Q1. a

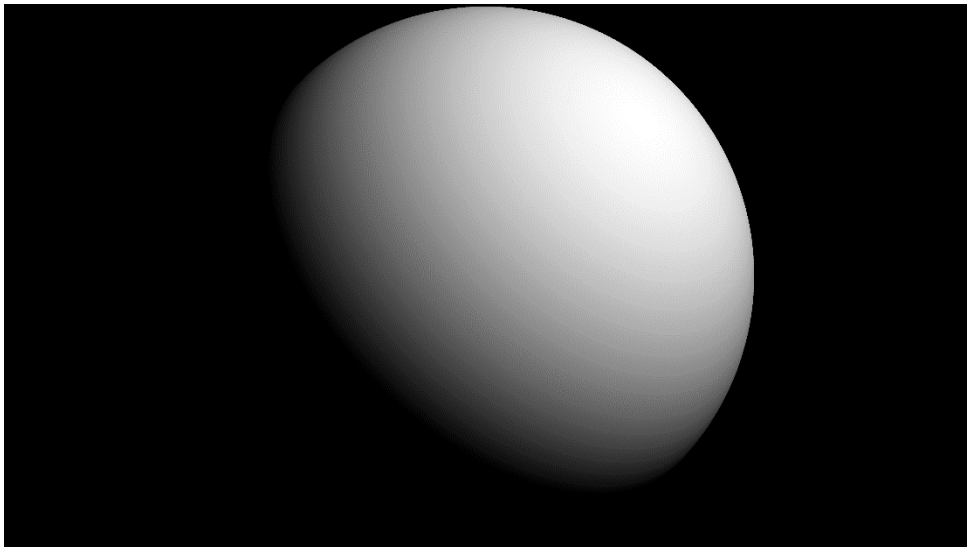
In Fig. 2a, vector n refers to surface normal vector, vector l is light source vector and vector v is viewing direction. So n-dot-l dot product is the dot calculation between vector n and l. In terms of n and l are normalized, this dot product is cosine value of the angle between n and l.

The projected area can be revealed from Lambertian BRDF equation. We can see that the projected area will become larger when L gets closer to the normal of original area. So $L = (\rho d/\pi)ICos(\theta)$, where $Cos(\theta)$ equals n-dot-l product.

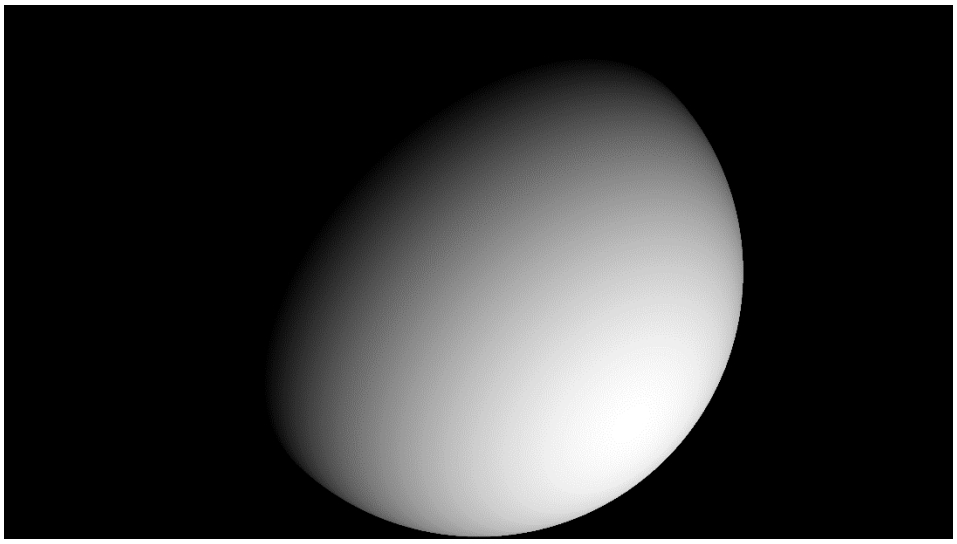
Viewing direction doesn't matter because we assume the surface is isotropic BRDF, which means BRDF of this Lambertian surface is a constant.

Q1.b

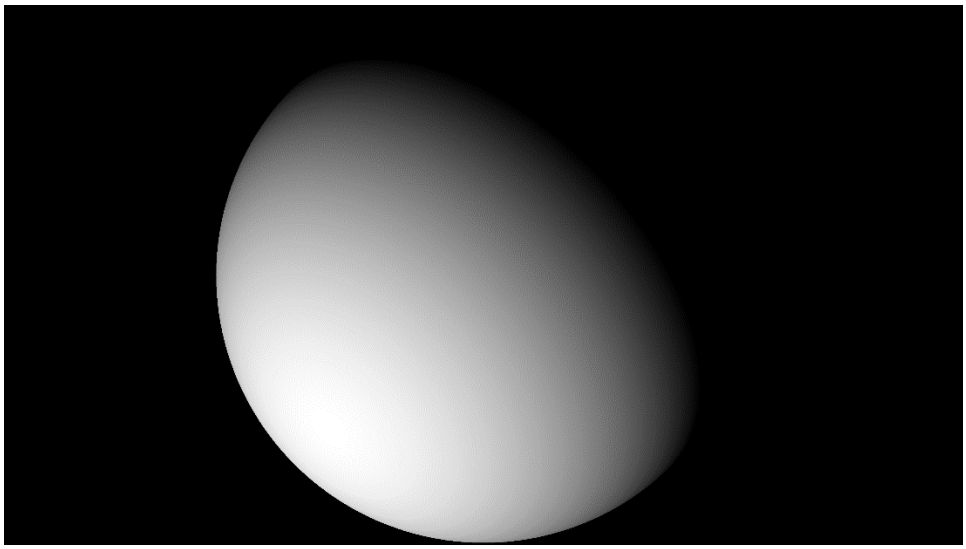
1b-a:



1b-b:



1b-c:



Q1.c

```
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.

    Parameters
    ---------
    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

    """

    # Your code here
    I = []
    for i in range(7):
        img = skimage.io.imread("../data/input_" + str(i + 1) +
".tif")
        img_rgb = rgb2xyz(img)
        img_y = img_rgb[:, :, 1]
        I.append(img_y.flatten())

    I = np.array(I)
    L = np.load(path + "sources.npy")
    L = L.T
    s = img.shape[:2]
    return I, L, s
```

Q1.d

The shape of I should be 7Xp. The rank of I should be at most 3 because the ranks of both B and L are at most 3.

The singular values of I:

[79.36348099 13.16260675 9.22148403 2.414729 1.61659626 1.26289066 0.89368302]

The singular values don't agree with the rank-3 requirement. The real rank is 7 because of noises in real life pictures.

Q1.e

I = L.T x B and y = Ax

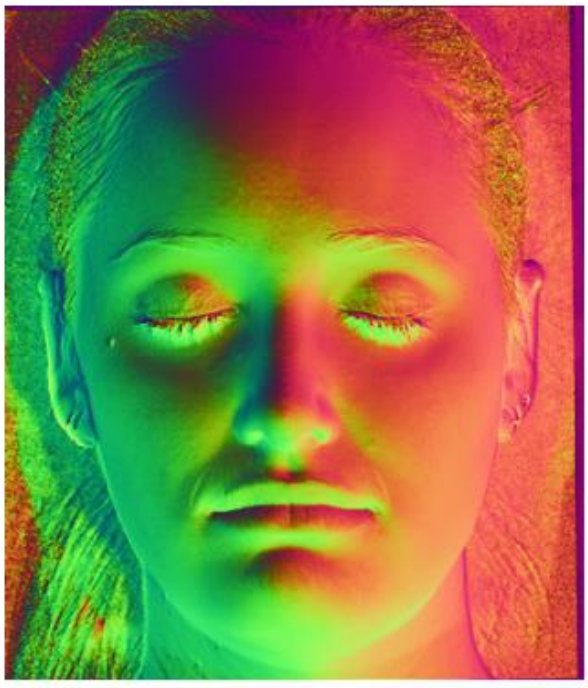
So I set I as y and A as L.T.

$A = [s1^T\ s2^T\ s3^T]^T$

Y is the values in illuminance channel of images.

Q1.f

This is the albedo image and we can find some parts of body are brighter than other parts, including ears, the bottom of nose and top of neck. One reason may be these parts can reflect more light because of smoothness or moisture.



This is the normal image and it basically matches my expectation of the curvature of the face. Under the rainbow colormap, color transition seems more natural.

Q1.g

The whole surface is represented by x, y and z.

So the surface derivative in direction x is $\partial z / \partial x$

So the surface derivative in direction y is $\partial z / \partial y$

Similarly, given the direction as $(-a, -b, 1)^T$, we can represent n as:

$n1 = (-a)^T / \text{sqrt}(a^2+b^2+1)$

$n2 = (-b)^T / \text{sqrt}(a^2+b^2+1)$

$n3 = 1 / \text{sqrt}(a^2+b^2+1)$

So $fx = \partial z / \partial x = -n1/n3$

$fy = \partial z / \partial y = -n2/n3$

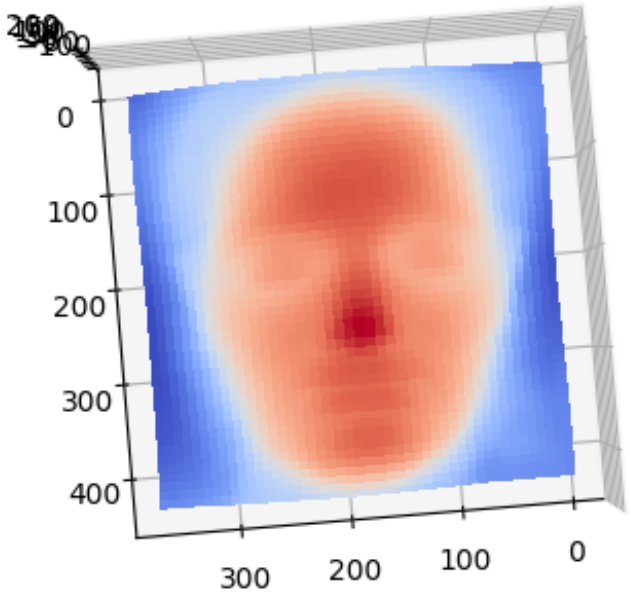
Q1.h

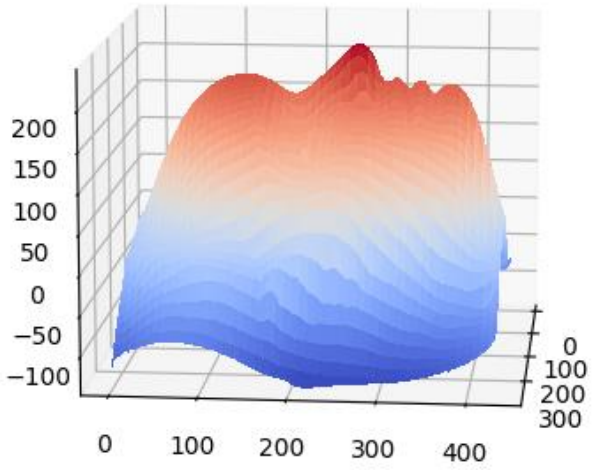
If we use g(0, 0)=1 to reconstruct the entire of g, the two procedures will produce the same result.

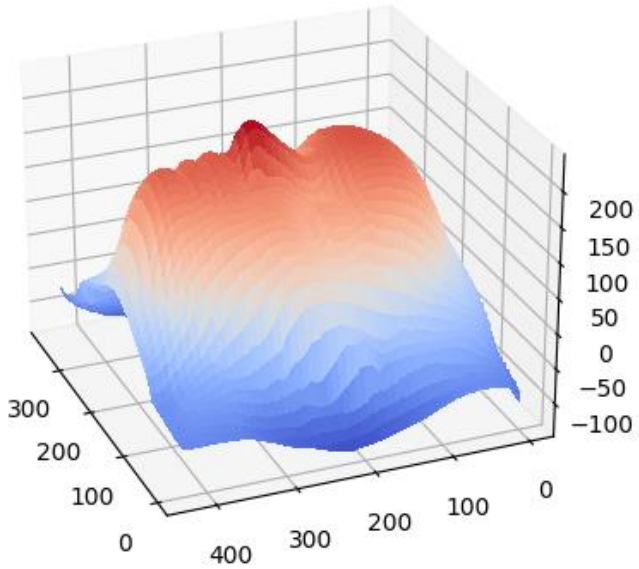
1. Use gx to calculate the first row: [1 2 3 4], then use gy to calculate the rest rows: [5 6 7 8], [9 10 11 12], [13 14 15 16].
2. Use gy to calculate the first column: [1 5 9 13], then use gx to calculate the rest columns: [2 6 10 14], [3 7 11 15], [4 8 12 16].

If we modify g(0, 0) to 0 instead of 1, the two methods will produce different results.
The noises and bias in real world images can make some values imprecise and gradients non-integrable.

Q1.i


0
100
200
300
400
300
200
100
0


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

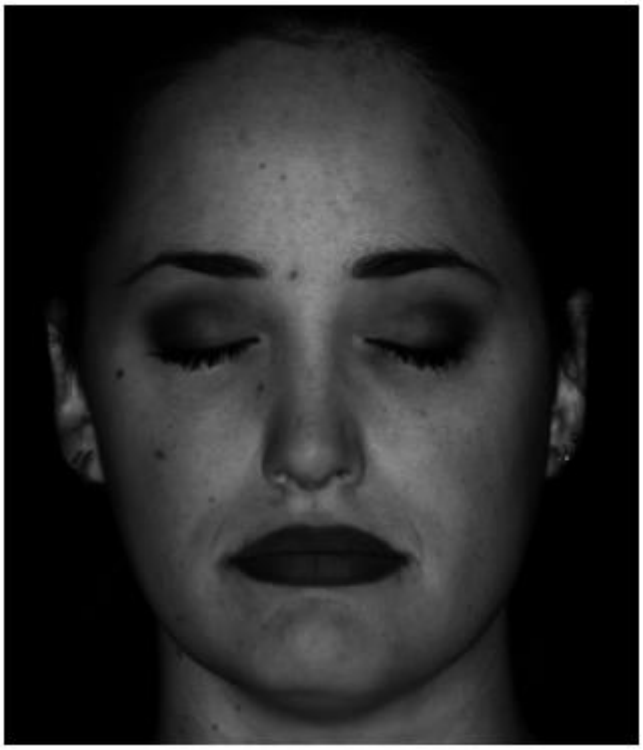

300
200
100
0
400
300
200
100
0
200
150
100
50
0
−50
−100

Q2.a

If we set all singular values except the top k from Σ to 0, then the new singular values vector has rank k exactly. In this way, we can have a new matrix M with exact rank k by multiplying this new singular values vector.

So Lhat = Uhat * sqrt(Σhat), Bhat = sqrt(Σhat) * Vhat.T

Q2.b

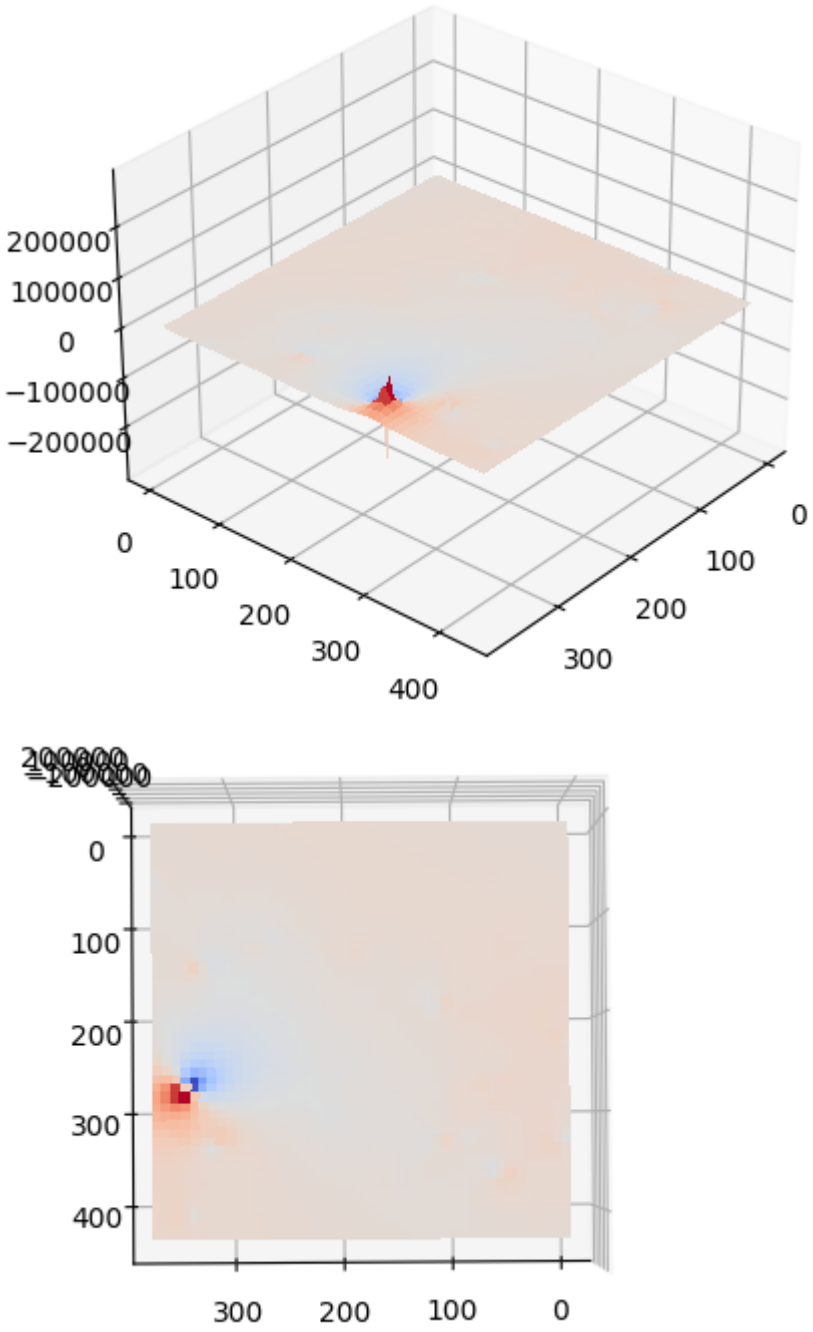


Q2.c

```
L0 = [[-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]]
Lhat = [[-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 ]]
```

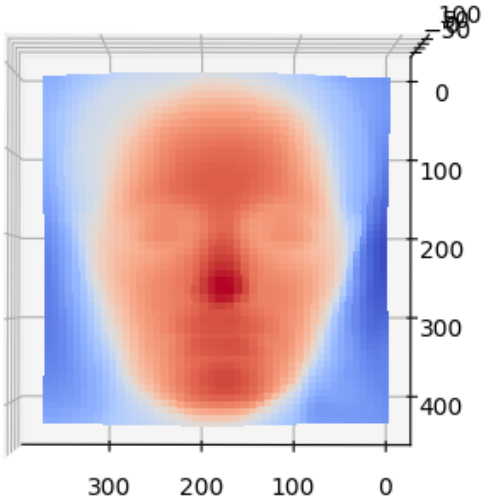
They are not the same but similar. We can make a small change to calculate Lhat and Bhat. For example, we can assign sqrt(Σhat) unevenly, such as Σhat^1/3 andΣhat^2/3.

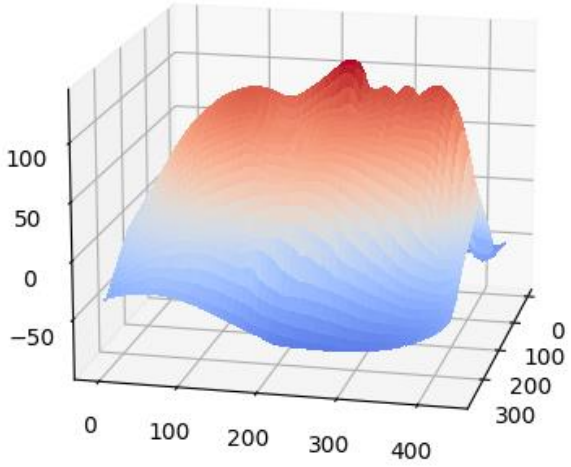
Q2.d

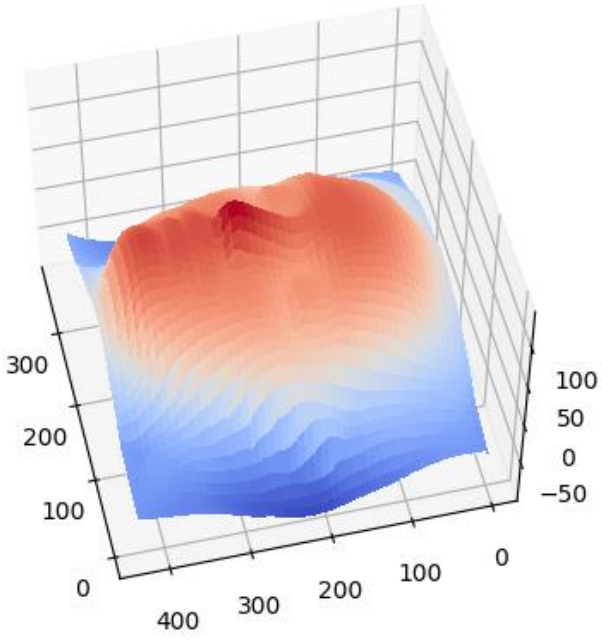


This doesn't look like a face.

Q2.e



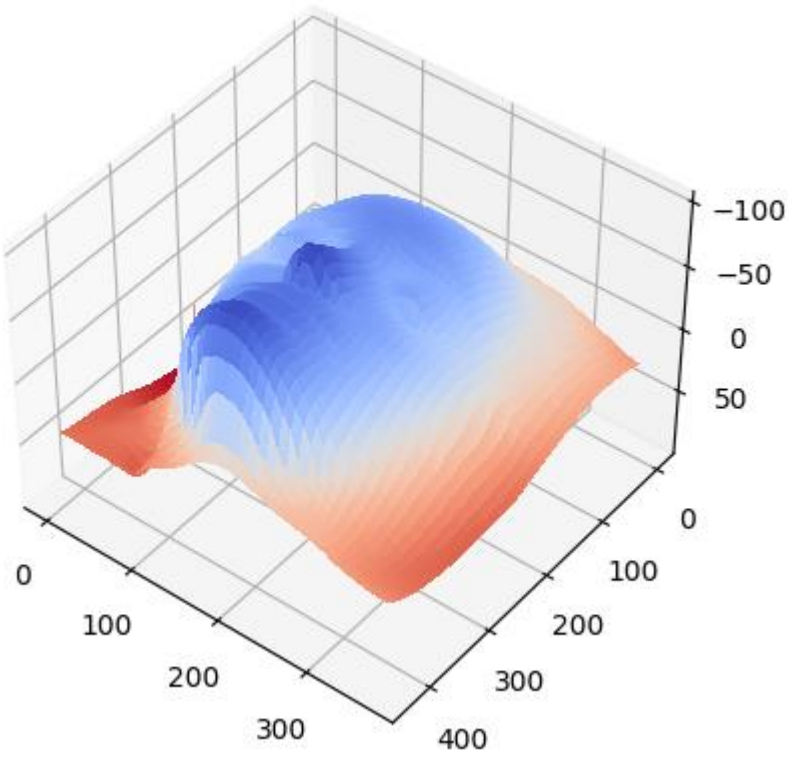




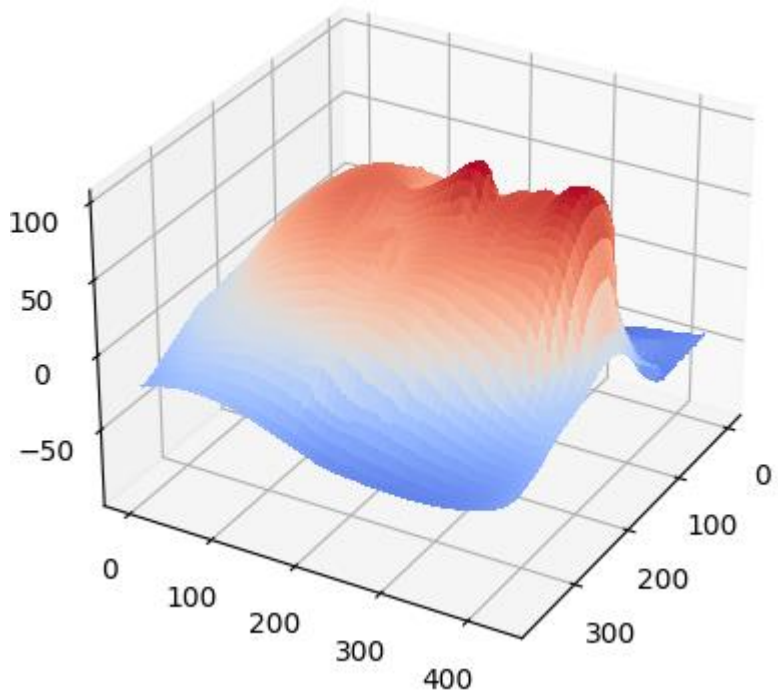
This surface looks like the one output by calibrated photometric stereo.

Q2.f

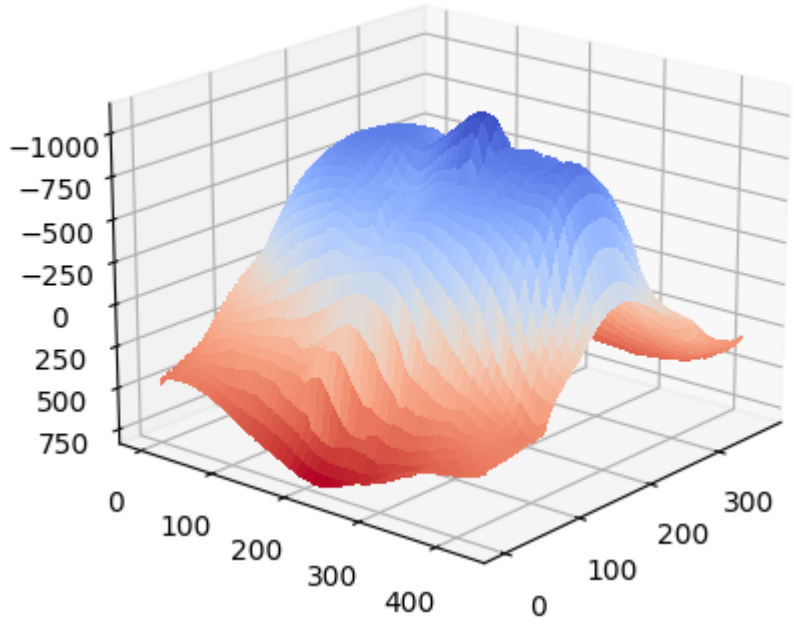
μ=0, ν=0, λ=-10



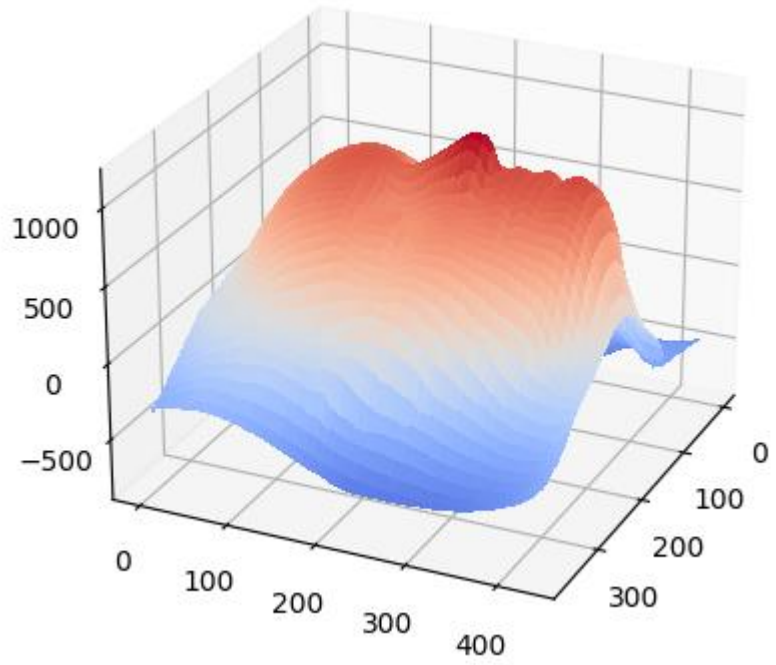
μ=0, ν=0, λ=10



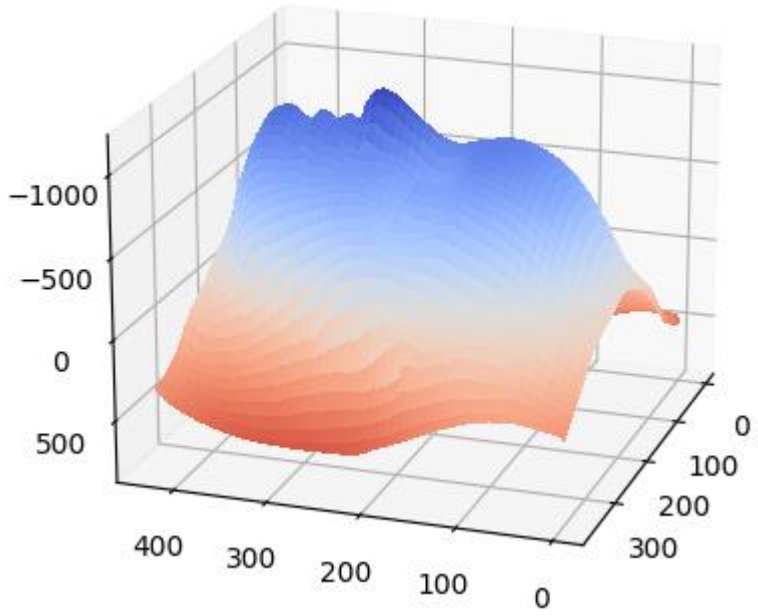
μ=0, ν=-10, λ=1



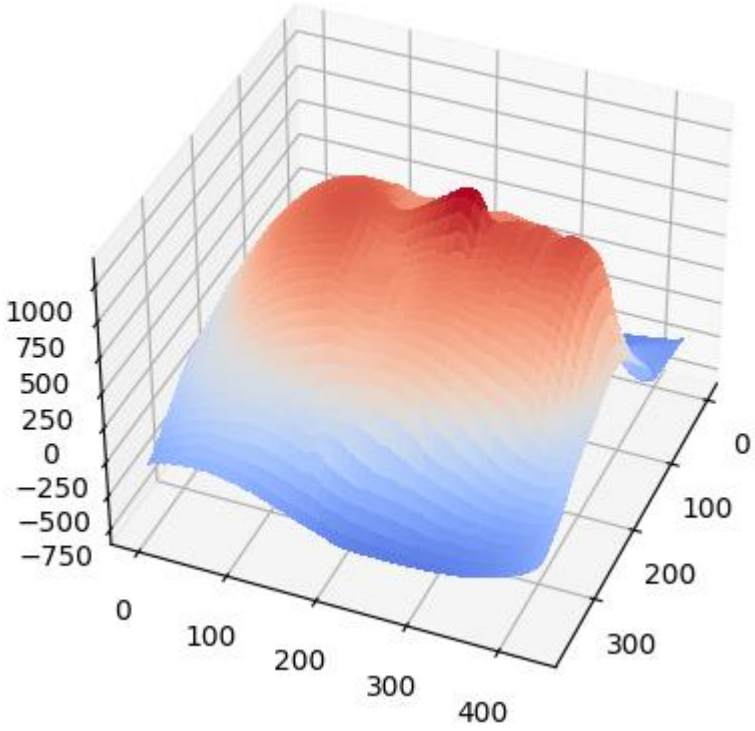
μ=0, ν=10, λ=1



μ=-10, ν=0, λ=1



μ=10, ν=0, λ=1



Because bas-relief means low-relief, flatten models created in our ways can be distorted. However, lighting can make up for this kind of distortions in real life.

μ can affect x of depth map, and the whole image will flatten when μ increases.

ν can affect y of depth map, and the whole image will flatten when ν increases.

λ can affect z of depth map, and the whole image will flatten when ν decreases.

Q2.g

As I mention in the last question, the smaller of λ, the flatter of the surface. So we should set λ as small as possible.

Q2.h

No, the ambiguity can only be solved entirely if we know the exact light source direction.