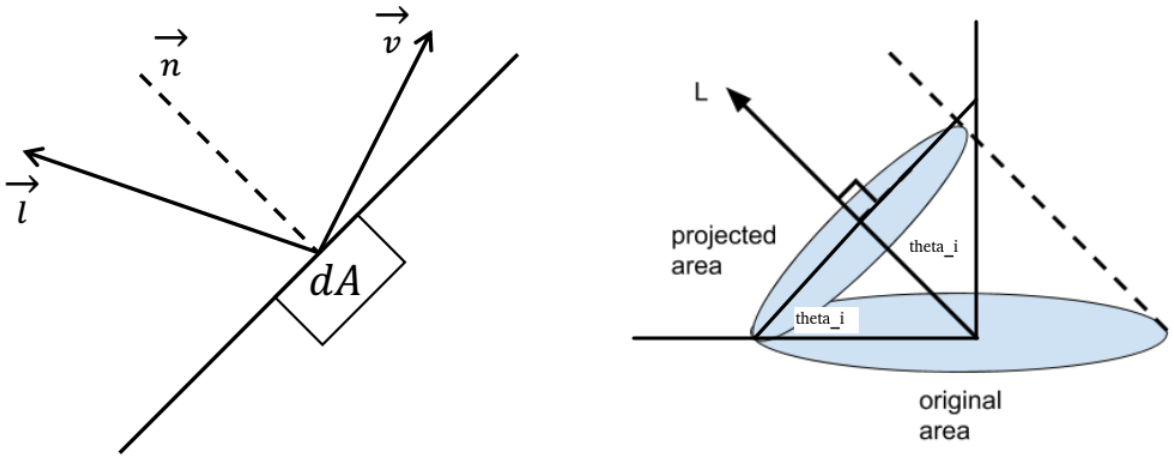
16-820 Advanced Computer Vision Assignment 5

Li-Wei Yang
liweiy@andrew.cmu.edu
Section A

Q1.a



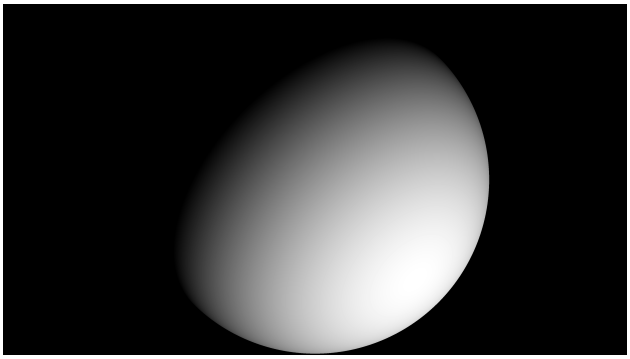
(a) Geometry of photometric stereo (b) Projected area

n-dot-l model, $L = \frac{\rho_d}{\pi} I_0 \cos\theta_i = \frac{\rho_d}{\pi} \vec{n} \cdot \vec{l}$, the dot product comes from the projection of light direction, $\vec{l}$, on to normal direction, $\vec{n}$.
From figure (b), we can see the area of projected area,A', is $A\cos\theta_i$. Thus $\cos\theta_i = A'/A$, substitute back to equation, $L = \frac{\rho_d}{\pi} I_0 \frac{A'}{A}$, $L = \frac{\rho_d}{\pi} I_0 \frac{A'}{A}$, so surface brightness is not a function of viewing direction, thus viewing direction does not matter.

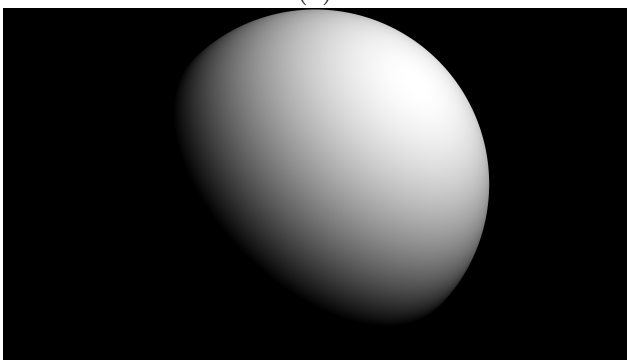
Q1.b, collaborator: Qilin Wu

```
def renderNDotLSphere(center, rad, light, pxSize, res):
    """
    Question 1 (b)

    Render a hemispherical bowl with a given center and radius. Assume that
    the hollow end of the bowl faces in the positive z direction, and the
    camera looks towards the hollow end in the negative z direction. The
    camera's sensor axes are aligned with the x- and y-axes.

    Parameters
    ----------
    center : numpy.ndarray
        The center of the hemispherical bowl in an array of size (3,)

    rad : float
        The radius of the bowl

    light : numpy.ndarray
        The direction of incoming light

    pxSize : float
        Pixel size

    res : numpy.ndarray
        The resolution of the camera frame

    Returns
    -------
    image : numpy.ndarray
        The rendered image of the hemispherical bowl
    """

    [X, Y] = np.meshgrid(np.arange(res[0]), np.arange(res[1]))
    X = (X - res[0] / 2) * pxSize * 1.0e-4
    Y = (Y - res[1] / 2) * pxSize * 1.0e-4
    Z = np.sqrt(rad**2 + 0j - X**2 - Y**2)
    X[np.real(Z) == 0] = 0
    Y[np.real(Z) == 0] = 0
    Z = np.real(Z)

    coordinates = np.stack((X, Y, Z), axis=2).reshape(res[1]*res[0], -1)
```

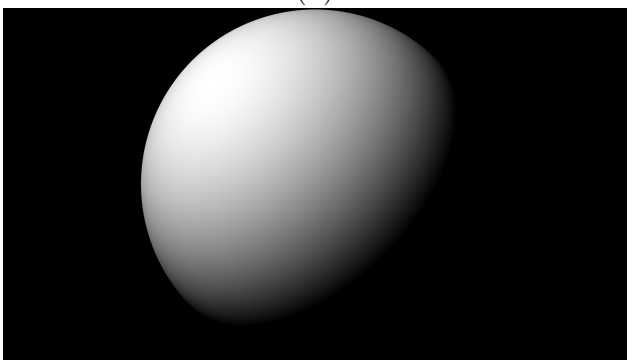
```
image = np.dot(coordinates, light).reshape(res[1], res[0])
image = np.clip(image, 0, 1)

return image
```



(a)



(b)



(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

    """

    for i in range(1, 8):
        im = imread(path + "input_" + str(i) + ".tif", dtype='uint16')
        im_xyz = rgb2xyz(im)
        lumi = im_xyz[:, :, 1]
        if i == 1:
            s = im.shape[:2]
            I = np.zeros((7, s[0]*s[1]))
        I[i - 1, :] = lumi.flatten()

    L = np.load(path + "sources.npy").T
    return I, L, s
```

Q1.d

A surface normal (or pseudo normal) in 3D space would have 3 DoF, thus we will need I to be at least rank 3 to uniquely define B.

The sigular values of **I** are:
79.36348099, 13.16260675, 9.22148403, 2.414729, 1.61659626, 1.26289066, 0.89368302.

The singular values are not agree with rank-3 requirement, because we have more constraint than unknowns. Only first three singular values are with large value implies they are more important, and it means most of B can be decide just using the first three values.

```
# Part 1(d)

U, S, Vh = np.linalg.svd(I, full_matrices=False)
print(U.shape, S.shape, Vh.shape)
print(S)
```

Q1.e

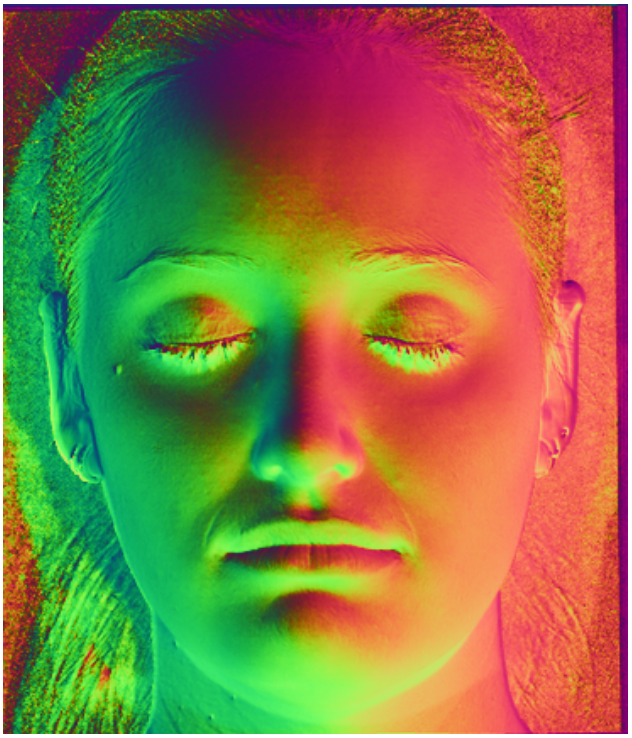
From $\mathbf{I} = \mathbf{L}^T\mathbf{B}$, we have two known $\mathbf{I}$ and $\mathbf{L}^T$. We can construct the linear system $\mathbf{Ax} = \mathbf{y}$, where $\mathbf{x} = \mathbf{B}$, $\mathbf{A} = \mathbf{L}^T$, and $\mathbf{y} = \mathbf{I}$. Solve $\mathbf{B}$ using least squares method.

```
def estimatePseudonormalsCalibrated(I, L):
    """
    Question 1 (e)

    In calibrated photometric stereo, estimate pseudonormals from the
    light direction and image matrices

    Parameters
    ----------
    I : numpy.ndarray
        The 7 x P array of vectorized images

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

    Returns
    -------
    B : numpy.ndarray
        The 3 x P matrix of pesudonormals
    """

    B = np.linalg.lstsq(L.T, I, rcond=None)[0]
    return B
```

Q1.f

albedo

Some unusual features are the brightness of entire figure is darker than I thought, this means people's skin consumes most of the light and does not reflect that much. There are also some bright area around the nose and ears, this may be resulted from the shadows. Because our model does not include the effect of shadows, when there are shadows we will lost information and hard to calculate albedo.



normal

The normal on the nose and lips have some abrupt changes, this is the transition of normal. Also the normal on the forehead is not symmetric, this may be caused by the bias of our lighting angle, we may have more pictures have light coming from the right instead of left. Other region seems align with my expectation of the curvature of a face.

```
def estimateAlbedosNormals(B):
    """
    Question 1 (e)

    From the estimated pseudonormals, estimate the albedos and normals

    Parameters
    ----------
    B : numpy.ndarray
        The 3 x P matrix of estimated pseudonormals

    Returns
    -------
    albedos : numpy.ndarray
        The vector of albedos

    normals : numpy.ndarray
        The 3 x P matrix of normals
```

```
    """

    albedos = np.linalg.norm(B, axis=0)
    normals = B/albedos
    # print(albedos.shape, normals.shape)
    return albedos, normals

def displayAlbedosNormals(albedos, normals, s):
    """
    Question 1 (f, g)

    From the estimated pseudonormals, display the albedo and normal maps

    Please make sure to use the ‘gray‘ colormap for the albedo image
    and the ‘rainbow‘ colormap for the normals.

    Parameters
    ----------
    albedos : numpy.ndarray
        The vector of albedos

    normals : numpy.ndarray
        The 3 x P matrix of normals

    s : tuple
        Image shape

    Returns
    -------
    albedoIm : numpy.ndarray
        Albedo image of shape s

    normalIm : numpy.ndarray
        Normals reshaped as an s x 3 image

    """

    albedoIm = albedos.reshape(s)
    normalIm = ((normals + 1)/2).reshape(3, -1)
    normalIm = normalIm.T.reshape(s[0], s[1], 3)
    return albedoIm, normalIm
```

Q1.g

We can calculate the gradient discretely:
$v_x = (1, 0, z_{x+1,y} - z_{x,y})$
using relation of $n \cdot v_x = 0$, we get:
$0 = (n_1, n_2, n_3) \cdot (1, 0, z_{x+1,y} - z_{x,y}) = n_1 + n_3 * (z_{x+1,y} - z_{x,y})$
As a result, $\frac{\partial f(x,y)}{\partial x} = z_{x+1,y} - z_{x,y} = -\frac{n_1}{n_3}$
Similarly, $v_y = (0, 1, z_{x,y+1} - z_{x,y})$
$0 = (n_1, n_2, n_3) \cdot (0, 1, z_{x,y+1} - z_{x,y}) = n_2 + n_3 * (z_{x,y+1} - z_{x,y})$
$\frac{\partial f(x,y)}{\partial y} = z_{x,y+1} - z_{x,y} = -\frac{n_2}{n_3}$

Q1.h

We first calculate the gradient of g:

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

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

Method 1:

$$g_1 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 161 \end{bmatrix}$$

Method 2:

$$g_2 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 161 \end{bmatrix}$$

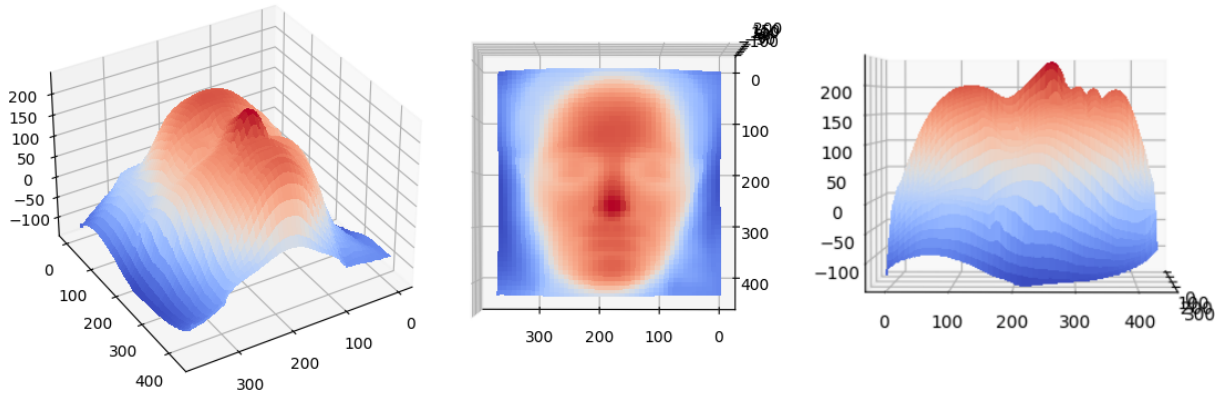
The results are the same, this implies the gradients are integrable, and can correspond to a true surface.

If we want to make the gradient non-integrable, we can have offset either g_x or g_y by a constant. By doing so, there would be inconsistency between g_x and g_y.

Why the method in (g) may produce non-integrable gradients? This is because if the gradient changes abruptly, like in some slope and edges, the gradient would be small in certain direction while be large on the other, this may produce inconsistency and thus make gradients non-integrable.

Q1.i

```
def estimateShape(normals, s):
    """
    Question 1 (j)

    Integrate the estimated normals to get an estimate of the depth map
    of the surface.

    Parameters
    ----------
    normals : numpy.ndarray
        The 3 x P matrix of normals

    s : tuple
        Image shape

    Returns
    ----------
    surface: numpy.ndarray
        The image, of size s, of estimated depths at each point

    """
    fx = (-normals[0, :]/normals[2, :]).reshape(s)
    fy = (-normals[1, :]/normals[2, :]).reshape(s)

    surface = integrateFrankot(fx, fy)
    return surface
```



Q2.a

We can do SVD on $\mathbf{I}$
$\mathbf{I} = \mathbf{U}\Sigma\mathbf{V}^T$
$\mathbf{I}$ will have dimension of 7*P, so $\mathbf{U}$ would be 7*7, $\mathbf{V}$ would be P*P, set all singular values to be 0, except top k values. Finally, choose top 3 rows from $\mathbf{U}$ as $\mathbf{L}$ and top 3 rows from $\mathbf{V}^T$ as $\mathbf{B}$.

Q2.b

```
def estimatePseudonormalsUncalibrated(I):
    """
    Question 2 (b)

    Estimate pseudonormals without the help of light source directions.

    Parameters
    ----------
    I : numpy.ndarray
        The 7 x P matrix of loaded images

    Returns
    -------
    B : numpy.ndarray
        The 3 x P matrix of pseudonormals

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

    """

    U, S, Vh = np.linalg.svd(I, full_matrices=False)
    S[3:] = 0
    B = Vh[:3, :]
    # L = ((U @ np.diag(S)).T)[:3, :]
    L = U[:3, :]

    print(B.shape, L.shape)
    return B, L
```

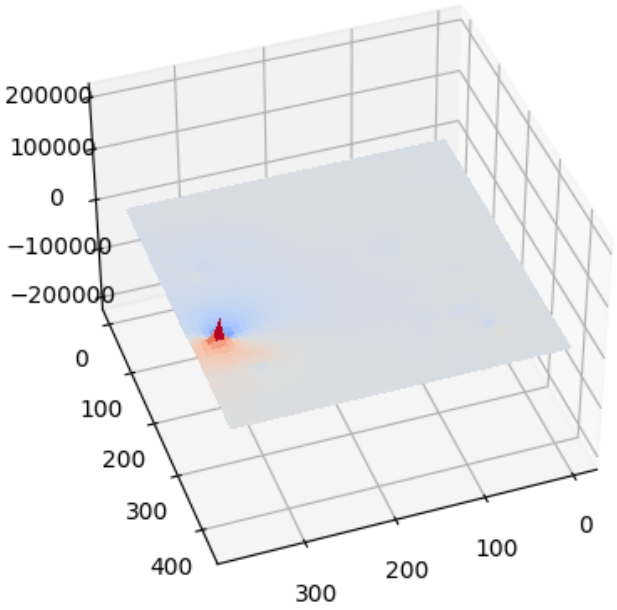
The albedos and normals are visualize on (d).

Q2.c

```
L_0: -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
L_hat: -0.3359  0.2612  0.6189  0.5973 -0.03714  0.2213  -0.1695
       -0.4344 -0.6387  0.3341 -0.3664 -0.3906   0.0377  -0.0598
       -0.2703  0.1376  0.1414 -0.1013  0.1541   0.0382   0.9233
```

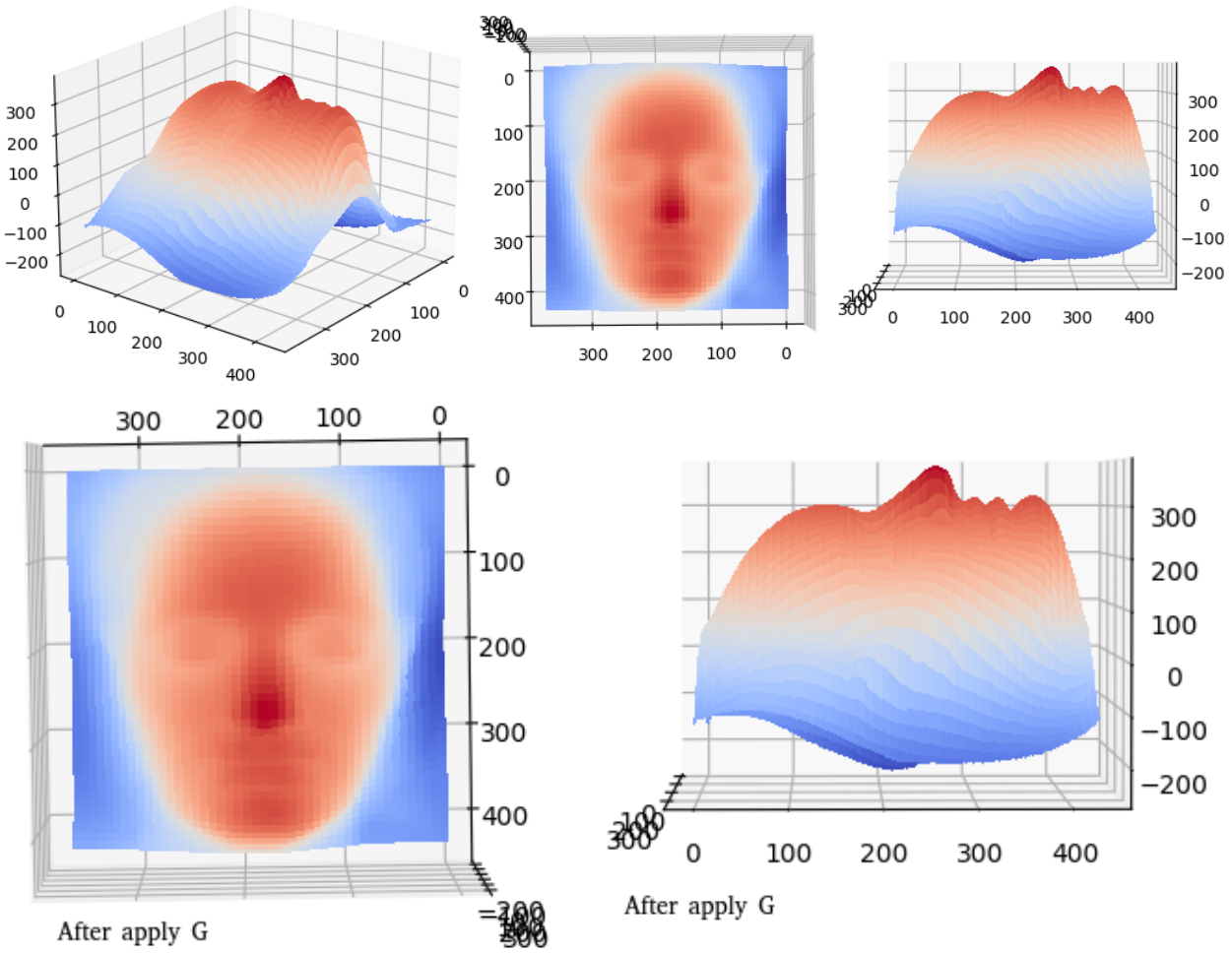
They are far from similar, potential ways to improve result is to normalize $\hat{\mathbf{L}}$ and $\mathbf{B}$, and modify $\mathbf{B}$ with $\mathbf{G}$ in (e).

Q2.d



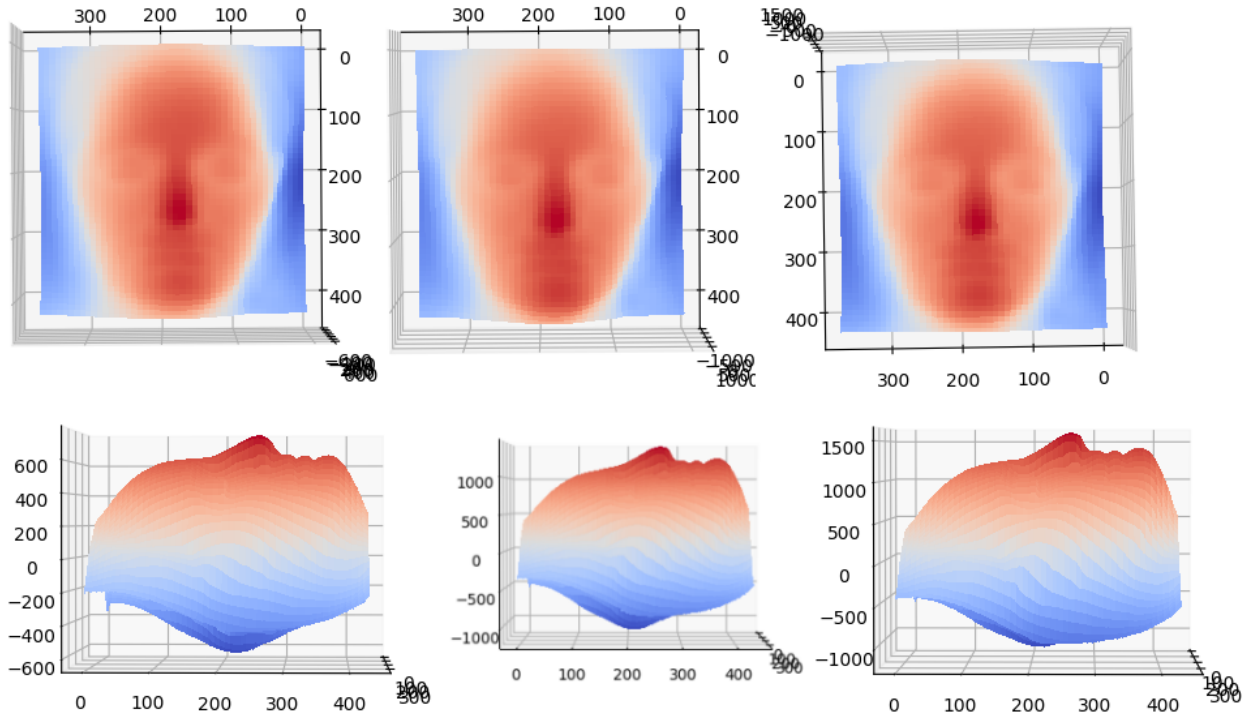
The result does not look like a face at all.

Q2.e

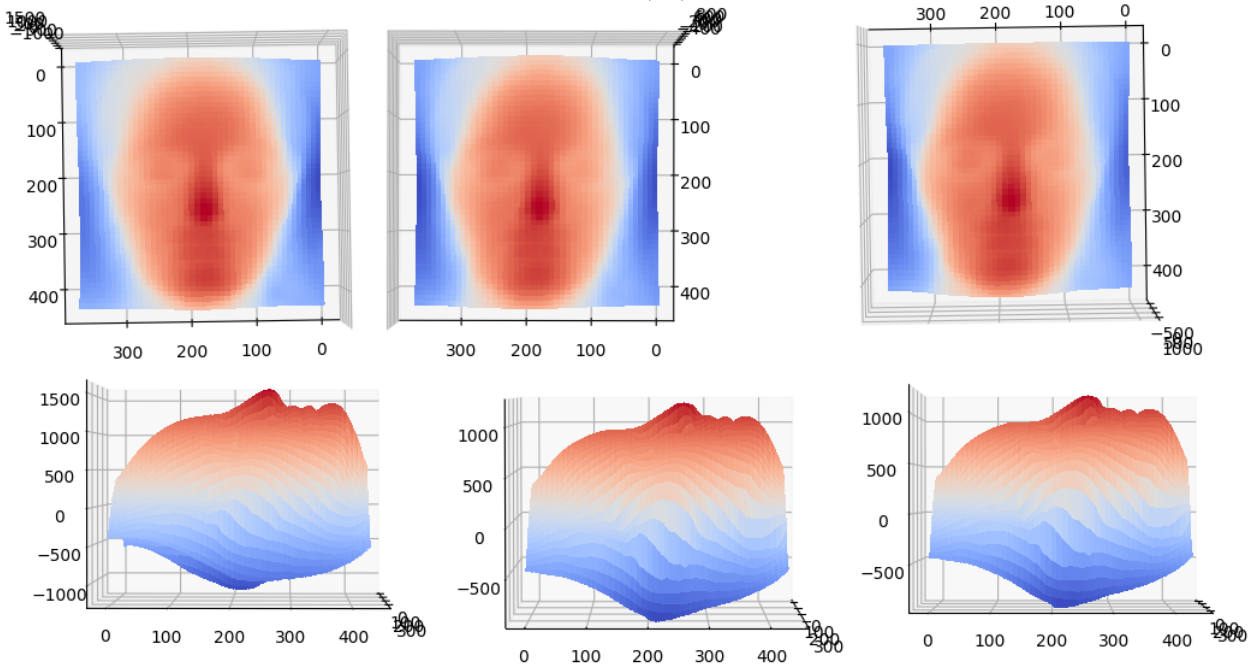


The result looks more similar to the calibrated one.

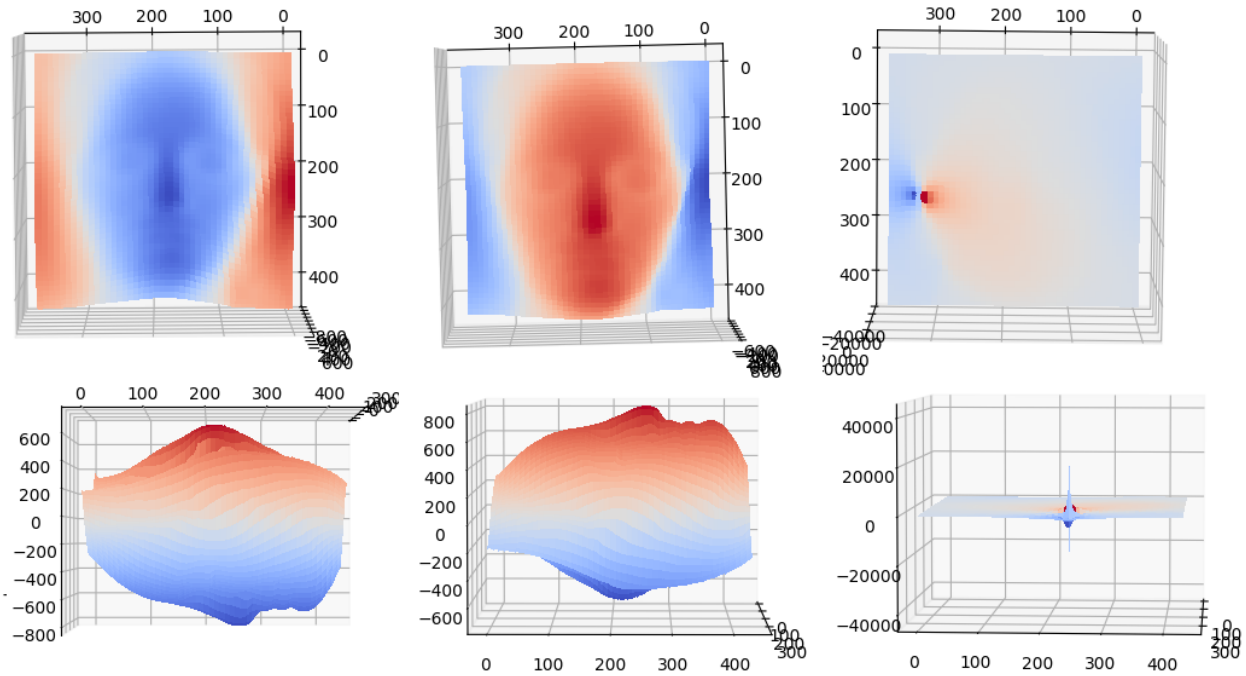
Q2.f



mu = -1, 2, 3



nu = 1, 2, 3



lambda = -1, 1, 3

We are using inverse of G transpose, this is the only way to ensure integrability.

increasing mu increase the height in general, but increase the height less on large x gradient region, like nose.
In the paper it should adjust the z value with mu proportion from x gradient. Changing nu slightly increase the height in general, but increase the height less on large y gradient region, like eyebrow. In the paper it should adjust the z value with nu proportion from y gradient. Changing lambda adjust the value of z according to current z, if lambda is too large, it would cause gradient explosion.

Why it is called bas-relief is named because the surface could either be inward or outward oriented, and it is a famous technique in old paintings or sculptures.

Q2.g

With mu = 0, nu = 0, lambda = ϵ, where ϵ is a infinite small value > 0. The transform would cause the gradient to vanish so the surface would be as flat as possible.

Q2.h

Acquiring more pictures with different lighting directions would help resolve ambiguity.