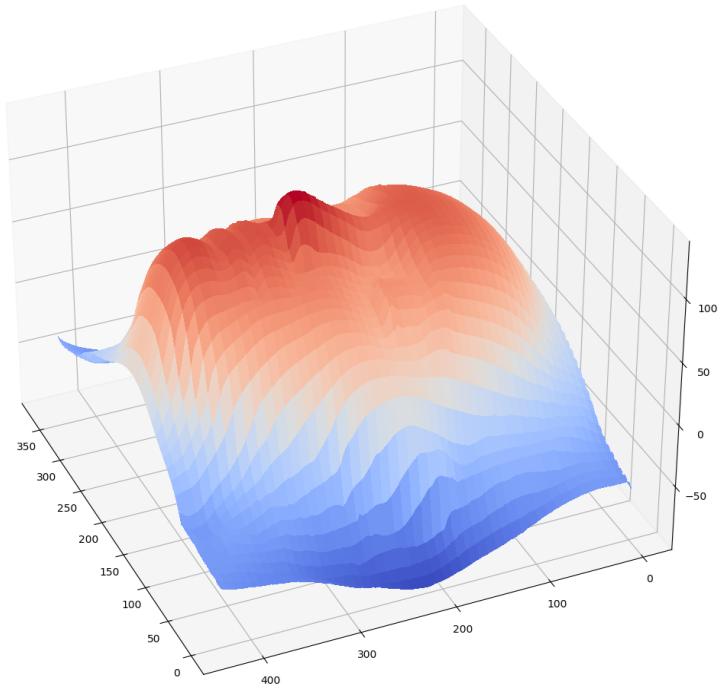
## Q2-e at page 8

Ans:

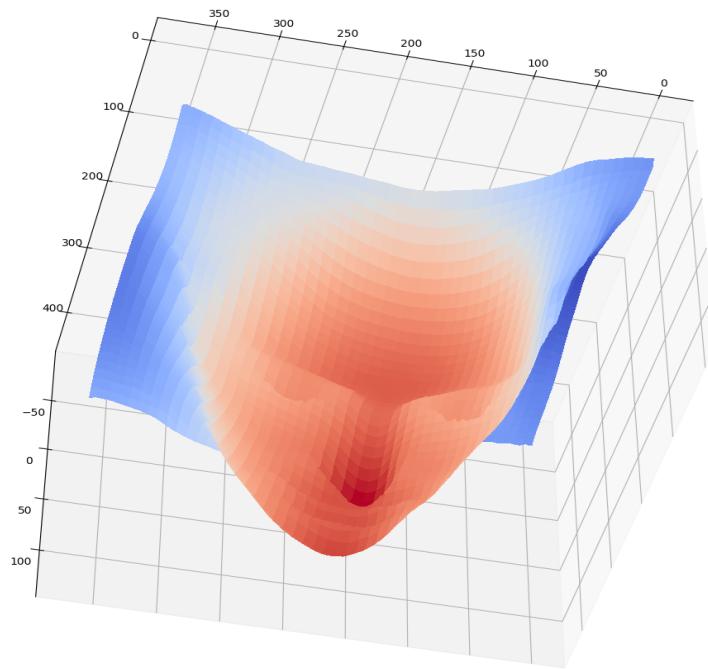
The following Figure 26 - Figure 28 show the results of estimated shape in three different viewpoints using pseudonormals and enforceIntegrability() in utils.py. The surface does look like the one output by calibrated photometric stereo in Q1-i question above, but it seems to be a little more flattened, as one can see the height range of the results here are only in the range of [-50, 100], while the results in Q1-i are in the range of [-100, 200].



**Figure 26:** Result of Shape Estimation in View 1



**Figure 27:** Result of Shape Estimation in View 2



**Figure 28:** Result of Shape Estimation in View 3

## Q2-f at page 9

Ans:

Let's discuss how the value of  $\mu \nu \lambda$  affect the reconstructed surface:

When  $\mu = [-0.1, -1, -10]$ , and  $\nu = 1, \lambda = 1$ :

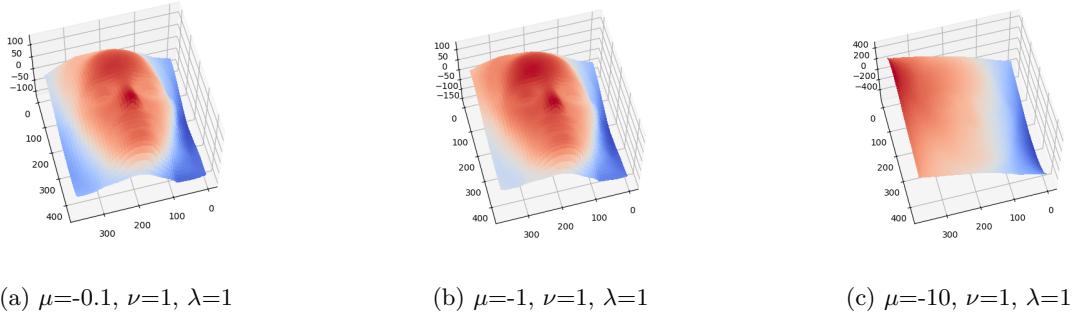


Figure 29: Impact of Negative  $\mu$  Values on Surface

When  $\mu = [0.1, 1, 10]$ , and  $\nu = 1, \lambda = 1$ :

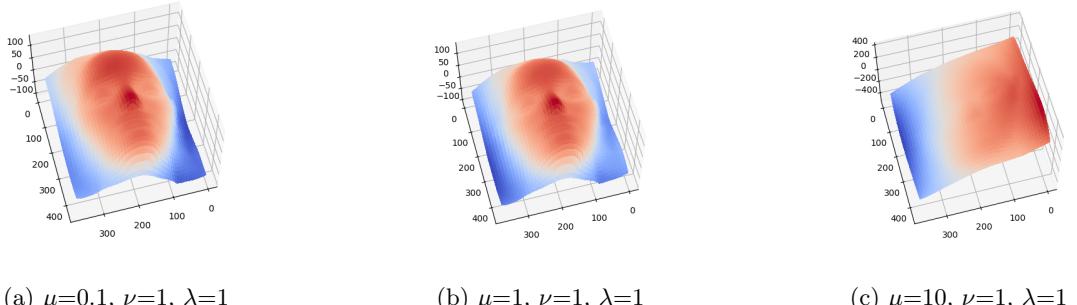


Figure 30: Impact of Positive  $\mu$  Values on Surface

When  $\nu = [-0.1, -1, -10]$ , and  $\mu = 1, \lambda = 1$ :

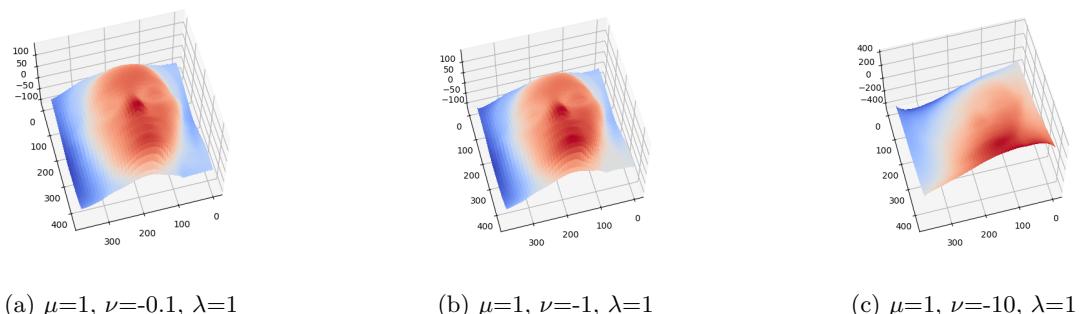


Figure 31: Impact of Negative  $\nu$  Values on Surface

When  $\nu = [0.1, 1, 10]$ , and  $\mu = 1, \lambda = 1$ :

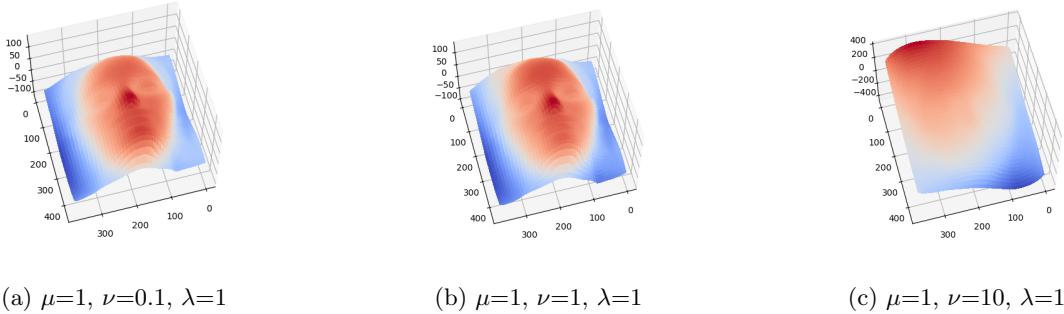


Figure 32: Impact of Positive  $\nu$  Values on Surface

When  $\lambda = [-0.1, -1, -10]$ , and  $\mu = 1, \nu = 1$ :

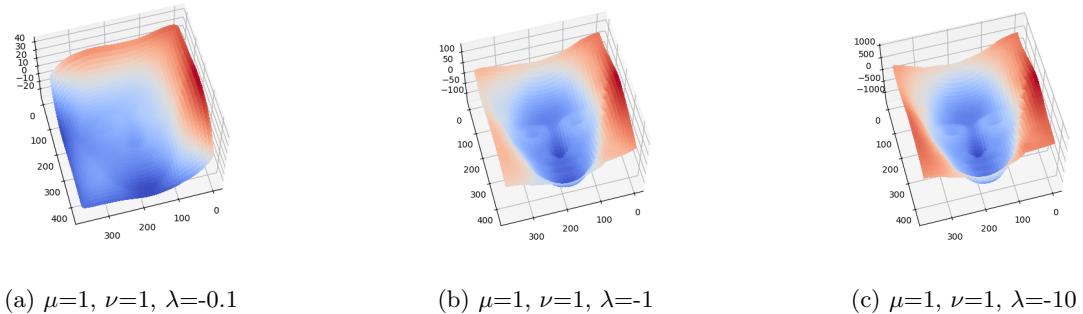


Figure 33: Impact of Negative  $\lambda$  Values on Surface

When  $\lambda = [0.1, 1, 10]$ , and  $\mu = 1, \nu = 1$ :

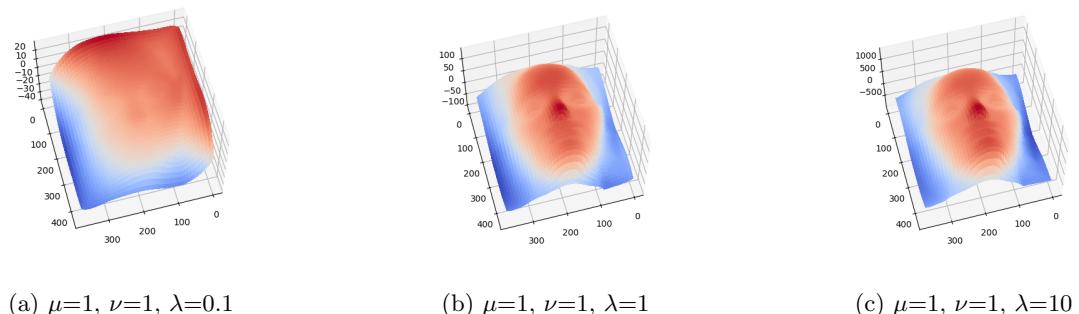


Figure 34: Impact of Positive  $\lambda$  Values on Surface

As we can see above, when  $\mu$  changes from small values to large values, the surface gets more flattened, while the positive and negative values affect the surface to be more flattened toward opposite directions.

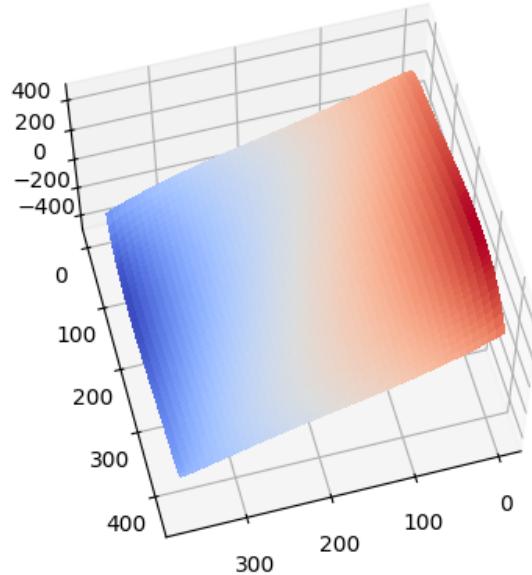
The  $\nu$  changes in similar way, as it changes from small values to large values, the surface gets more flattened, while the positive and negative values affect the surface to be more flattened in opposite directions. The  $\lambda$  changes in another particular way: as its value change from small to large, the curve of the surface becomes sharper, while the positive and negative values affect the surface to be sharper in opposite directions.

The reason why bas-relief ambiguity is named so is that the GBR transform preserves the dot product of surface normal and light direction while producing different surface shape by tilting and scaling curved depth (the x and y components of  $\mathbf{B}$  are not changed, only z component becomes the combination of x, y and z components after GBR transform), which can be seen from above Figure 29 to Figure 34. That is to say,  $\mathbf{I}$ , which is  $\mathbf{L}^T \mathbf{B}$ , would be equal to  $\mathbf{I}' = \mathbf{L}^T \mathbf{B}'$ , where  $\mathbf{B}' = \mathbf{G}^{-\mathbf{T}} \mathbf{B}$ , because the appearance  $\mathbf{I}$  only depends on the  $\cos \theta$  between  $\mathbf{L}$  and  $\mathbf{B}$ . As a result, we cannot reconstruct a unique shape based only on appearance  $\mathbf{I}$ .

## Q2-g at page 9

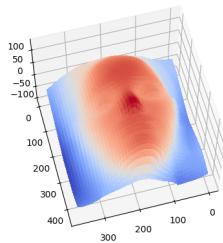
Ans:

As we can see the trend in Q2-f, the larger absolute value of  $\mu$  or  $\nu$ , not both, the more flattened the reconstructed shape is. Besides, the smaller absolute value of  $\lambda$ , the more flattened the reconstructed shape is. Thus, we can use the setting of  $\mu = 10$ ,  $\nu = 1$ , and  $\lambda=0.001$  to generate the flattened shape as below Figure 35.

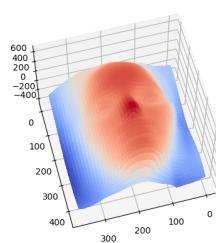


**Figure 35:** Result of Flattened Shape

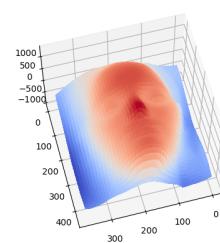
One of the interesting fact is that when we set both  $\mu$  and  $\nu$  to be large, the reconstructed shape would not be flattened:



(a)  $\mu=1, \nu=1$



(b)  $\mu=5, \nu=5$



(c)  $\mu=10, \nu=10$

## **Q2-h at page 9**

Ans:

Acquiring more pictures does not help resolve the ambiguity, because with only known appearance  $\mathbf{I}$ , we can only calculate the reconstructed  $\mathbf{B}$  up to a matrix of the form in Eq. 2 above, the GBR transform, generating the same appearance as original  $\mathbf{B}$  does. To resolve the ambiguity, we may need the light direction  $\mathbf{L}$ , the depth information, or other features like the geometry of the object.

## Collaborations

Ans:

Though I do not have collaborators, I found the following websites helpful on understanding the concepts in this homework.

1. [https://www.ri.cmu.edu/pub\\_files/pub3/baker\\_simon\\_2003\\_3/baker\\_simon\\_2003\\_3.pdf](https://www.ri.cmu.edu/pub_files/pub3/baker_simon_2003_3/baker_simon_2003_3.pdf).
2. [https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/classes/cs294-appearance\\_models/sp2001/cache/belhumeur99.pdf](https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/classes/cs294-appearance_models/sp2001/cache/belhumeur99.pdf)
3. [https://matplotlib.org/stable/api/\\_as\\_gen/matplotlib.pyplot.imsave.html](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imsave.html)
4. [https://en.wikipedia.org/wiki/Singular\\_value\\_decomposition](https://en.wikipedia.org/wiki/Singular_value_decomposition)
5. [http://graphics.cs.cmu.edu/courses/15-463/2019\\_fall/lectures/lecture16.pdf](http://graphics.cs.cmu.edu/courses/15-463/2019_fall/lectures/lecture16.pdf)