Computer Vision, 16720A - Homework #6

Neeraj Basu
neerajb@andrew.cmu.edu

December 3, 2020

1a

Given two vectors whose bases are at the same point, we can use the dot product to determine how much of one vector is projected onto the other. In this instance, we are calculating the length of the projection of n vector onto the l vector. This projection can alternatively be thought of as the cos of θ as depicted in Figure 1 (Lambert's Cosine Law).

I include this $cos(\theta)$ representation to show how the viewing direction of the projection of n onto l is angle invariant. The area of the projected area is conical in nature due to the fact that we are dealing with lambertian surfaces implying light is reflected off in all directions uniformly. The cross section of this conical nature is what gives rise to the projected area as seen in Fig 2a of the writeup.

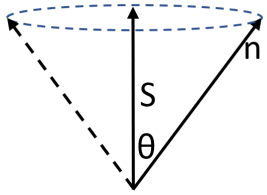

Figure 1: Viewing Angle Invariant

1b

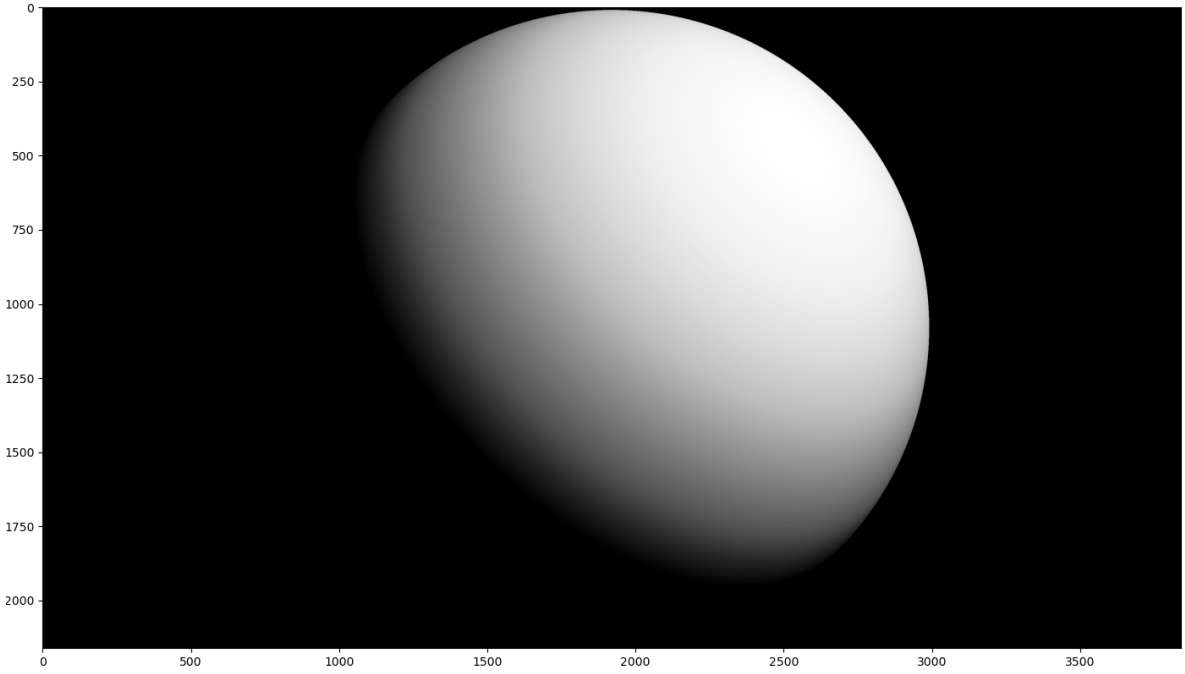

Figure 2: S = [1,1,1]/$\sqrt{3}$

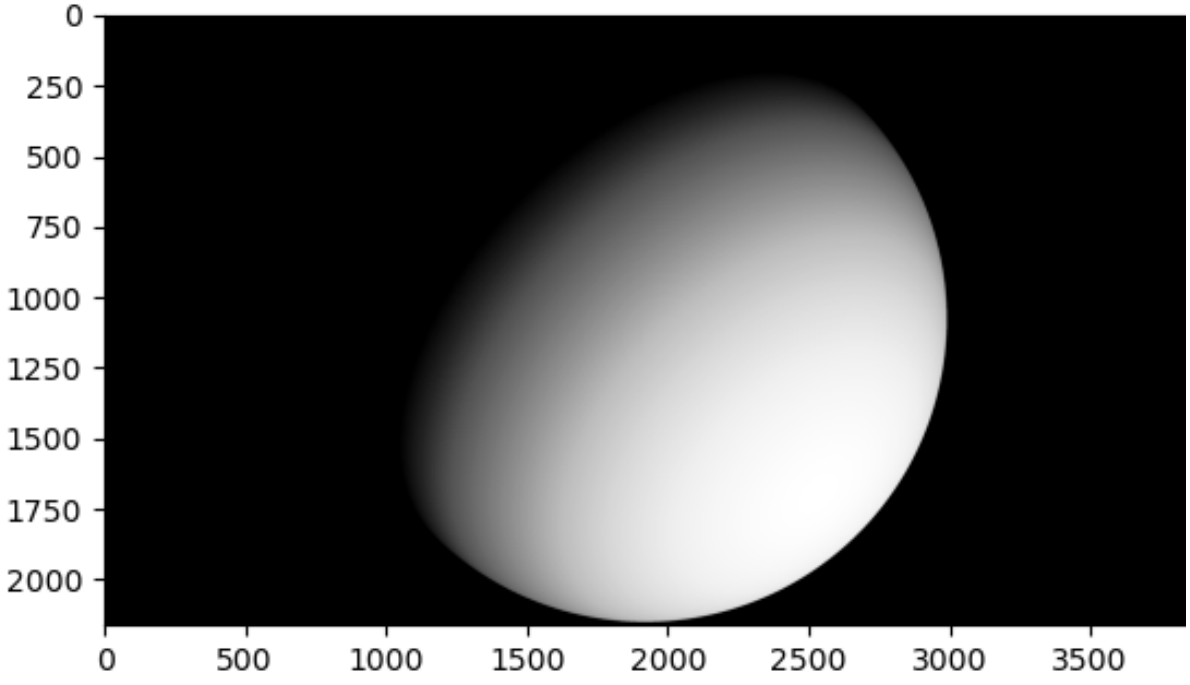

Figure 3: S = [1,-1,1]/$\sqrt{3}$

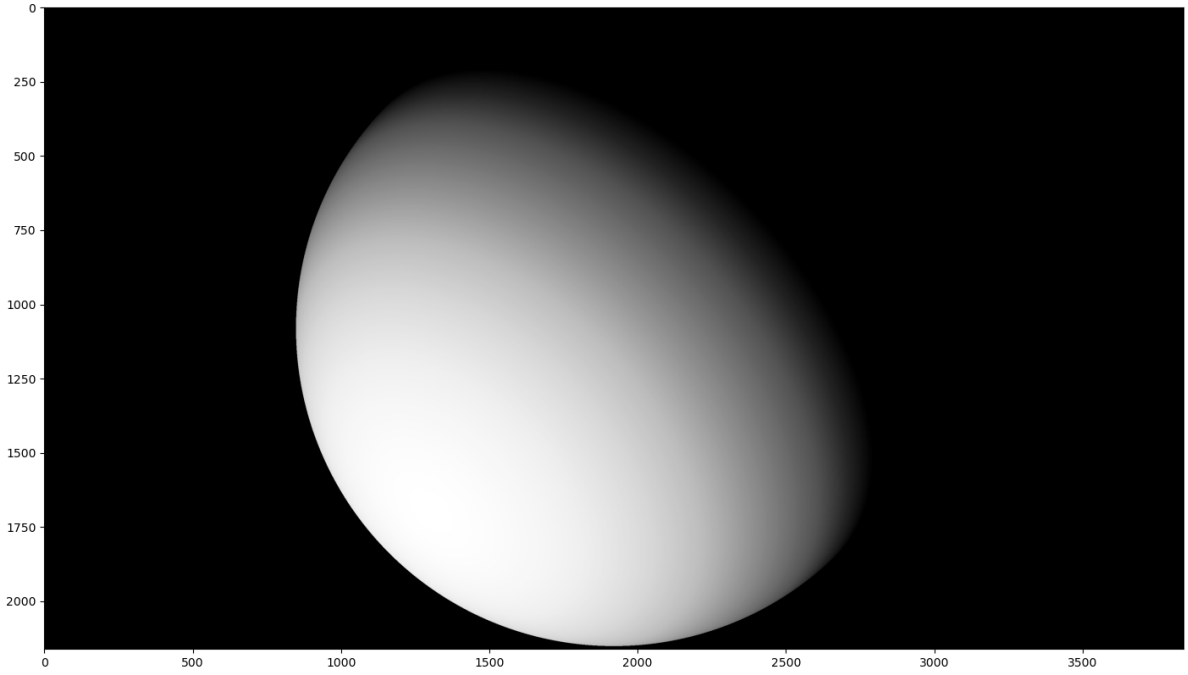

Figure 4: S = [1,-1,-1]/$\sqrt{3}$

1c

```
def loadData(path = "../data/"):

    n = 8
    data_arr = []
    for i in range(1,8):
        data = imread(path + "input_" + str(i) + ".tif").astype(np.uint16)
        data_xyz = rgb2xyz(data)
        data_arr.append(data_xyz)
    data_arr = np.array(data_arr)

    I = []
    for arr in data_arr:
        I.append(arr[:,:,1].flatten())
    I = np.array(I)

    L = np.load(path + "sources.npy").T

    s = data_arr[0,:,:,0].shape

    return I, L, s
```

Checking sizes of I, L and s:

```
    print(I.shape)
    (7, 159039)
    print(L.shape)
    (3, 7)
    print(s)
    (431, 369)
```

1d

Generally speaking, normals in 3 space are 3D vectors composed of x,y,z components. However in our case where we are calculating unit normals, our z component is normalized to 1. Therefore from pixel to pixel, there is no third dimension in the z-axis and the unit normal vector is 2D. However, we still need a method in which we can display reflectively within a 2D image. We can accomplish this by calculating the albedo value at each pixel. Therefore our third dimension is the albedo value. When combining the unit normals with our albedo values, we end with a psuedonormal vector which is 3D and rank 3.

When preforming an SVD on I, I get the following vector as the singular values.

$$\begin{bmatrix}79.36348099 & 13.16260675 & 9.22148403 & 2.414729 & 1.61659626 & 1.26289066 & 0.89368302\end{bmatrix}$$

In regards to Python's matrix rank calculations, this vector is rank 7 which is different than expected. We can see the first three columns have a higher magnitude than the following four. These top three singular values are representative of the highest contributing dimensions while the others are a result of noise and the fact that SVD is an approximation. Because this 7xP matrix was composed over multiple images each with many pixels, there is bound to be observational noise which effects the SVD estimation of I.

1e

Given the equation:

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

Our objective is to solve for the pseudonormals, B. We can solve for B with the following steps. First multiply both sides of the equation by L.

$$LI = LL^T B \tag{2}$$

Isolating B:

$$(LL^T)^{-1} LI = B \tag{3}$$

We now have an equation of the form Ax = y where:

$$A = (LL^T)^{-1} L \tag{4}$$

$$x = I \tag{5}$$

$$y = B \tag{6}$$

To reinforce the dimensionality, L is a 3x7, I is a 7xP and B is a 3xP for the entire set of images. For a single image L is a 3x1, I is a 1xP and B is a 3xP.

Equation (3) is of the same form which I used in the function *estimatePseudonormalsCalibrated* when solving for B:

```
def estimatePseudonormalsCalibrated(I, L):

    B = np.linalg.inv(L @ L.T) @ L @ I

    return B
```

1f

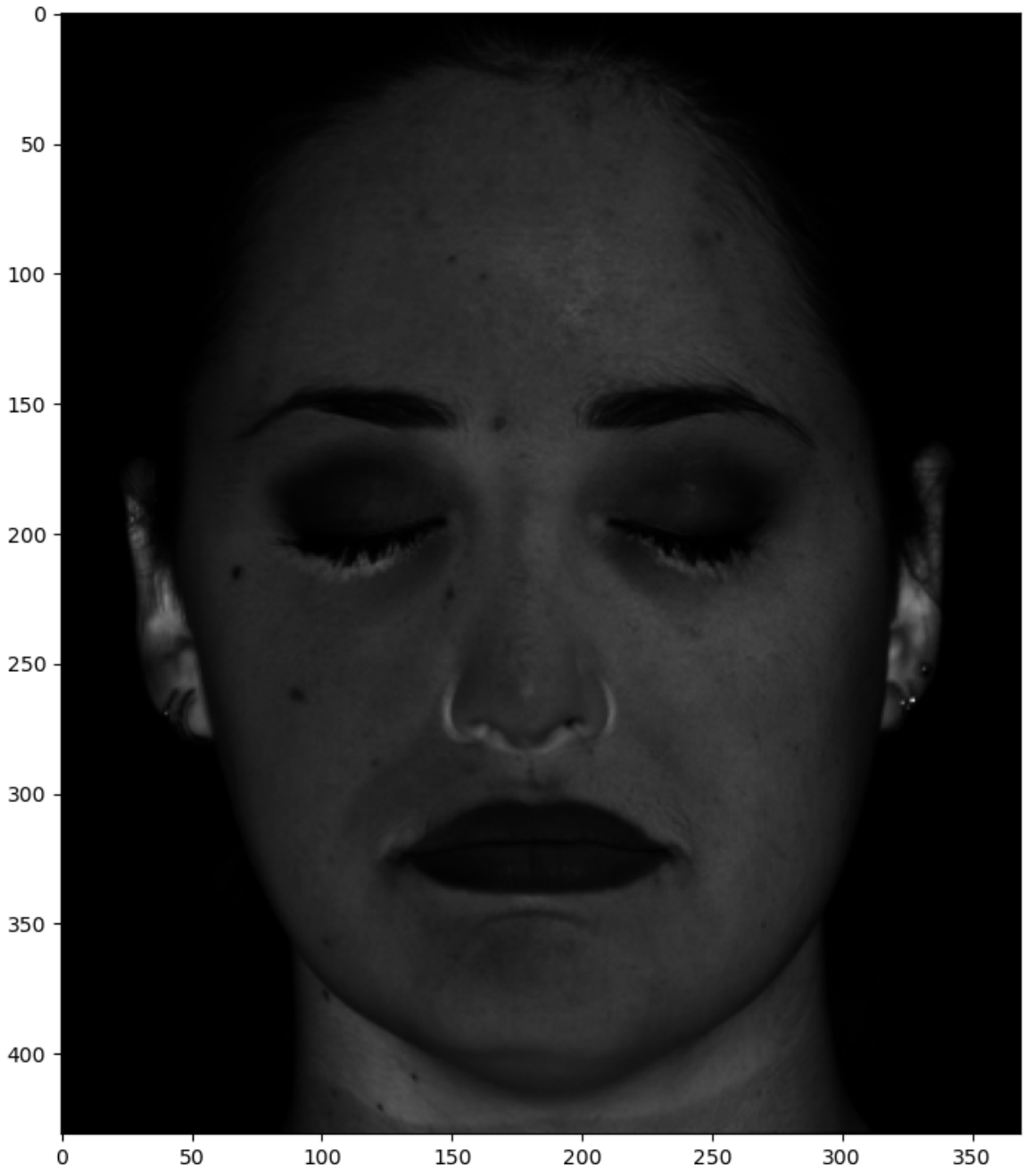

Figure 5: Q1 Albedoes

One parameter this albedo estimation method is not robust to is the influence of a cast shadow. Since the albedo is representative of the amount of incident light reflected off a surface, any features subject to a shadow may not recover the albedo value correctly. Looking at the original images in the /data folder we can see there is a heavy shadow cast on the upper portion of her neck caused by the overhang

of her chin. Therefore, her neck in the albedo image is appearing to be a lighter color than expected. We can see this same phenomenon occurring under her nose, under her eyelashes and on the insides of her ear.

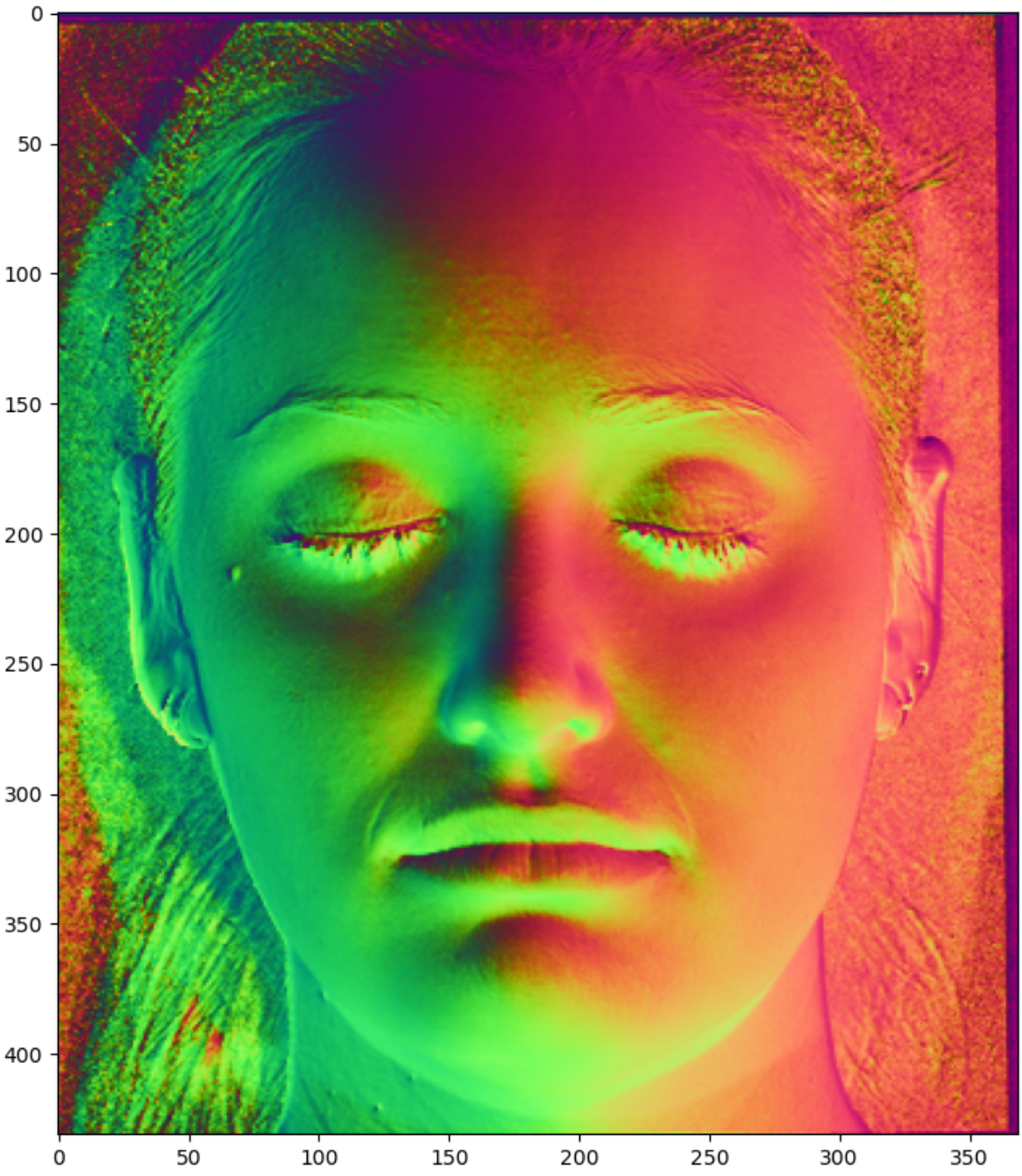

Figure 6: Q1 Normals

Yes - the normals match my expectation of curvature in the face given the way I normalized the

vectors. Each vector that was originally in the range of (-1, -1, -1) to (1, 1, 1) was normalized to (0, 0, 0) to (1, 1, 1). Then each component of the vector was used to represent that vector as a single color on the RGB image. Therefore the colors are representative of of the general direction of the normals condensed into a single value. Given the input images, it's rational that two dominating colors emerge both representing relatively opposite directions on either side of the face given the face is symmetrical.

1g

Generally speaking, the surface normal at a point in 3D space on a surface $F(x, y, z)$ is defined by the gradient:

$$n = \nabla F(x, y, z) \tag{7}$$

In the 1D case where z = f(x), we can rewrite the equation of the surface as:

$$F(x, y, z) = z - F(x, y) \tag{8}$$

Taking the gradient with respect to each of the variables in equation (8), our equation for n looks like:

$$n = \left(-\frac{dF}{dx}, -\frac{dF}{dy}, 1\right) \tag{9}$$

Here we can see equation (9) is synonymous with the equation of the normal $n = (n1, n2, n3)$ if we replace $\frac{dF}{dx}$ and $\frac{dF}{dy}$ with the following values and multiple every component by n3:

$$f_x = \frac{dF}{dx} = -\frac{n1}{n3} \tag{10}$$

$$f_y = \frac{dF}{dy} = -\frac{n2}{n3} \tag{11}$$

1h

Constructing both g_x and g_y with the given formula, we end up with a loss of a column and a row respectively. Therefore g_x is a 4x3 and g_y a 3x4.

$$g_x = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{12}$$

$$g_y = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \end{bmatrix} \tag{13}$$

Using g_x to construct the first row of g:

$$g = \begin{bmatrix} 1 & 2 & 3 & 4 \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \end{bmatrix} \tag{14}$$

Followed by g_y to reconstruct the rest of g:

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

Using g_y to construct the first column of g:

$$g = \begin{bmatrix} 1 & . & . & . \\ 5 & . & . & . \\ 9 & . & . & . \\ 13 & . & . & . \end{bmatrix} \tag{16}$$

Followed by g_x to reconstruct the rest of g:

$$g = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \tag{17}$$

Re-construction of the image using both methods, results in the same matrix proving that this set of discrete samples is integrable. However, we cannot always guarantee that a set of discrete samples will reasonably reconstruct an image. For example, if we were to introduce non-uniform noise to each the gradient fields in the example problem above, this might cause the gradient field to become non-integrable. If there are multiple paths in which the function can integrate from point 1 to point 2, we can no longer deterministically say the set of samples will reconstruct the same images.

This image taken from the following source visualizes both an integrable and a non-integrable gradient field. If there is too much noise in the image or in the sampling process, there is the possibility of there existing multiple paths to traverse from one point to the next during reconstruction.

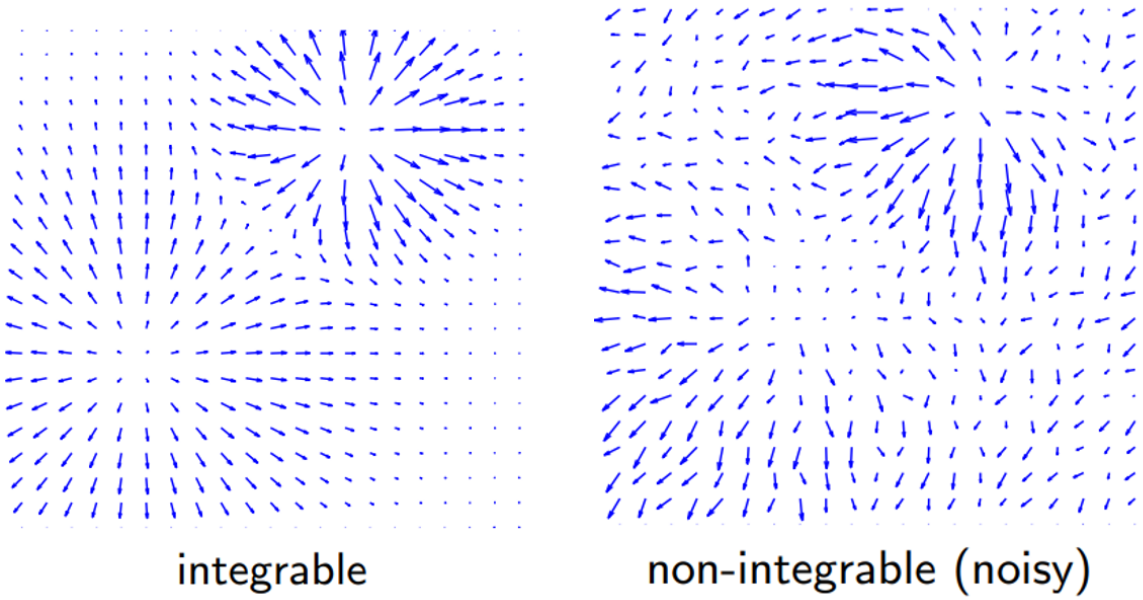

Figure 7: Integrable vs. Non-Integrable Gradient Fields

1i

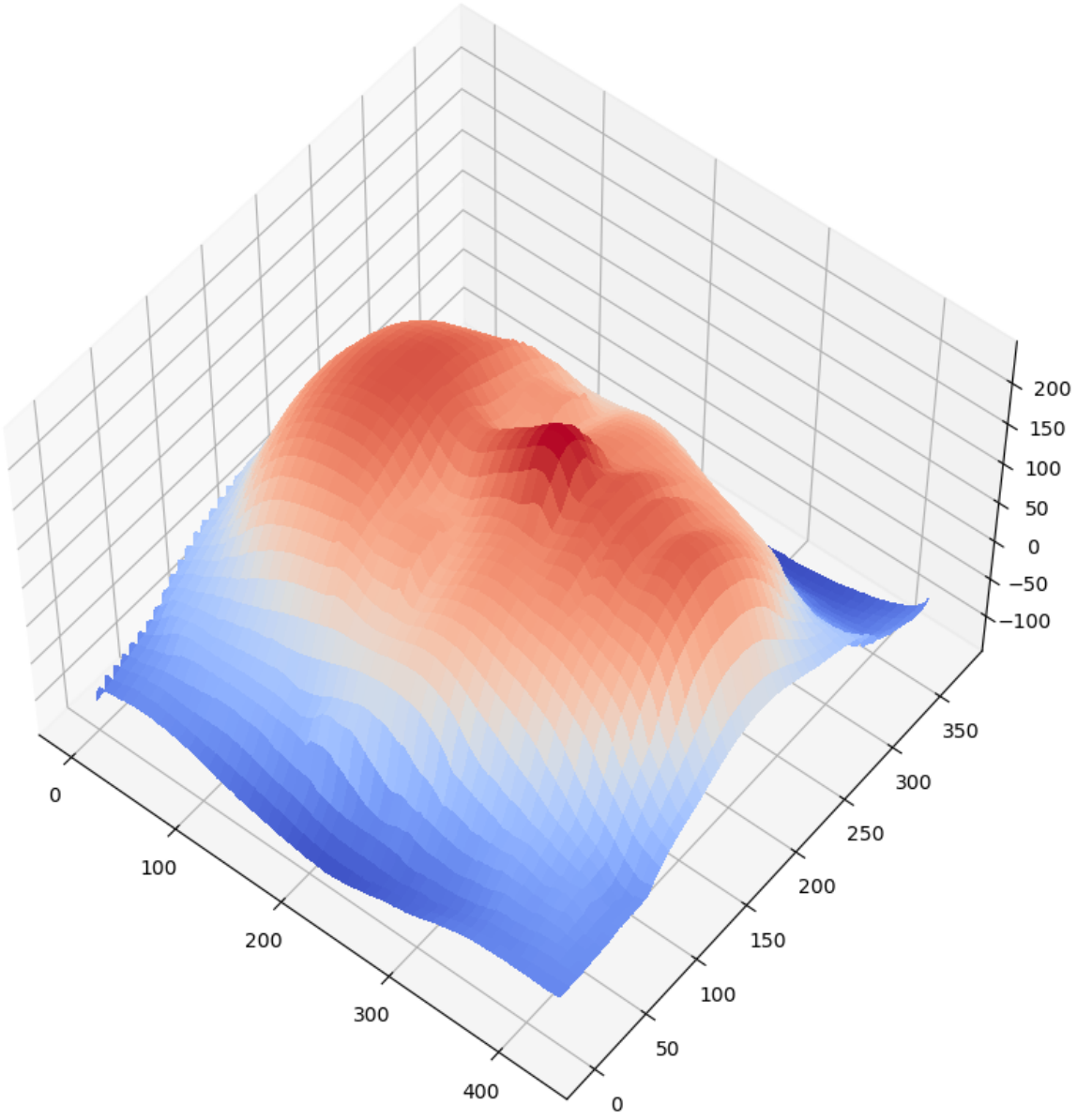

Figure 8: Q1 Surface Plot 1

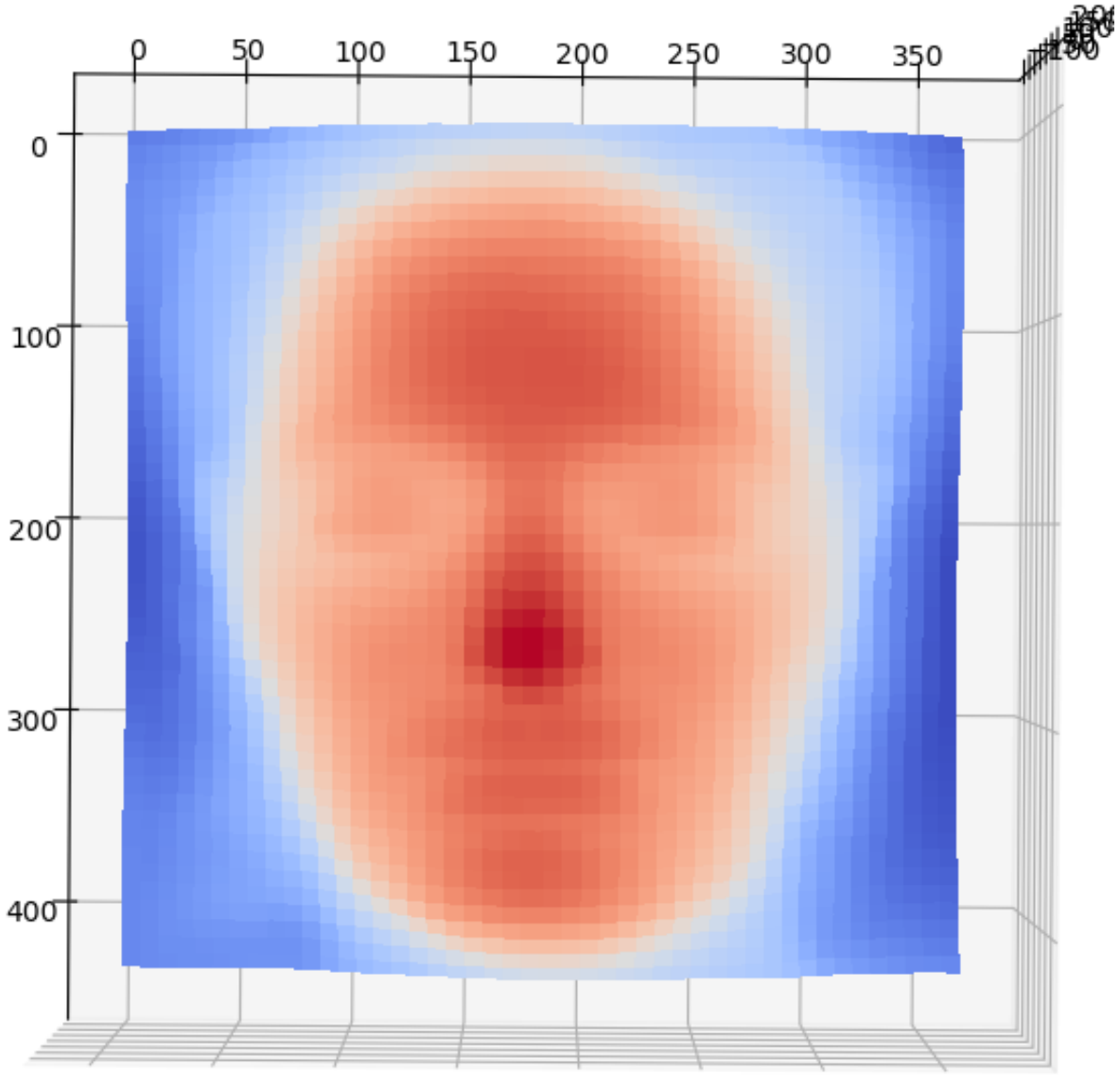

Figure 9: Q1 Surface Plot 2

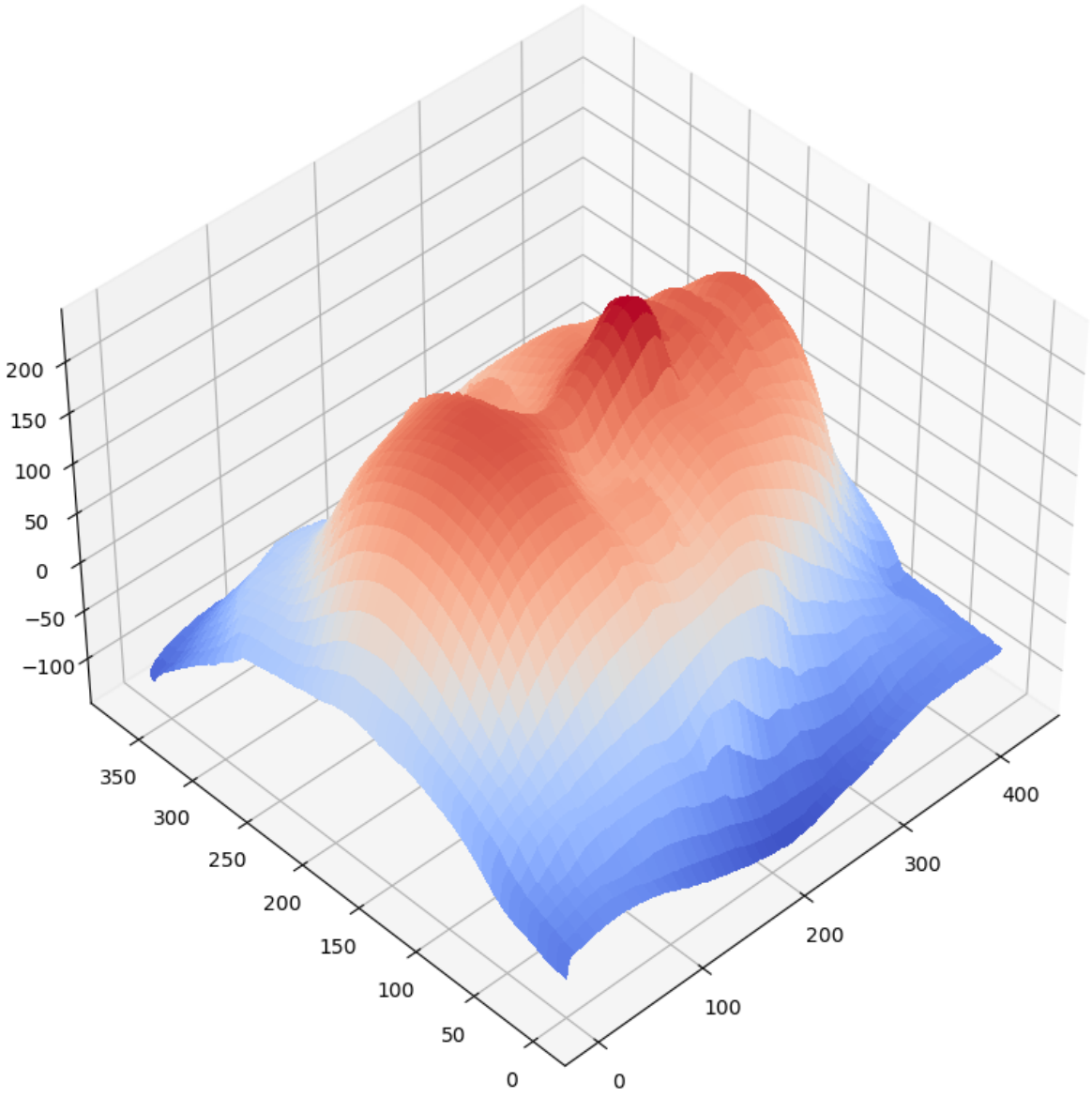

Figure 10: Q1 Surface Plot 3

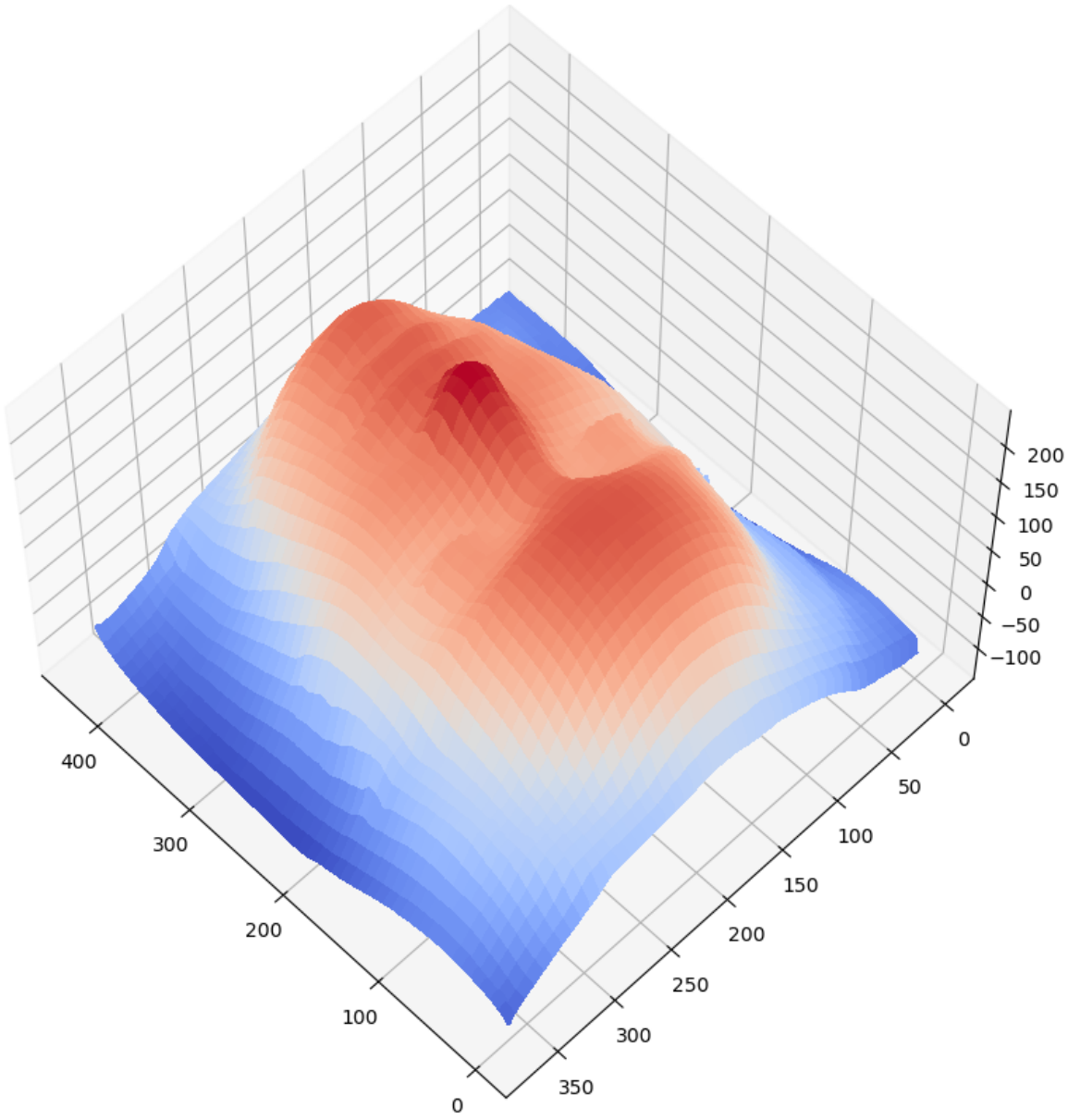

Figure 11: Q1 Surface Plot 4

2a

Based off my explanation in question 1d, we know the rank of the singular values vector of I is of rank 7 opposed to rank 3 because of noise. In order to force the rank 3 constraint on I, we set all rows and columns not associated with the top three singular values equal to zero. We will then denote the new SVD decomposition matrices and vectors as:

$$\hat{U} : 7x3$$

$$\hat{\Sigma} : 3x1$$

$$\hat{V} : 3xP$$

We can now use these new matrices and vectors to recreate I. Please note my dimensions in the Python code are slightly varied given the dimension of the return values of numpy's SVD funcion:

$$\hat{I} = \hat{U}\hat{\Sigma}\hat{V}^T \tag{18}$$

After forcing this constraint on the factorization, we can now estimate $\hat{B}$ and $\hat{L}$ with the following equations:

$$\hat{L} = \hat{U}\hat{\Sigma} \tag{19}$$

$$\hat{B} = \hat{V}^T \tag{20}$$

2b

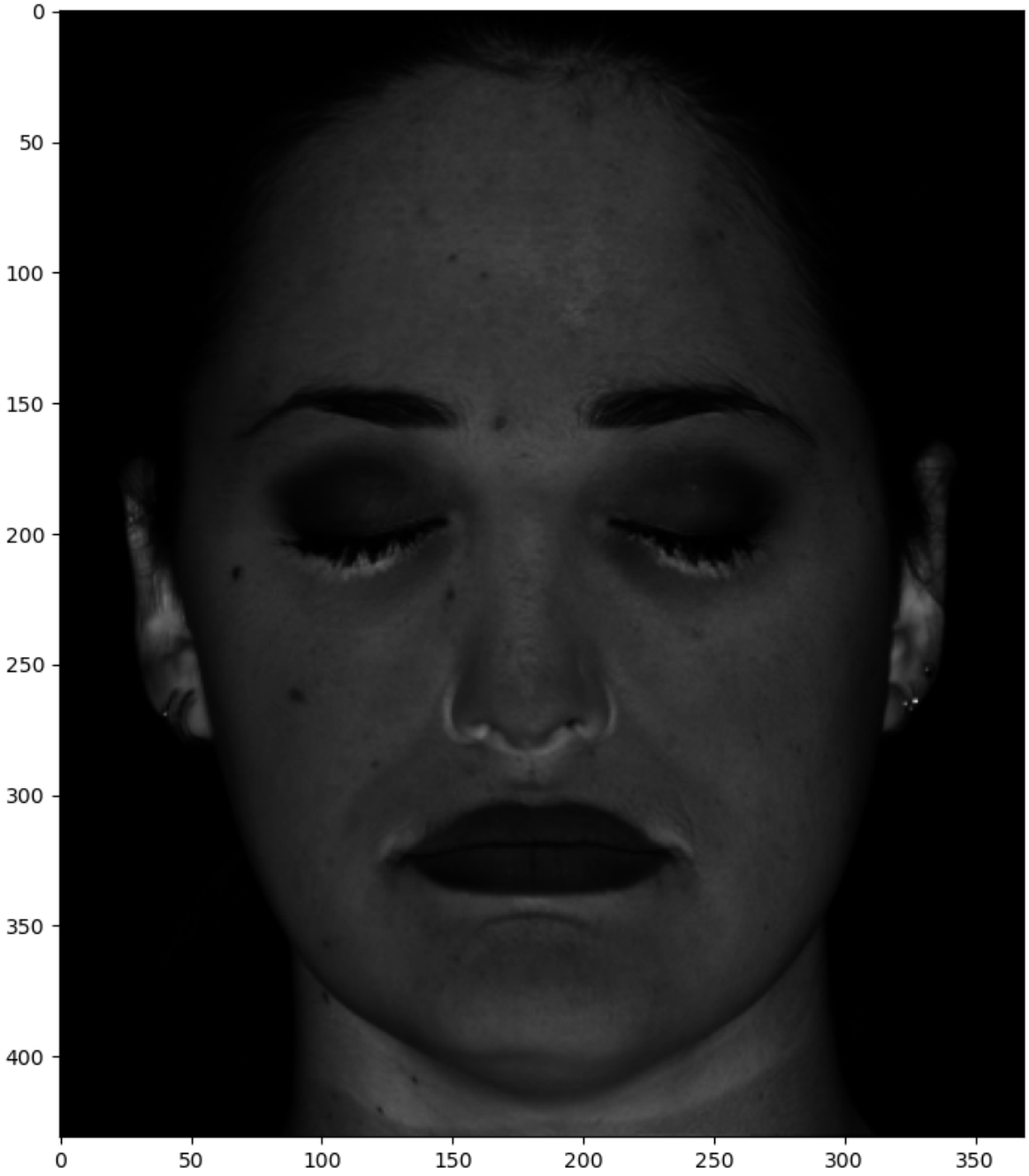

Figure 12: Q2 Albedoes

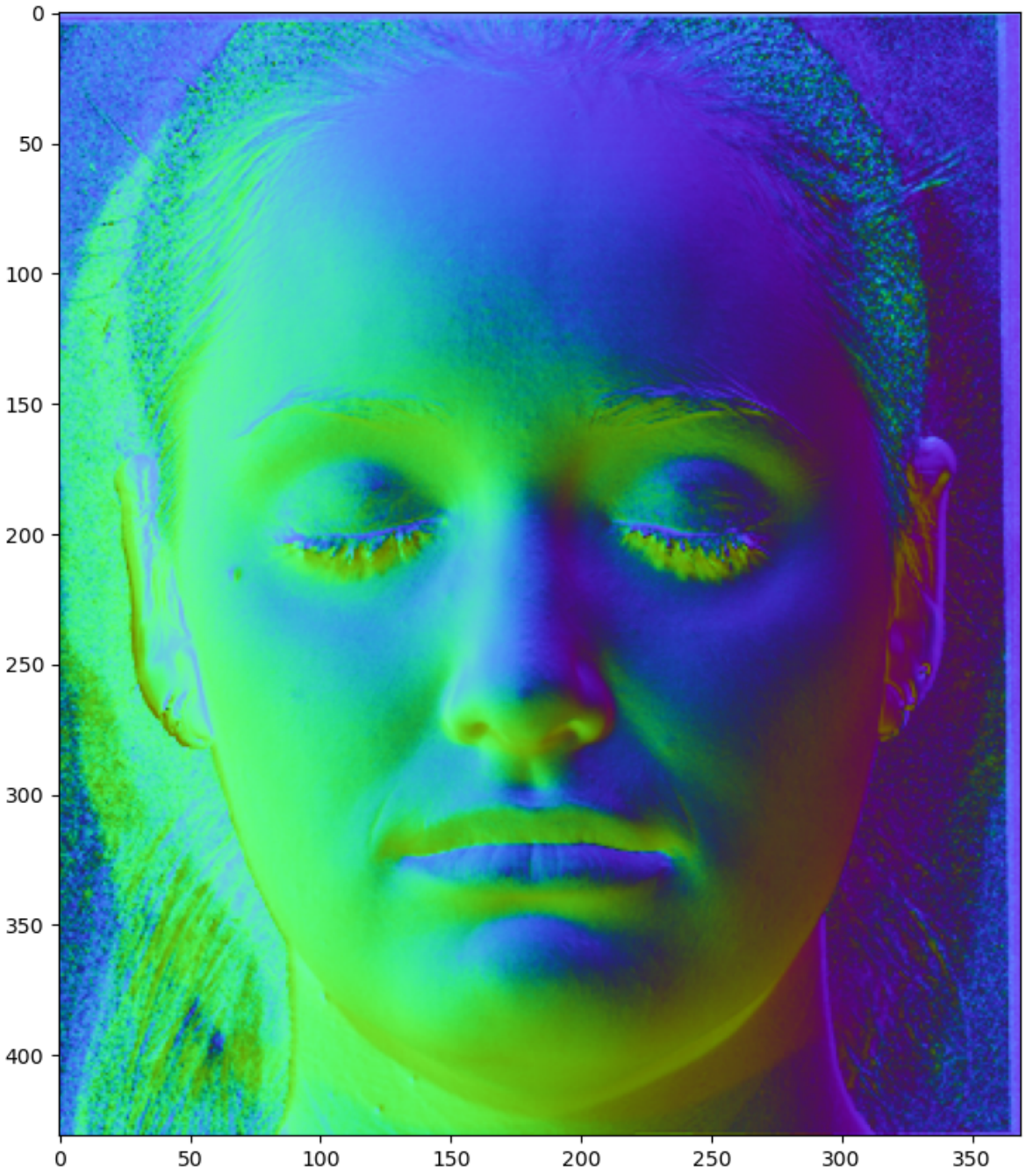

Figure 13: Q2 Normals

Please note my images for 2b are displayed with the normalization factor I will present in 2c. I did not find a significant difference in the display of the albedos and normal images.

2c

Observing the value of L pre and post factorization, we can see the matrices deviate in magnitude rather significantly. Therefore L is not accurately representative of L0.

$$L0 = \begin{bmatrix} -0.1418 & 0.1215 & -0.069 & 0.067 & -0.1627 & 0. & 0.1478 \\ -0.1804 & -0.2026 & -0.0345 & -0.0402 & 0.122 & 0.1194 & 0.1209 \\ -0.9267 & -0.9717 & -0.838 & -0.9772 & -0.979 & -0.9648 & -0.9713 \end{bmatrix}$$

$$L = \begin{bmatrix} -26.66059692 & -34.47622162 & -21.45222072 & -33.36284755 & -31.99401452 & -30.170518 & -29.8665421 \\ 3.43866505 & -8.40647171 & 1.81078981 & -2.2711264 & 8.43764405 & 1.69084744 & -2.87597905 \\ 5.70699349 & 3.08107583 & 1.30403367 & -0.0525438 & -0.94372086 & -2.77169539 & -5.7181112 \end{bmatrix}$$

Alternatively we can use the Tomasi–Kanade factorization to improve this L. By using the following equations for our $\hat{B}$ and $\hat{L}$. Please note my dimensions in the Python code are slightly varied given the dimension of the return values of numpy's SVD funcion:

$$\hat{L} = \hat{U}\hat{\Sigma}^{\frac{1}{2}} \tag{21}$$

$$\hat{B} = \hat{\Sigma}^{\frac{1}{2}}\hat{V}^T \tag{22}$$

Using this factorization, we can get an L which is closer to L0:

$$L = \begin{bmatrix} -2.99267472 & -3.86998525 & -2.40803005 & -3.74500806 & -3.59135539 & -3.38666635 & -3.3525448 \\ 0.94780484 & -2.31708946 & 0.49911094 & -0.62599426 & 2.32568155 & 0.46605103 & -0.79271078 \\ 1.87934697 & 1.01461663 & 0.42942606 & -0.01730299 & -0.3107729 & -0.91273581 & -1.8830081 \end{bmatrix}$$

2d

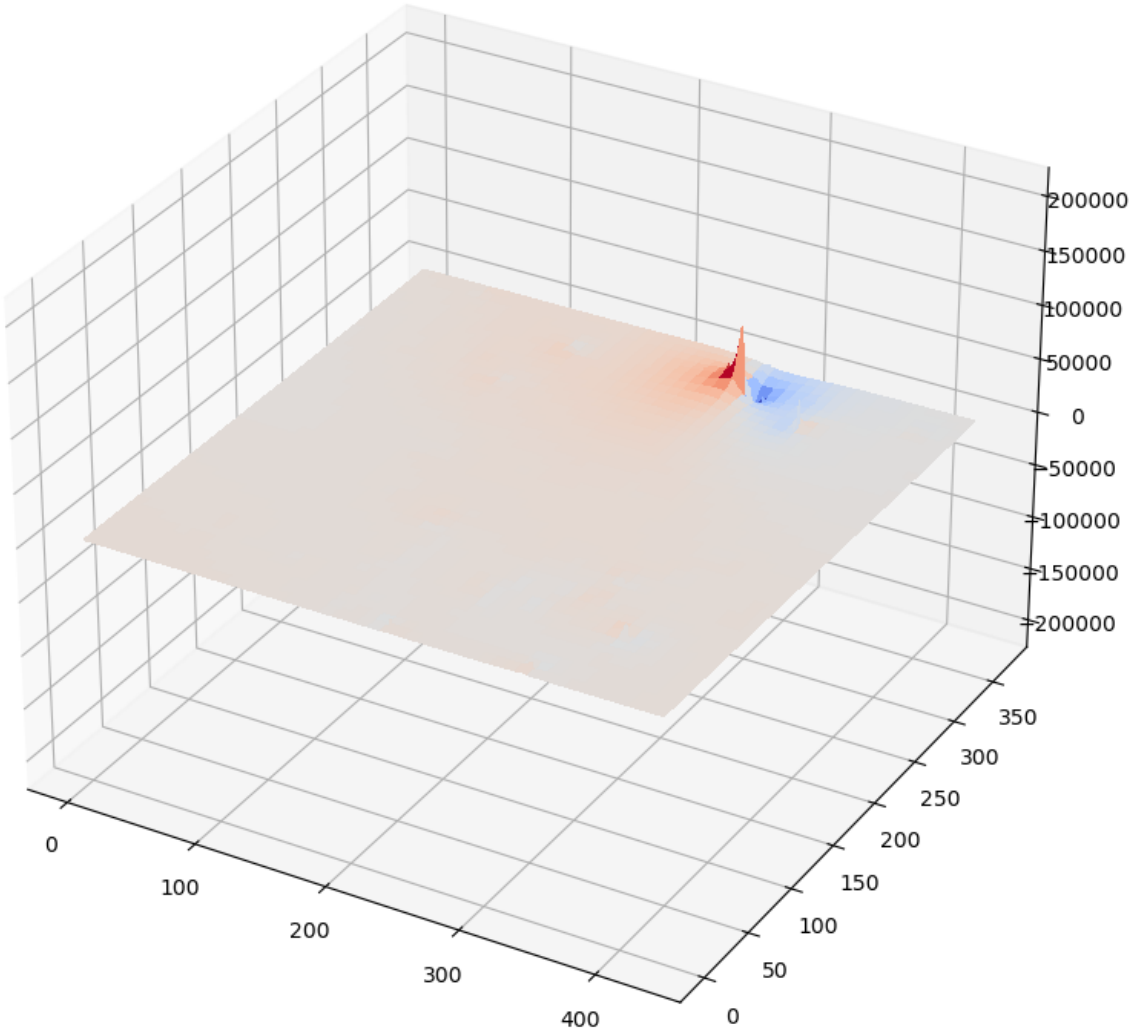

Figure 14: Q2 Surface - Not Enforcing Integrability

Without enforcing integrability, this output clearly is not representative of a face.

2e

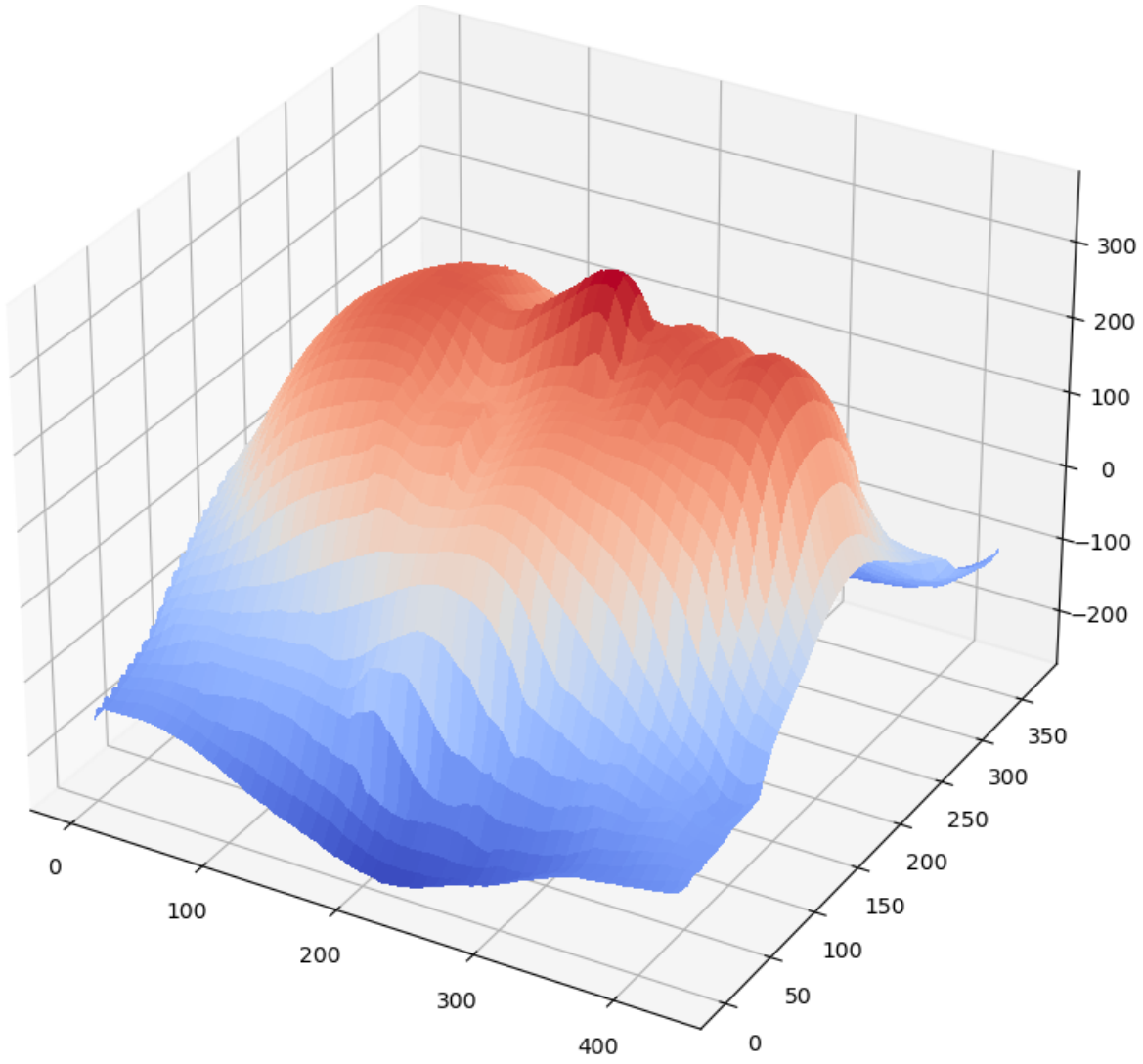

Figure 15: Q2 Surface 1 - Enforcing Integrability

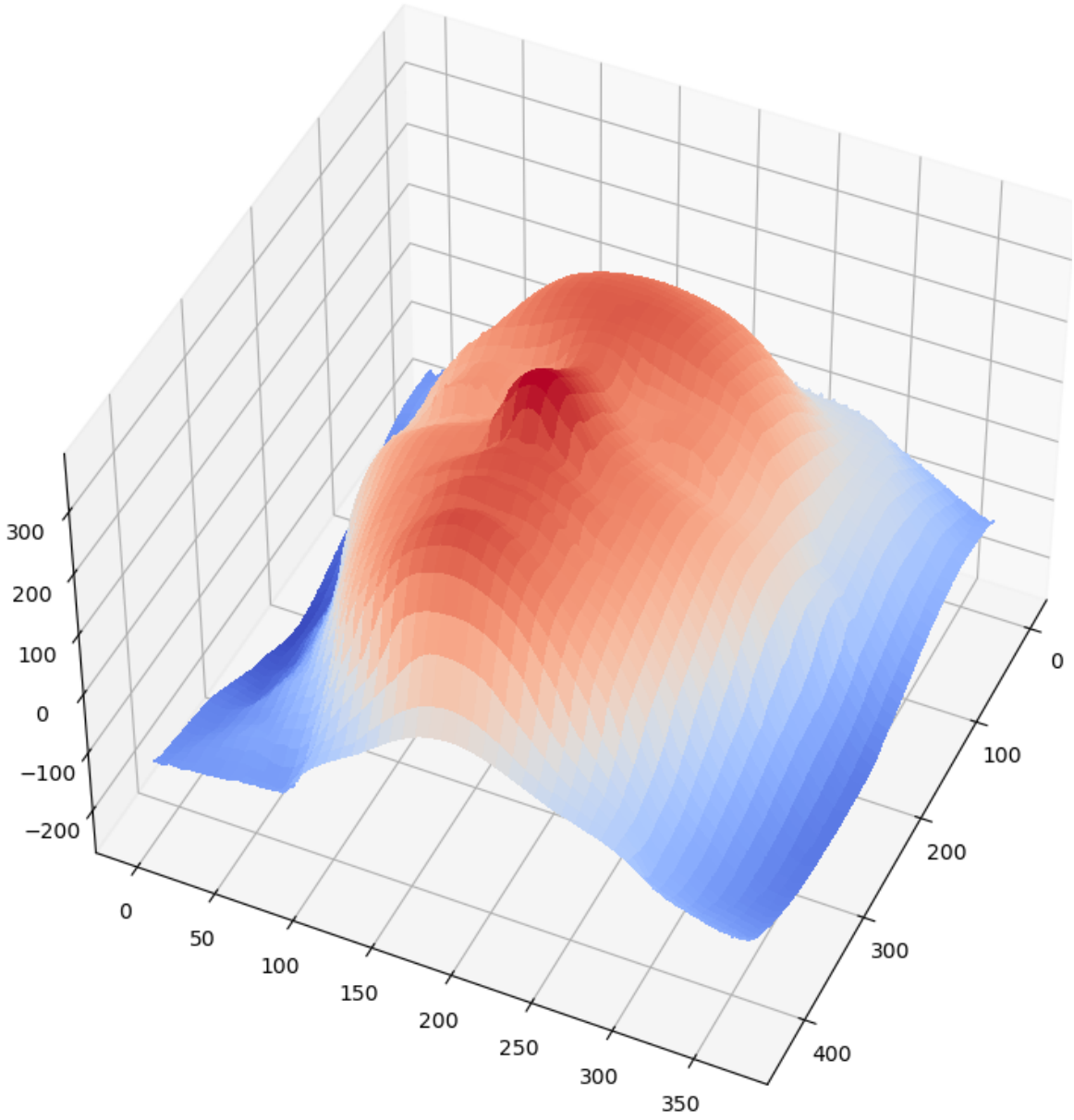

Figure 16: Q2 Surface 2 - Enforcing Integrability

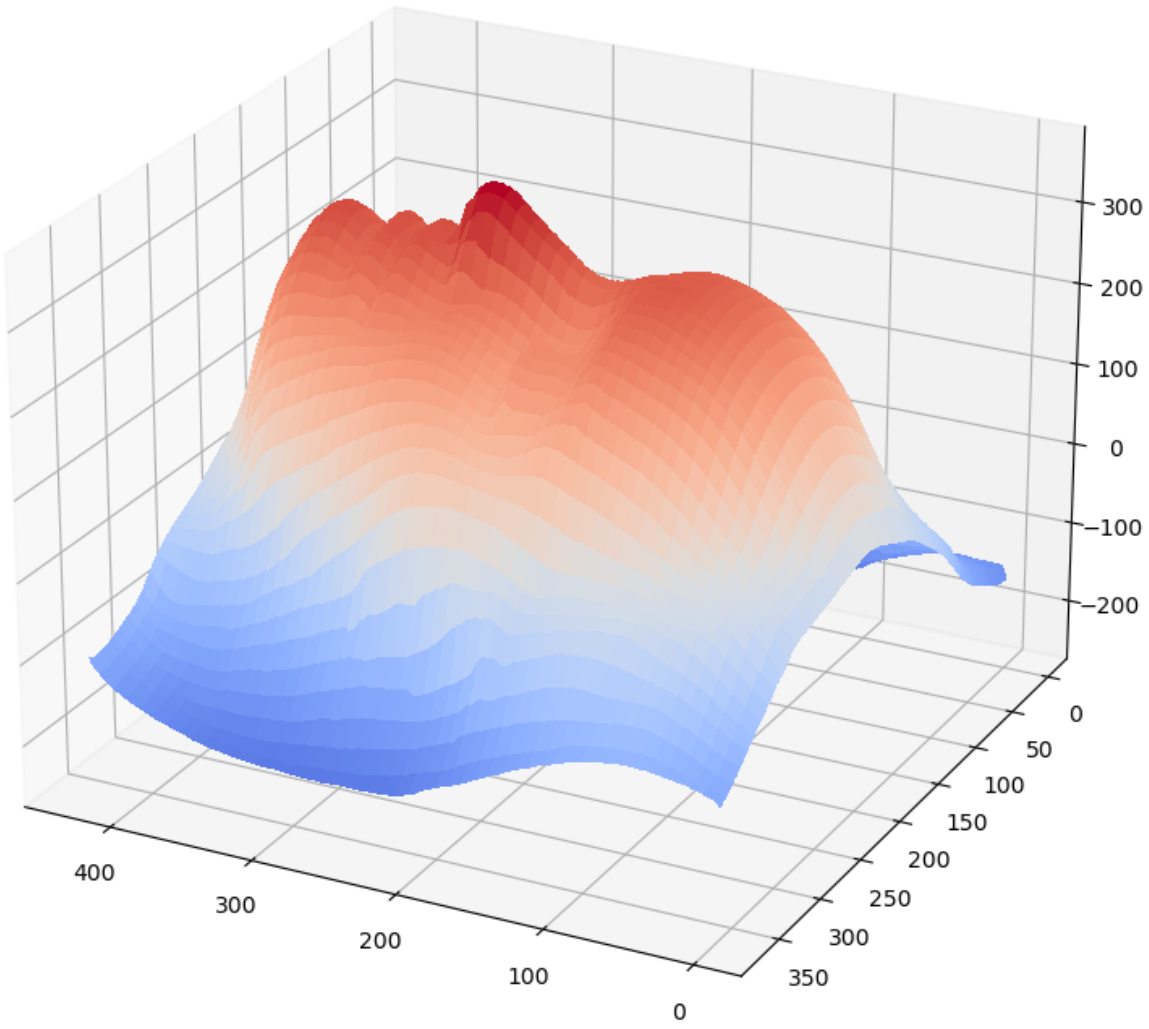

Figure 17: Q2 Surface 3 - Enforcing Integrability

After enforcing itegrability, the output looks like the expected output of a calibrated photometric stereo camera.

2f

An increase in μ causes the surface to pivot around the x-axis while also flattening out the surface. An increase in ν causes the surface to rotate around the diagonal axis while also flattening out the surface. The λ parameter proportionally scales the magnitude of the z-coordinates. i.e. a higher λ leads to highter z values and a lower λ leads to lower z values.

Bas-relief sculptures are famous for having perceived depth despite being relatively flat surfaces. From a frontal POV, these sculptures have full 3D depth. The ambiguity we are describing is aptly named the bas-relief ambiguity as we cannot discern the depth of an image when looking at it head on. Any small motions the viewer makes will not relieve this problem either. The only was to truly understand the magnitude of depth would be for the viewer to move enough until they can see the sides of the sculpture.

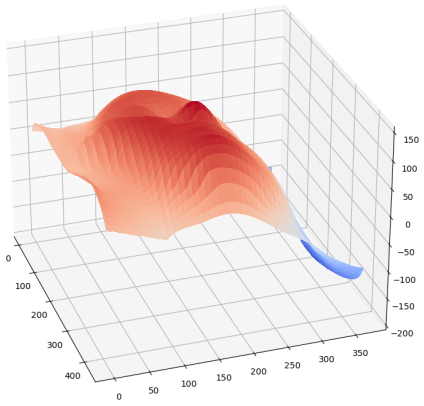

Figure 18: Q2 Bas-Relief [u = 3, v = 0, $\lambda = 1$]

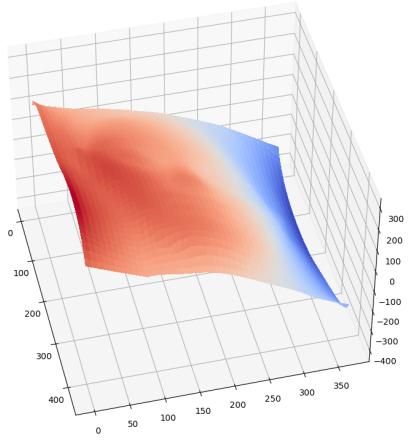

Figure 19: Q2 Bas-Relief [u = 8, v = 0, $\lambda = 1$]

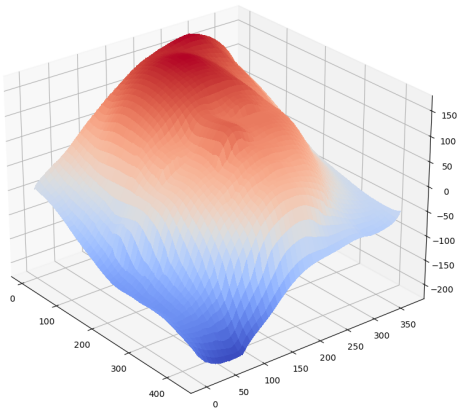

Figure 20: Q2 Bas-Relief [u = 0, v = 4, $\lambda = 1$]

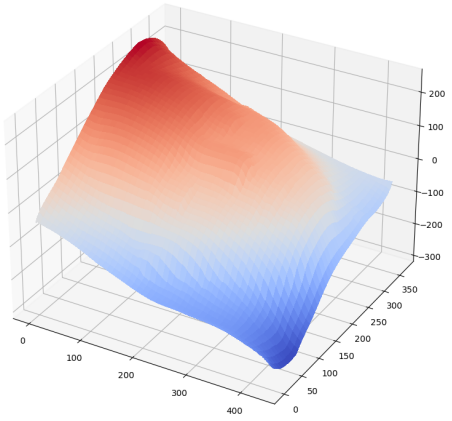

Figure 21: Q2 Bas-Relief [u = 0, v = 6, $\lambda = 1$]

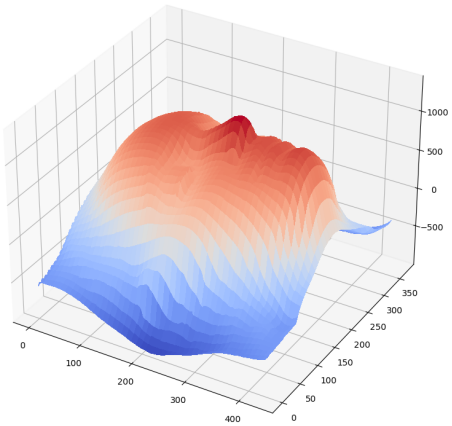

Figure 22: Q2 Bas-Relief [u = 0, v = 0, $\lambda = 10$]

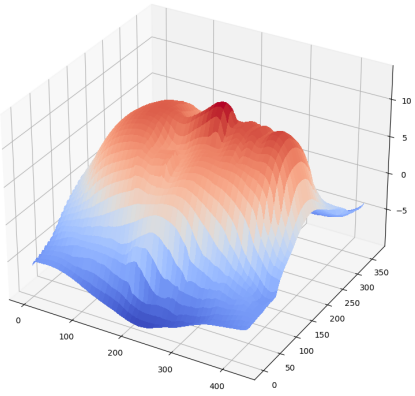

Figure 23: Q2 Bas-Relief [u = 0, v = 0, $\lambda = 0.1$]

2g

Looking at the equation for the bas-relief equation found here, we see the z-value is directly scaled by λ.

$$\bar{f}(x, y) = \lambda f(x, y) + \mu x + \nu y \tag{23}$$

Therefore to achieve the flattest surface possible, we would want to scale the z-coordinate which is representative of depth by a very small factor while keeping u and v equal to zero. An example of setting λ to a small value can be found below where the z-coordinates have a very small magnitude implying "flatness".

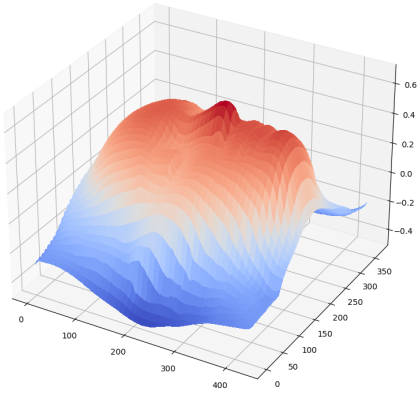

Figure 24: Q2 Bas-Relief [u = 0, v = 0, $\lambda = 0.005$]

2h

Adding in additional shadowing and shading will not help resolve this ambiguity. For each image we are passed with a unique lighting source, there exists a bas-relief transformation of the form found in equation (23).

Therefore, for each image we can at best determine the relief up to a 3 DoF linear equation. One way to resolve this ambiguity and determine depth would be for the viewer to move a significant distance.