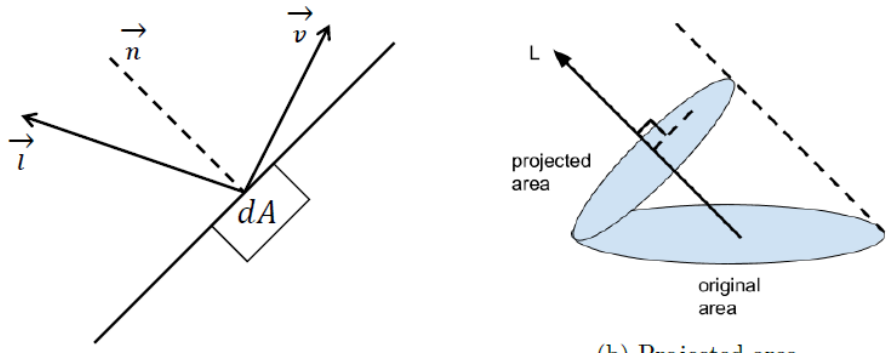
16720: Computer Vision

HW 6: Photometric Stereo

Harsh Dhruva

AndrewID: hdhruva

1 a)



(a) Geometry of photometric stereo

(b) Projected area

In the n-dot-l lighting model above, $\vec{l}$ represents the lighting direction or the source direction, $\vec{n}$ represents the surface normal to the area dA, and $\vec{v}$ represents the viewing direction. Lets say the angle between the source and the surface normal is θ.

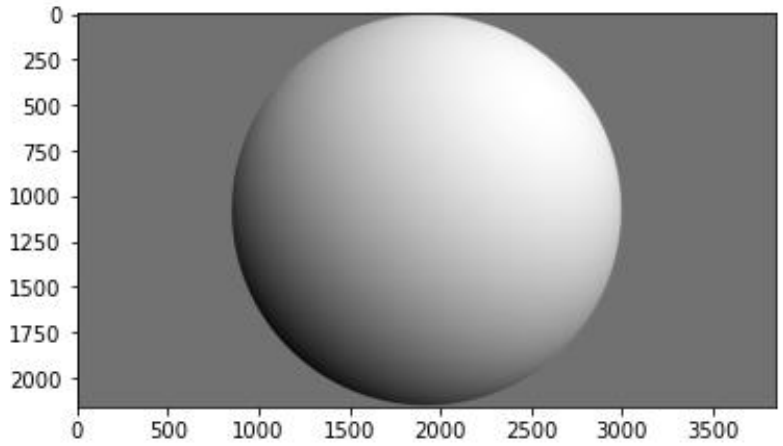
The dot product comes from the assumption that the object is Lambertian and therefore follows the Lambert cosine law, that says that the amount of light received by the user is proportional to $cos\theta$ which is $\vec{n} \cdot \vec{s}$ assuming directional unit vectors.

According to the Lambertian cosine law, if the value of θ is zero then, $cos\theta = 1$, which means the maximum intensity will be when the source direction is the same as the surface normal. So we can assume the light source at an angle θ to the original area dA is equivalent to the light source normal to the projected area $dAcos\theta$.

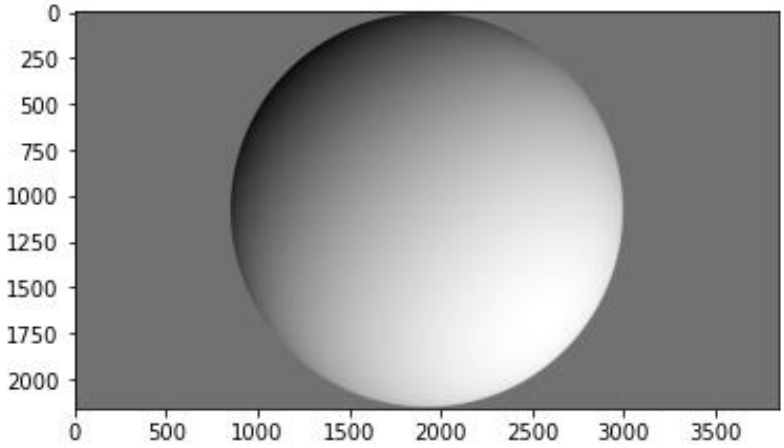
We assume diffused reflection which means the surface appears bright from all directions therefore it is independent of the viewing direction.

1 b)

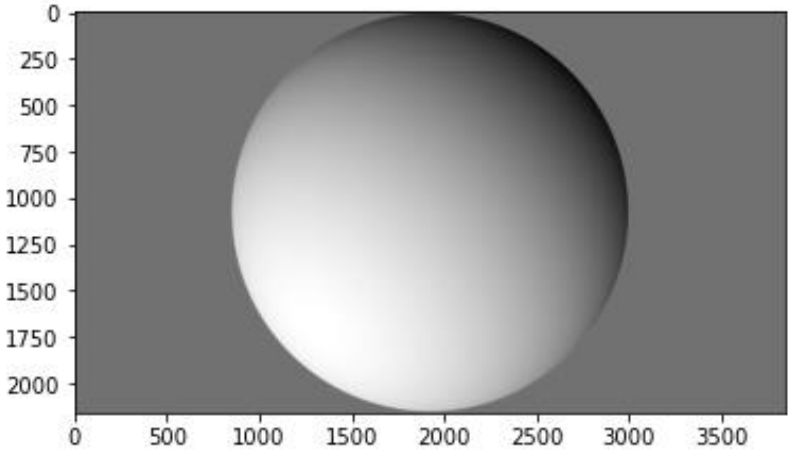
Lighting direction = $(1,1,1)/\sqrt{3}$



Lighting direction = $(1,-1,1)/\sqrt{3}$



Lighting direction = $(-1,-1,1)/\sqrt{3}$



1 c)

All images were datatype “uint16”

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

    """
    Question 1 (c)

    Load data from the path given. The images are stored as input_n.tif
    for n = {1...7}. The source lighting directions are stored in
    sources.mat.

    Paramters
    ---------
    path: str
        Path of the data directory

    Returns
    -------
    I : numpy.ndarray
        The 7 x P matrix of vectorized images

    L : numpy.ndarray
        The 3 x 7 matrix of lighting directions

    s: tuple
        Image shape

    """
    img = imread('../data/input_1.tif')
    H, W, _ = img.shape
    s = (H, W)
    I = np.zeros((7, H*W))
    for i in range(7):
        img = imread('../data/input_{}.tif'.format(i+1))
        img = rgb2xyz(img)
        img = img[:,:,1].flatten()
        I[i,:] = img

    L = np.load('../data/sources.npy')
    L = L.T
    return I, L, s
```

1 d)

The rank of **I** should be 3 since there are three degrees of freedom in the pseudo normal B, two degrees of freedom from the surface normal (since magnitude is 1) and one degree of freedom from albedo. Therefore we have three unknows we have to solve for to get the required information. We would not be able to solve the problem with a single equation since the known value of θ there are multiple solutions for **n** that would result in the same **n** dot **l**.

The SVD of **I** reports the following singular values. Here we see that the first three singular values are larger compared to the rest. The rank of **I** in this case is 7, so they do not completely agree with the rank 3 requirement. However, even this overdetermined system can be used to solve for the surface normal. The smaller singular values could possibly be due to the noise in the images, or that the light source positions were less significant to render varied intensities.

```
array([79.36348099, 13.16260675,  9.22148403,  2.414729  ,  1.61659626,
       1.26289066,  0.89368302])
```

1 e)

$$I = L^T B$$

$7 \times P \quad 7 \times 3 \quad 3 \times P$

For one pixel,

$$\begin{bmatrix} I_{11} \\ I_{21} \\ \vdots \\ I_{71} \end{bmatrix} = \begin{bmatrix} L^T \end{bmatrix} \begin{bmatrix} B_{11} \\ B_{21} \\ B_{31} \end{bmatrix}$$

$7 \times 1 \quad 7 \times 3$

Stack for all the pixels,

$$\begin{bmatrix} [L^T] & 0 & \cdots & 0 \\ 0 & [L^T] & & 0 \\ \vdots & & \ddots & \\ 0 & \cdots & 0 & [L^T] \end{bmatrix} \begin{bmatrix} B_{11} \\ B_{21} \\ B_{31} \\ B_{12} \\ B_{22} \\ \vdots \\ B_{3P} \end{bmatrix} = \begin{bmatrix} I_{11} \\ I_{21} \\ \vdots \\ I_{71} \\ I_{12} \\ I_{22} \\ \vdots \\ I_{72} \\ \vdots \\ I_{7P} \end{bmatrix}$$

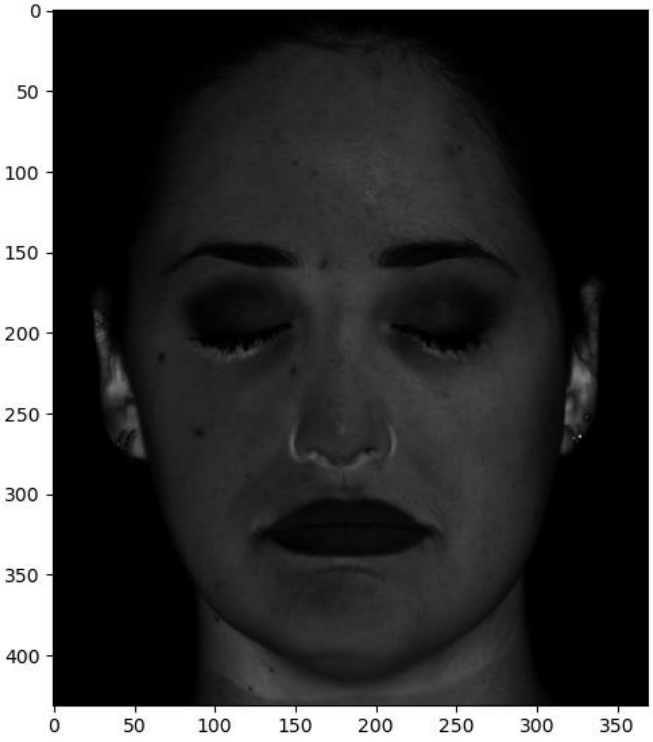
$7P \times 3P \quad 3P \times 1 \quad 7P \times 1$

$$Ax = y$$

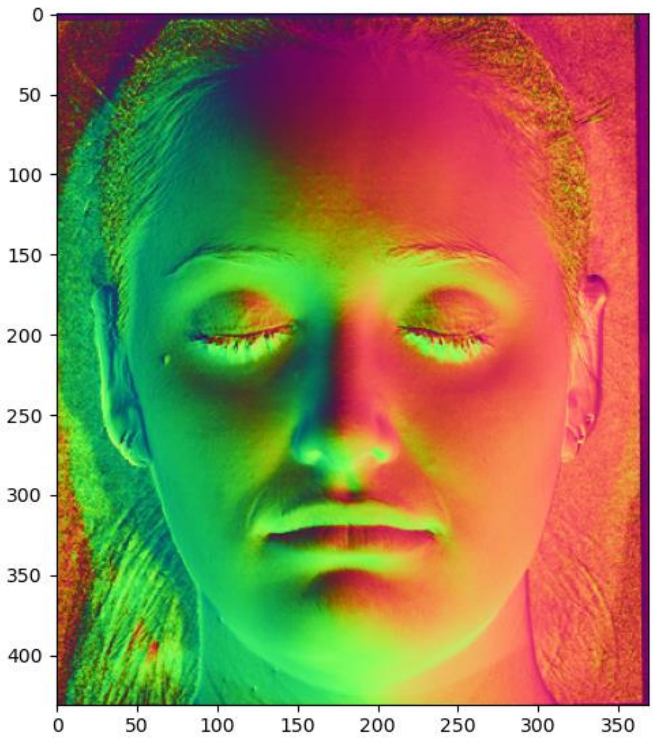
I looped through all the pixels and stacked the B matrix, since I ran into some memory issues, even using sparse module with these matrices.

1 f)

Albedo



Normals (normalized)



The Albedo Image is shown on the left. The face in the Albedo Image looks flatter. It does not really demonstrate the structure of the face with different shading under the lighting effect, as the normal image shows. However, it does distinguish the features like eyes, nose, mouth quite well, not necessarily depicting the most accurate shape of them. We see that the albedo is higher at the bottom of the nose, in the ears, neck and bottom of eyelids, as more light is being reflected there for some reason. Maybe light bounces around because of the shape of the ears, and the eyelashes and projects stronger. Or that the layers of skin in those areas are thinner, and light is not absorbed as much.

Yes, the normal image on the right matches the expectation of the curvature of the face. We can distinguish the curved features of the face such as the nose, lips and the eyes quite distinctly.

1 g) $z = f(x,y)$ $\quad n = (n_1, n_2, n_3)$

$z - f(x,y) = 0$ ← Implicit form

The surface normal is the gradient of the Implicit Form.

$$n = \nabla F(x,y,z) = (f_x, f_y, f_z)$$

$$f_x = -\frac{\partial f(x,y)}{\partial x}, \quad f_y = -\frac{\partial f(x,y)}{\partial y}, \quad f_z = 1$$

$$\therefore \quad n = (n_1, n_2, n_3) = \left(-\frac{\partial f(x,y)}{\partial x}, -\frac{\partial f(x,y)}{\partial y}, 1 \right)$$

2 dof since $\|n\| = 1$

matching the 1

$$\therefore n = \left(\frac{n_1}{n_2}, \frac{n_2}{n_3}, 1 \right)$$

$$\therefore \quad \frac{\partial f(x,y)}{\partial x} = -\frac{n_1}{n_3}, \quad \frac{\partial f(x,y)}{\partial y} = -\frac{n_2}{n_3}$$

1 h)

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

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

$$g = \begin{matrix} & \begin{matrix}0 & 1 & 2 & 3\end{matrix} \\ \begin{matrix}0\\1\\2\\3\end{matrix} & \begin{bmatrix}1 & 2 & 3 & 4\\ 5 & 6 & 7 & 8\\ 9 & 10 & 11 & 12\\ 13 & 14 & 15 & 16\end{bmatrix}\end{matrix} \Rightarrow g_x = \begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

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

1. $g(0,0) = 1$

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

using g_x

g_y

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

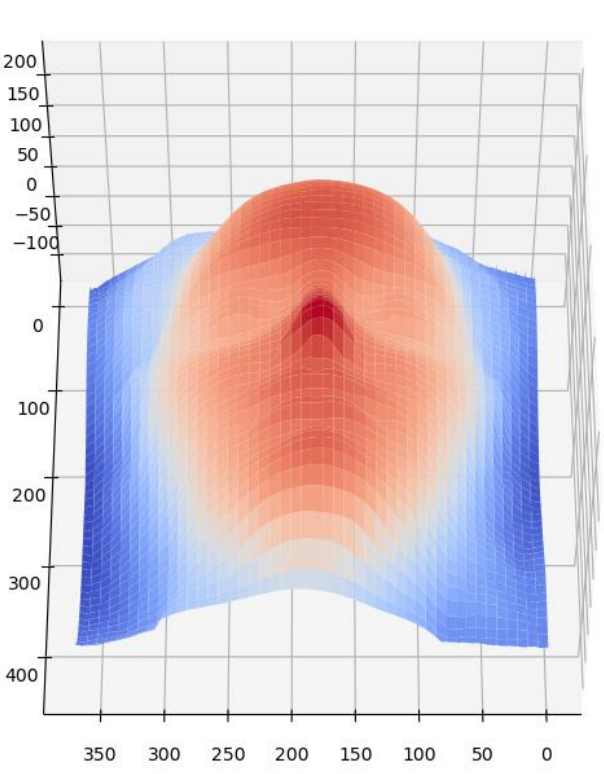
g_x

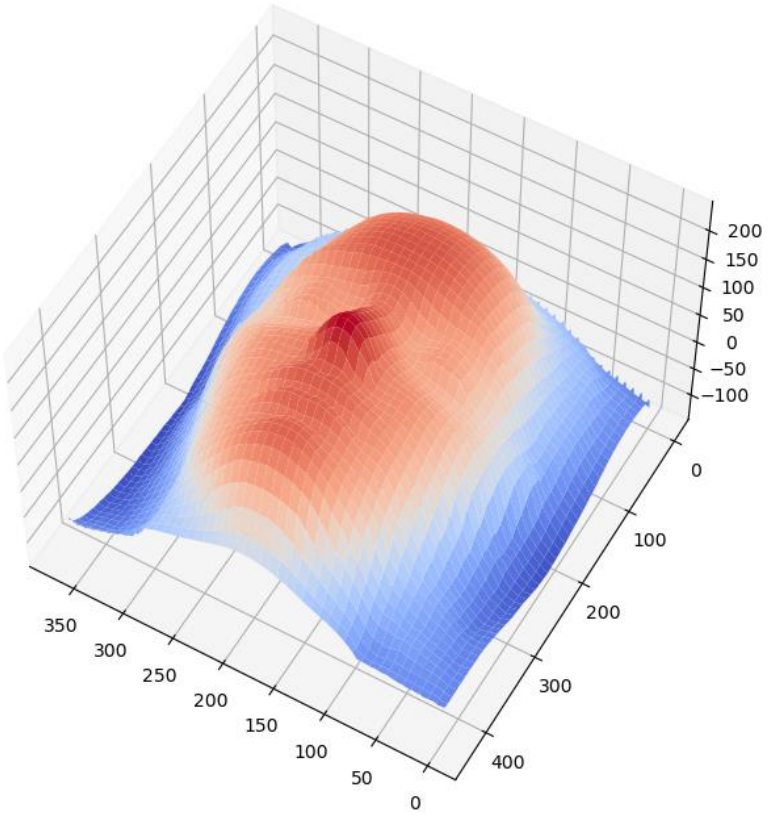
g_y

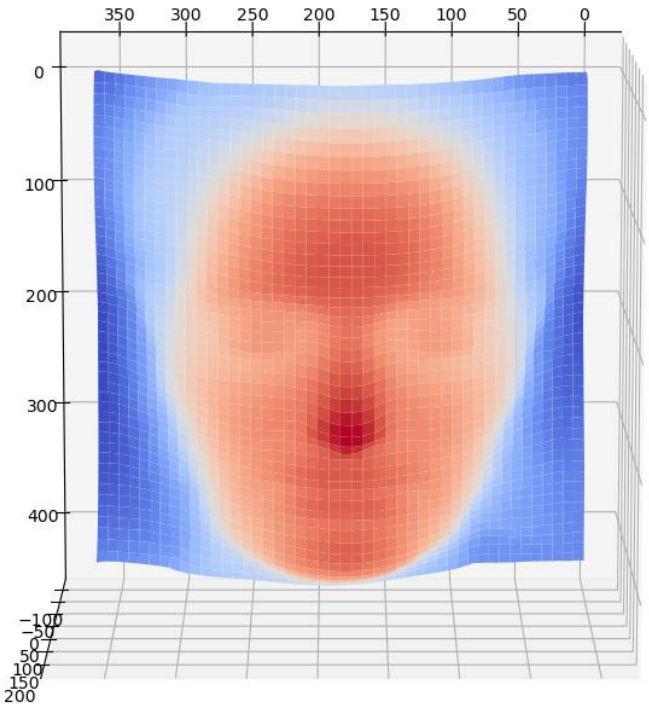
Yes the reconstructed g's are the same.

We could modify the gradients above by abruptly changing a value in the Matrix. This would then make the g reconstruction not match. The gradients estimated in our way are only due considering the change from the neighboring pixel. Therefore if there is nose, or an abrupt change of intensity in a small area, say by some laser, there would be a high value in the gradient matrix, which affects the smoothness of the surface or the uniformity of the gradients, since we measure gradient based on neighboring pixels. This would make the gradients non integrable. Therefore a better way would probably be considering bigger pixel area intensity changes for the surface to be rendered smoother.

1 i)


200
150
100
50
0
−50
−100
0
100
200
300
400
350 300 250 200 150 100 50 0


200
150
100
50
0
−50
−100
0
100
200
300
400
350 300 250 200 150 100 50 0


350 300 250 200 150 100 50 0
0
100
200
300
400
−100
−50
0
50
100
150
200

2 a)

Once $\hat{\Sigma}$ is obtained by making the last four values in the diagonal matrix O, $\begin{bmatrix} s_1 & 0 & 0 & \cdots \\ 0 & s_2 & 0 & \\ 0 & 0 & s_3 & \\ \vdots & & & \ddots \end{bmatrix}$

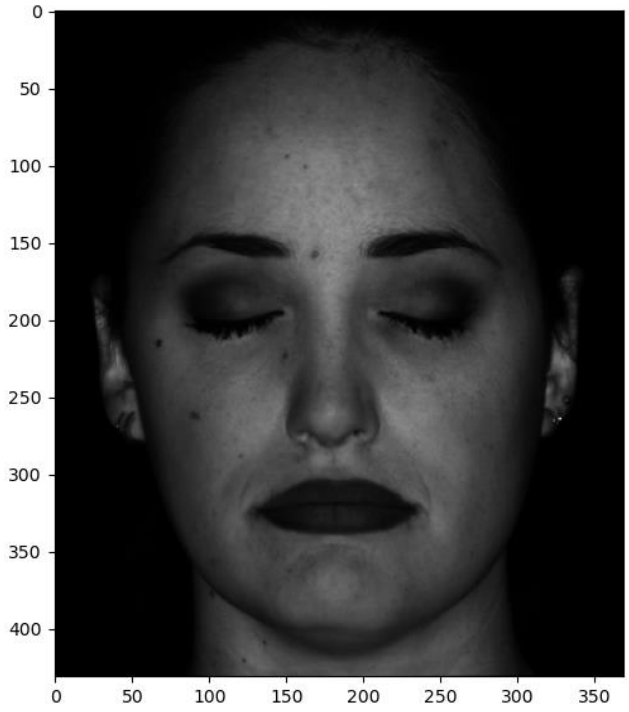
we have $I = L^T B$ (7×P = 7×3 · 3×P)

$\hat{M} = U \hat{\Sigma} V^T$ (7×7 = 7×7 · 7×P)

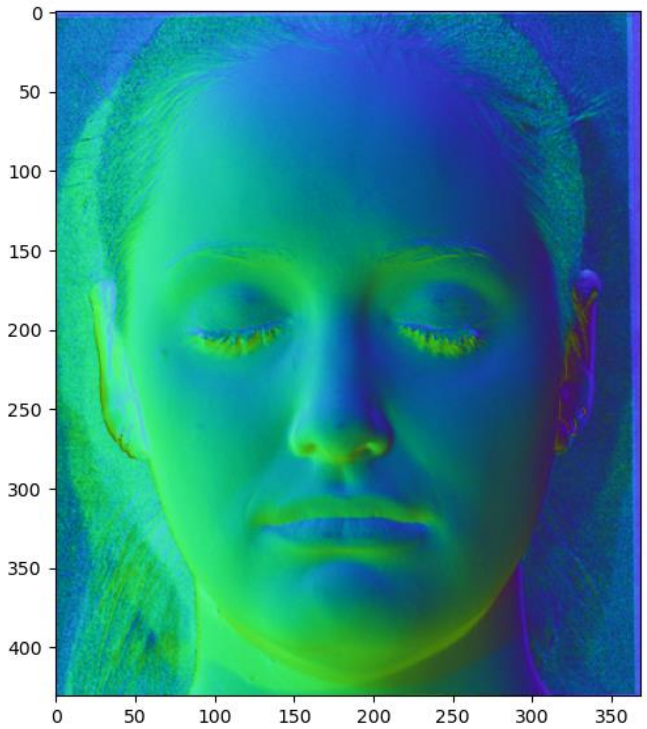
$\therefore$ To factorize, $L = U \hat{\Sigma}^{1/2}$ $\quad B = \hat{\Sigma}^{1/2} V^T$

then truncate by removing the zeros and reshape appropriately.

2 b)



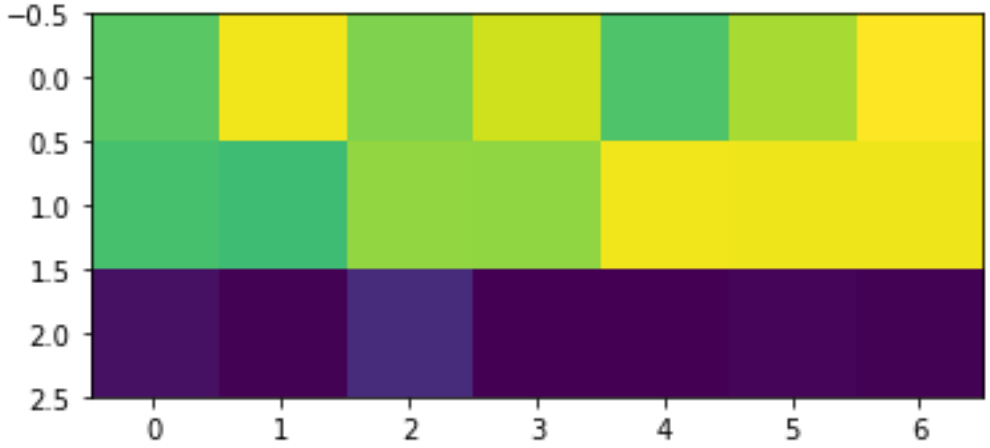
Albedo



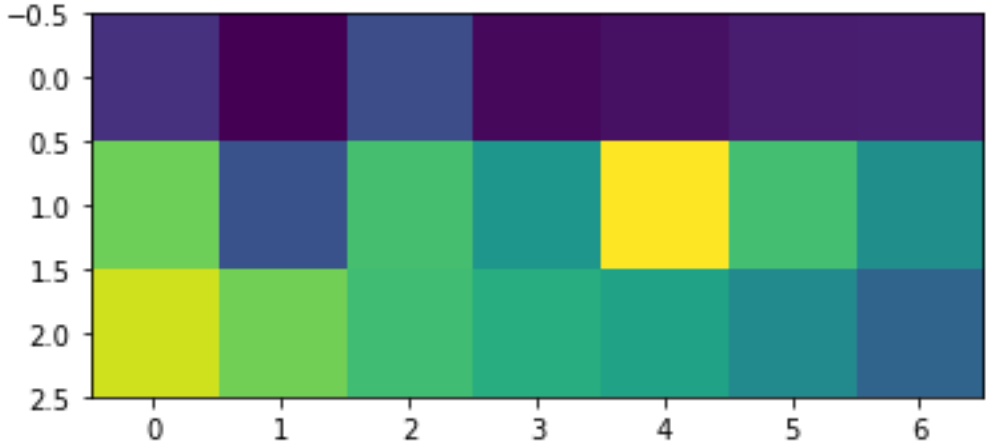
Normal

(normalized)

2 c)



L.



not really similar (unless x and z are flipped)

$\hat{L}$

Instead of calculating $\hat{L}$ and $\hat{B}$ the way in (a), with the same U, S, V

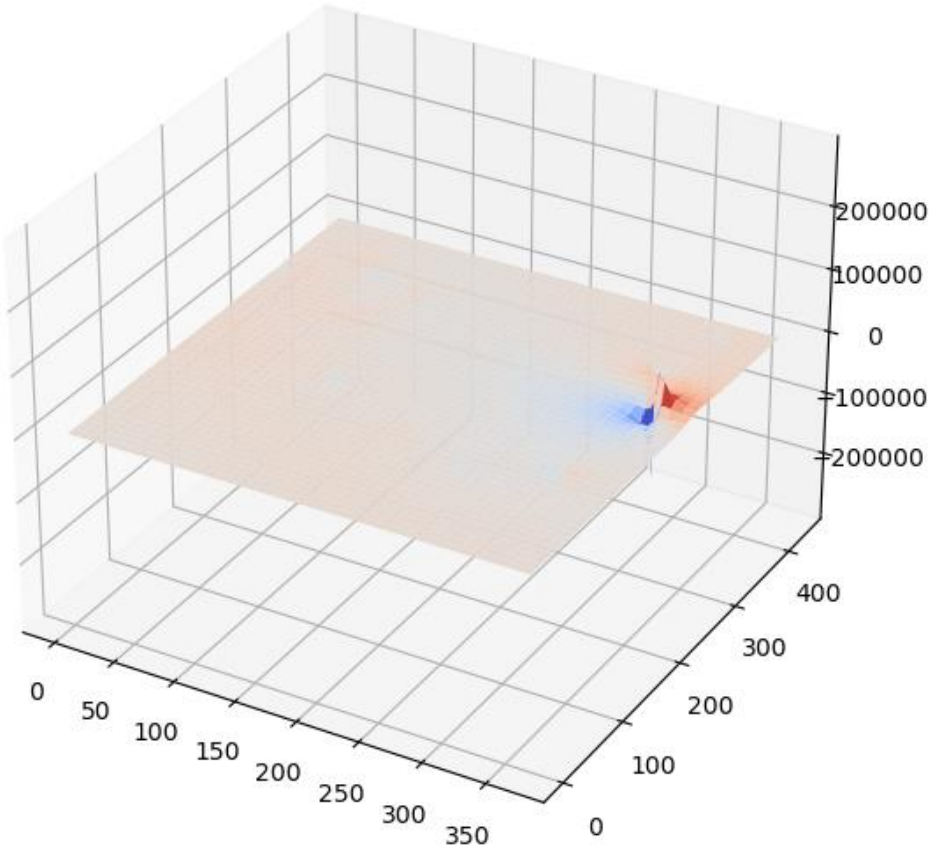
$$\hat{L} = US \quad , \hat{B} = V^T$$

This would change $\hat{L}$ and $\hat{B}$ but keeps Image rendered the same.

or $\hat{L} = U \quad , \hat{B} = \Sigma V^T$

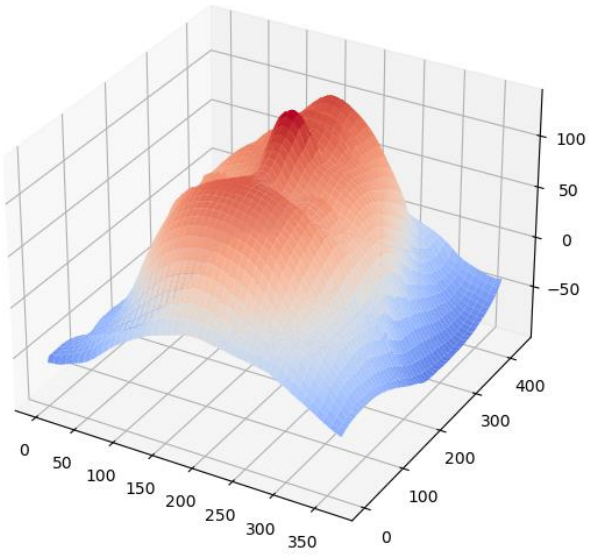
More generally, if you change the sources $\hat{L} \rightarrow A\hat{L}$, you can appropriately change sendonormals by $\hat{B} \rightarrow A^{-T}\hat{B}$ to get the same Images. We can only determine surface normals upto an invertible linear transform.

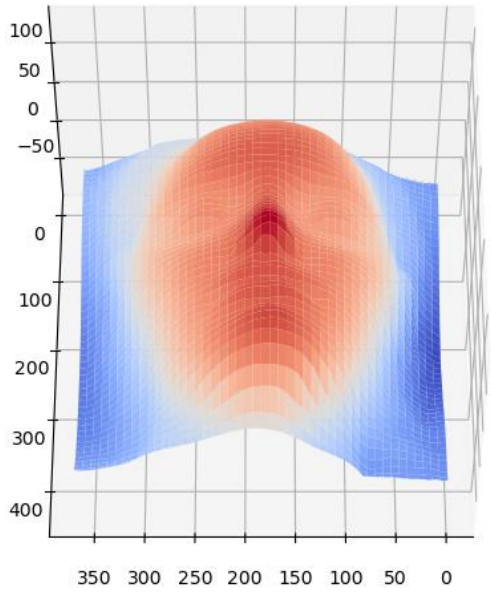
2 d)

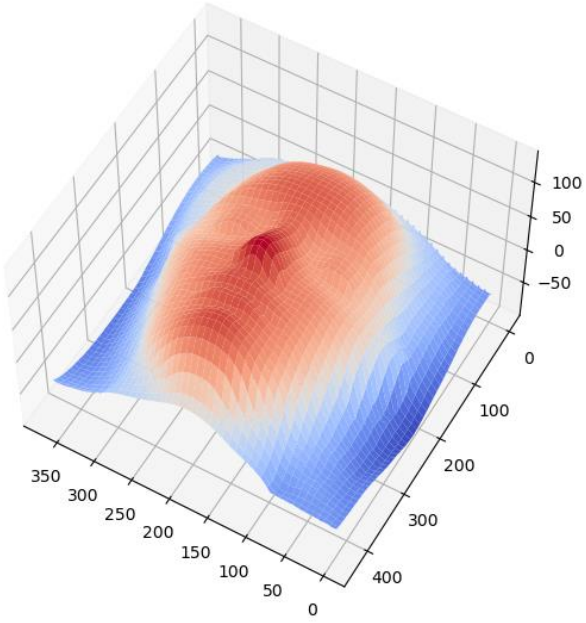


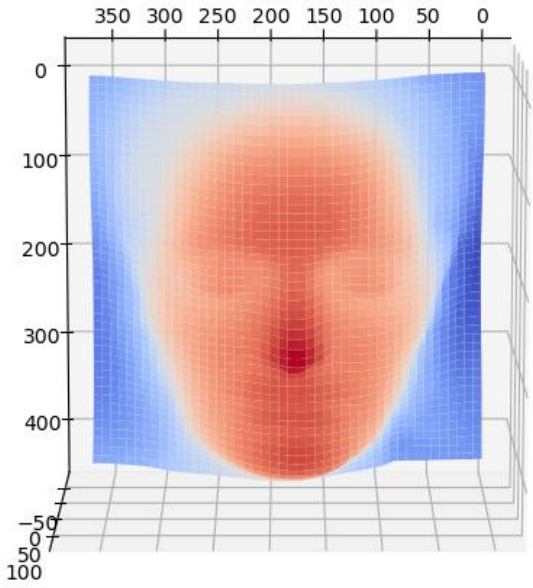
Does not look like a face

2 e)



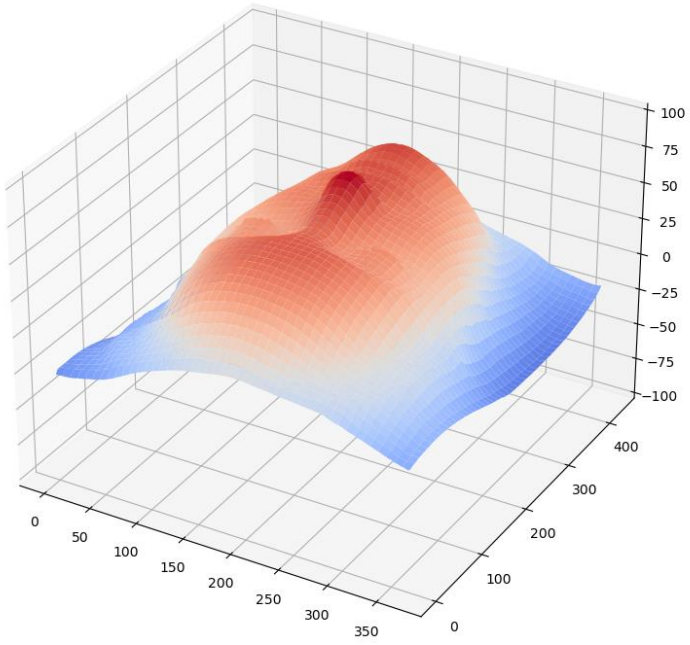


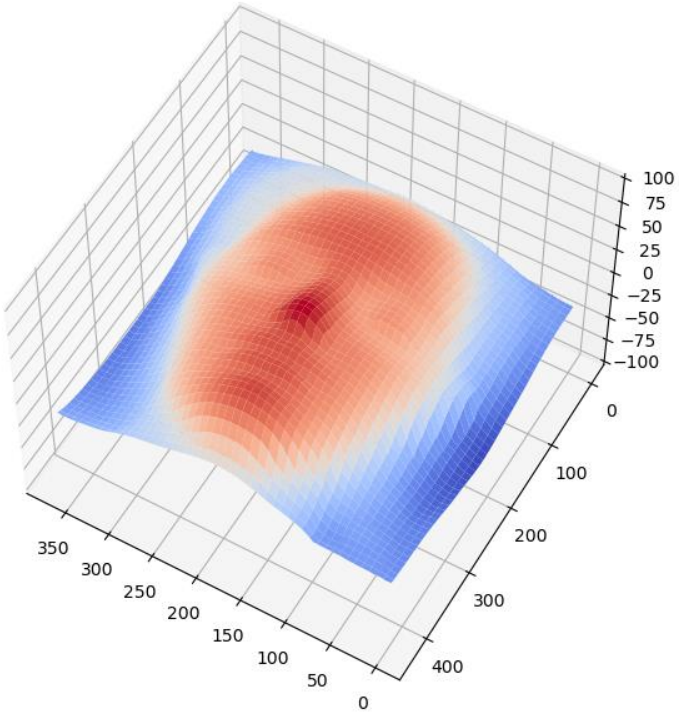




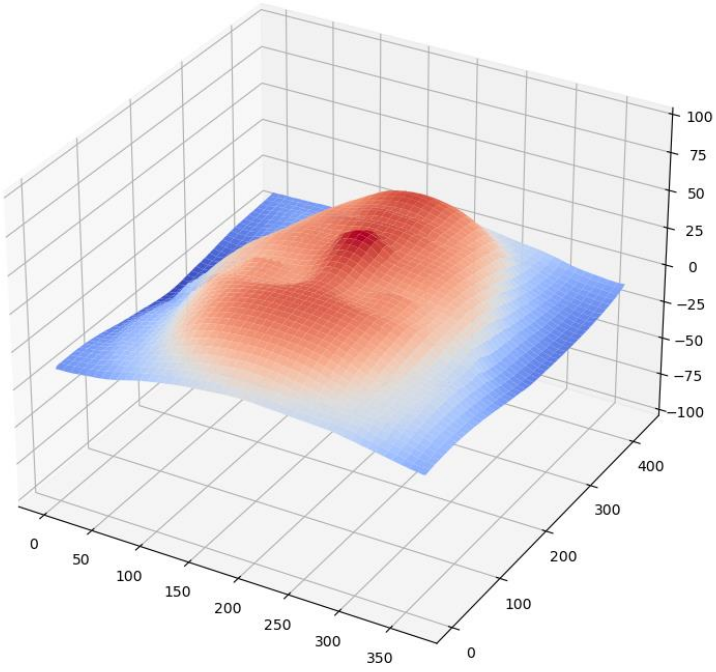
Yes, this is very similar to the one estimated earlier

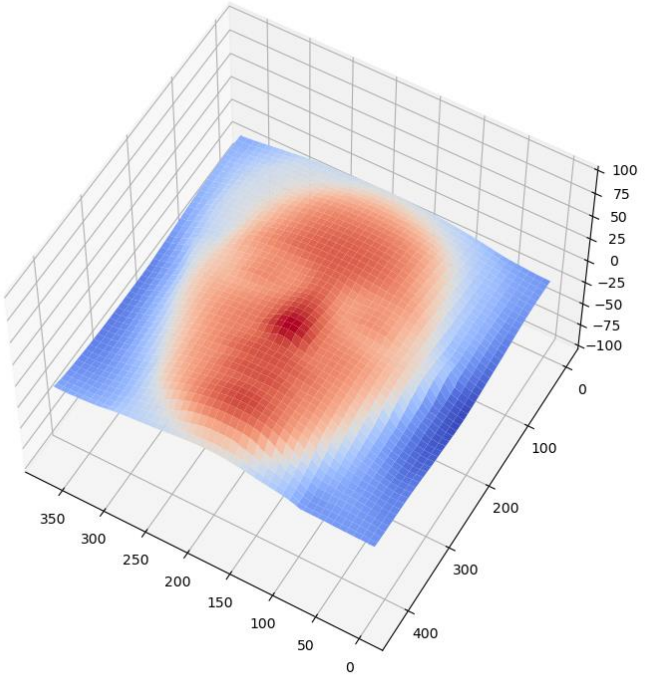
2 f) $\mu = 0, v = 0, \lambda = 0.5$





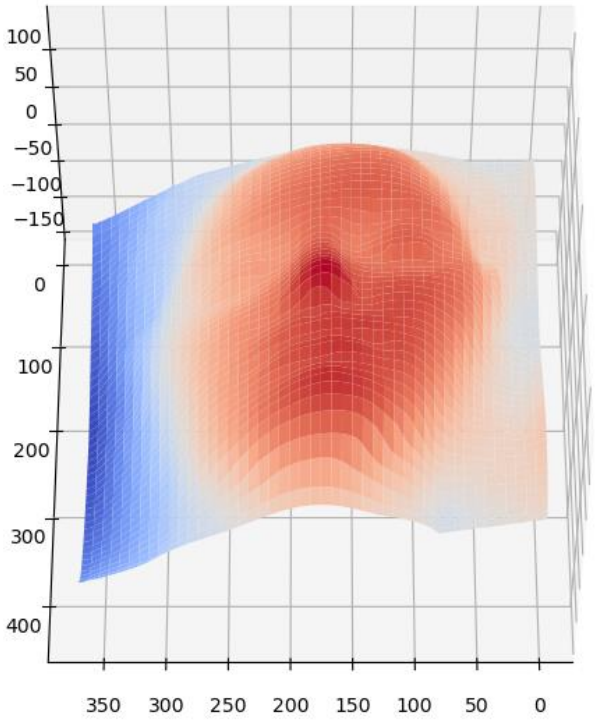
$\mu = 0, v = 0, \lambda = 0.25$

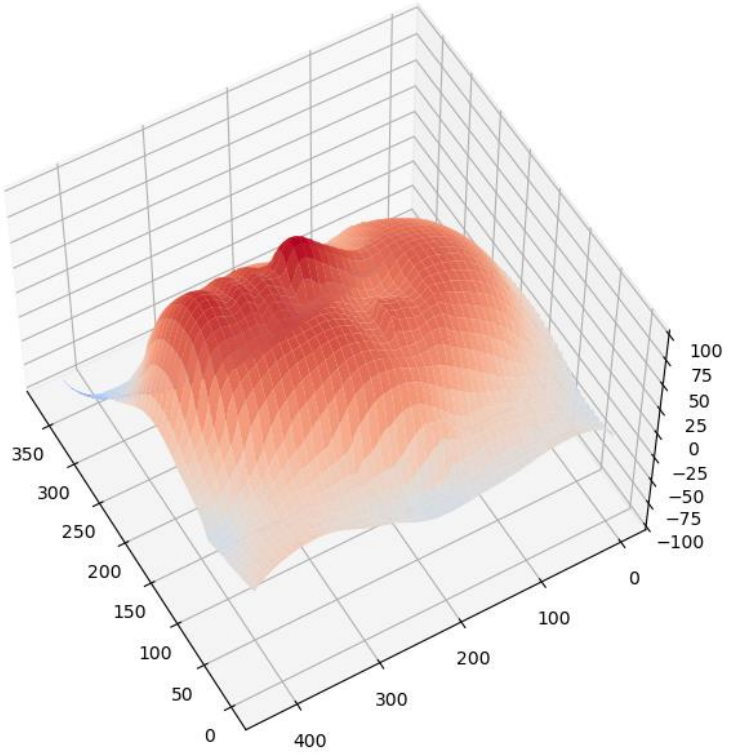




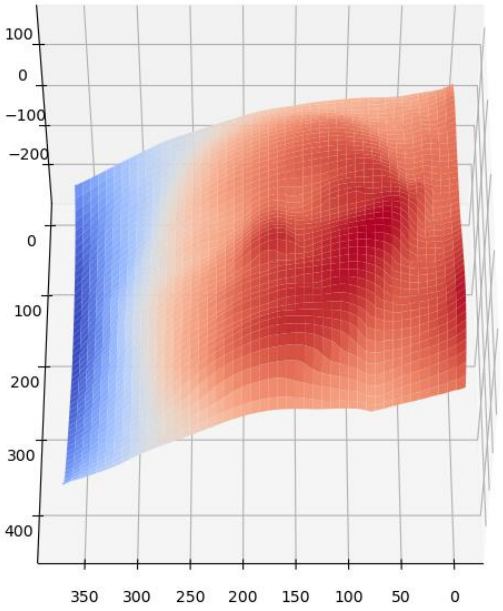
Here by varying the value of λ we see that the depth of the surface changes, as all the z values are scaled by λ. It flattens it and elongates according to the λ value.

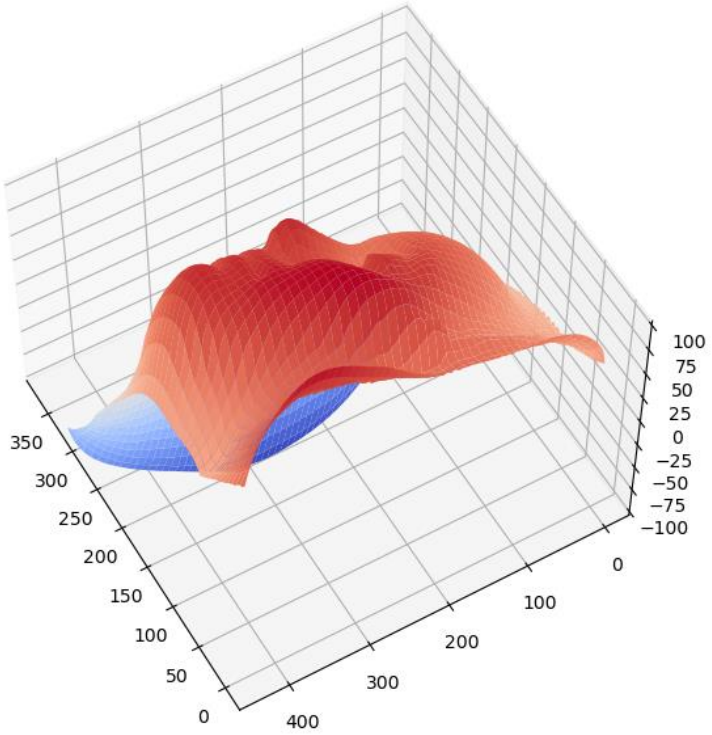
$\mu = 2, v = 0, \lambda = 1$



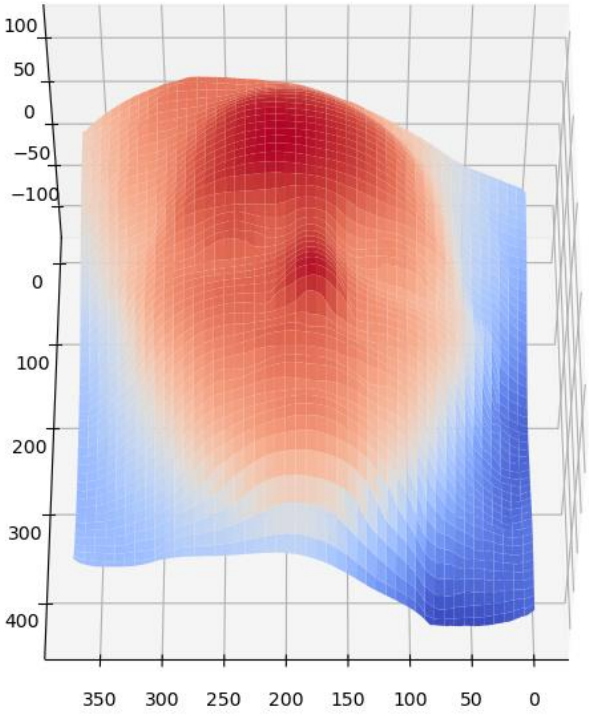


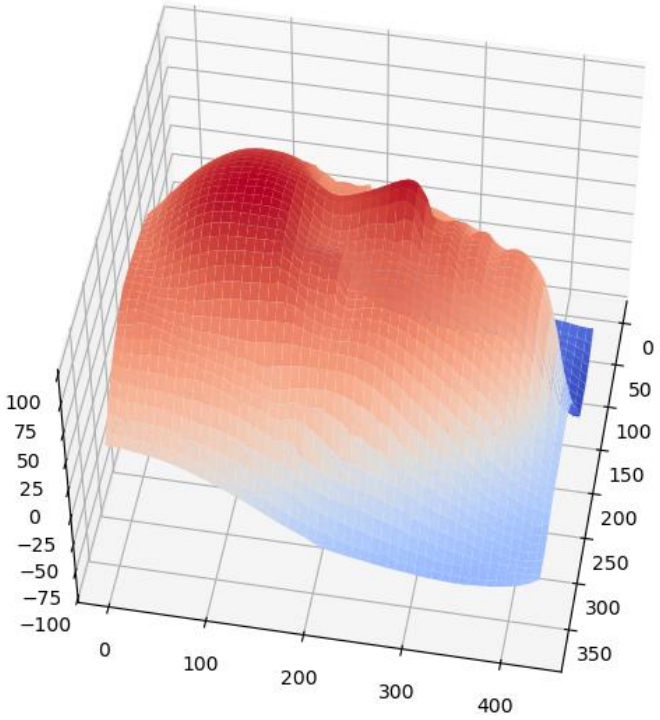
$\mu = 5, v = 0, \lambda = 1$



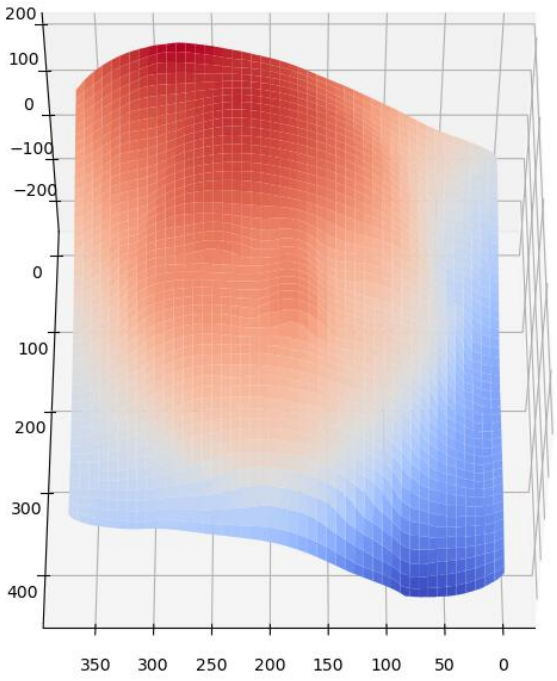


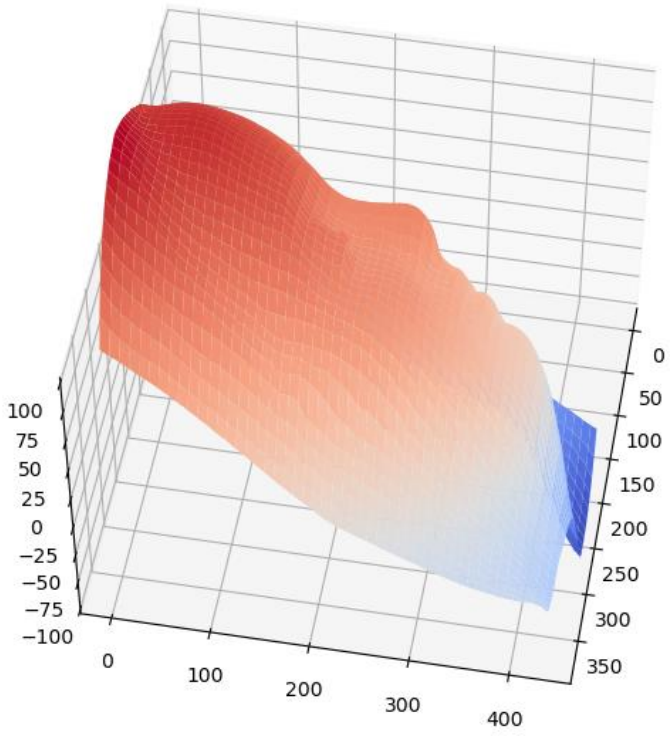
$\mu = 0, v = 2, \lambda = 1$





$\mu = 0, v = 5, \lambda = 1$





Bas relief ambiguity or low relief is named as such since it projects the only slightly from the ground while giving the effect of the original shape when viewed from a particular vantage point, when the parameters are adjusted accordingly. The Bas reliefs are usually defined when $\lambda < 0.5$, or < 1, which produces that flattened surface.

$$f(x; y) = \lambda f(x; y) + \mu x + \nu y$$

λ: This affects the depth of the surface. A smaller lambda value would scale the z values to be smaller, making the surface flatter, whereas a larger value for increase the height of the surface, or elongate it if viewed from the side.

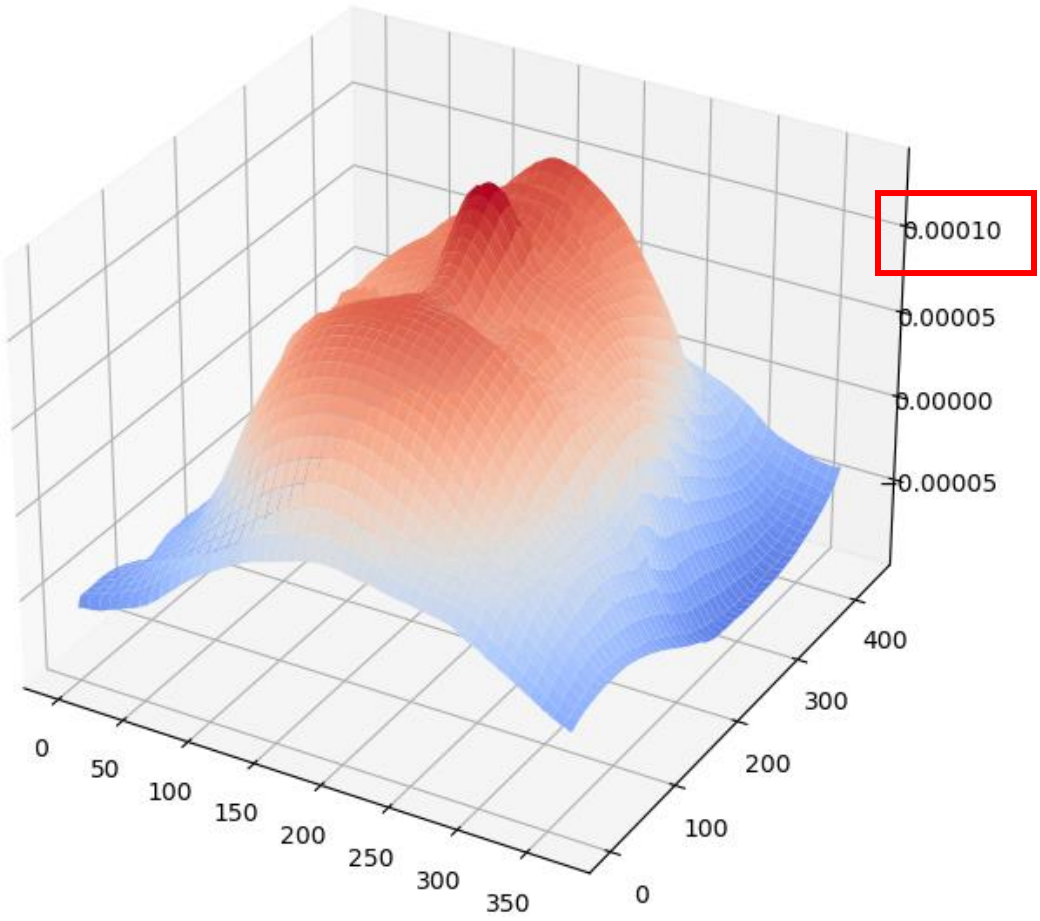
μ: This skews the surface in the x direction, elongating it and tilting the plane based on the value. The x gradient increases.

ν: This skews the surface in the y direction, elongating it and tilting the plane based on the value. The y gradient increases.

2 g)

To make the surface as flat as possible, you would have to reduce the λ value as that scales the z component of the surface. We could keep reducing it until the shape features are preserved.

I tried reducing it until $\lambda = 0.000001$ which produces the following result



2 h)

It is not necessary that with more light sources we will be able to resolve the ambiguity. Since the uncalibrated lighting directions only determine the surface normals up to an invertible linear transform.

Even though increasing the lighting sources and images could affect the singular values, which could potentially render a more accurate reconstruction, there would always exist an ambiguity A. which when applied to the light sources and the pseudo normals by the inverse transpose which would yield the same rendering.