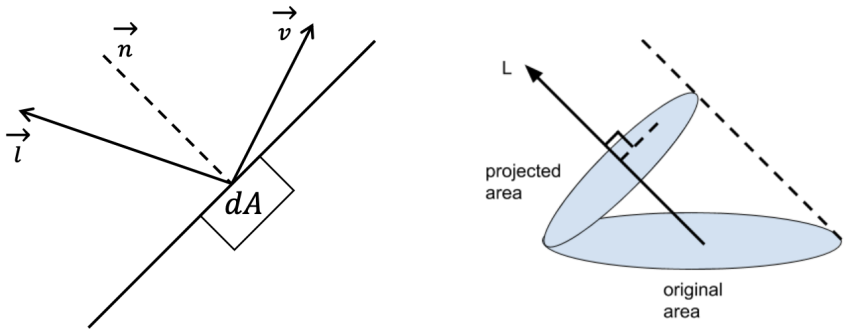
16-820 Advanced Computer Vision
Homework 5: Photometric Stereo

Abhinandan Vellanki

1 Calibrated Photometric Stereo

1.a Understanding $n - dot - l$ lighting



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

Figure 1: Geometry of photometric stereo

In the figure 1(a), the geometry is as follows:

- dA : Infinitesimally small area of surface being studied
- $\vec{l}$: Direction of illumination source
- $\vec{n}$: Direction of normal to the surface dA
- $\vec{v}$: Viewing Direction

From the Lambertian Reflection Model:

$$BRDF = \frac{Radiance}{Irradiance}$$

For a diffuse surface, the BRDF is $\frac{\rho_d}{\pi}$, where ρ_d is the albedo.

$$\implies BRDF = \frac{\rho_d}{\pi} = \frac{L}{E}$$

Irridiance (E) is defined as $E = \frac{Icos\theta_i}{r^2}$. where I is the Intensity of the incoming light, θ_i is the angle of incidence and r is the distance from light source to the surface, which is not specified. Hence the equation becomes:

$$L = \frac{\rho_d}{\pi} I cos\theta_i$$

cos θ_i can be written as $\vec{n} \cdot \vec{l}$, i.e., dot product of surface normal and direction of incoming light.

In Figure 1(b), the projected area or foreshortened area is introduced to scale the light incident on the actual area being studied. When the light is incident at an angle θ_i, the irridiance is scaled by cos θ_i to accurately account for the loss of intensity with varying θ_i

By definition of a Lambertian reflection model, the surface is assumed diffuse and that for a given light source and angle of incidence, the surface appears equally bright from all viewing angles.

1.b Rendering $n - dot - l$ lighting

Code:

```
    def renderNDotLSphere(center, rad, light, pxSize, res):
        [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)
        image = None

        # Normalize surface normals
        N = np.stack([X - center[0], Y - center[1], Z - center[2]], axis
    =-1)
        # Subtracting center coordinates for relative position
        N_norm = np.linalg.norm(N, axis=-1, keepdims=True)
        N = np.divide(N, N_norm, where=N_norm != 0)

        # Normalize light direction
        L = light / np.linalg.norm(light)

        # N-dot-L shading using dot product
        image = np.dot(N, L)
        image = np.clip(image, 0, None)

        return image
```

Screenshots:

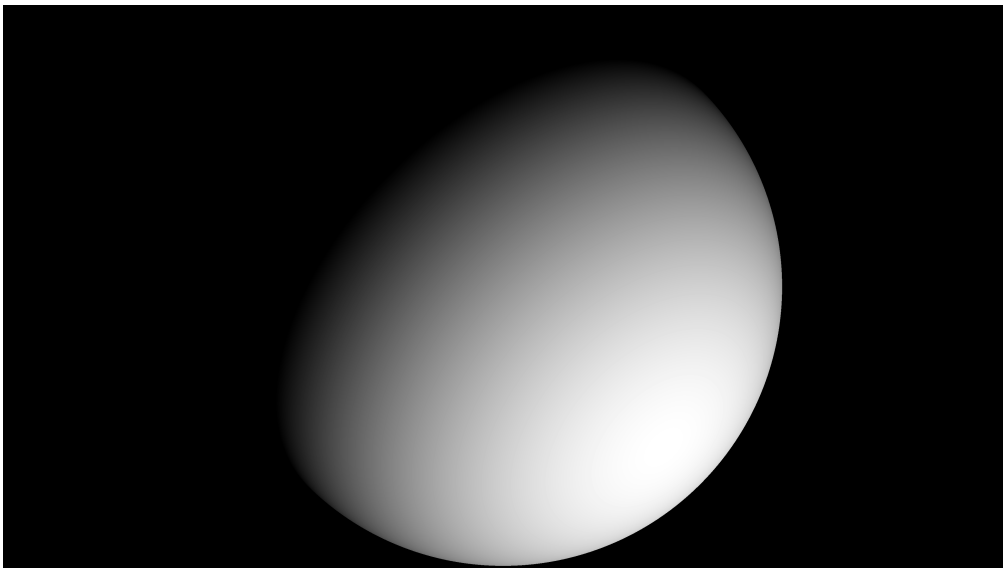


Figure 2: Lighting Direction: $(1, 1, 1)/\sqrt{3}$

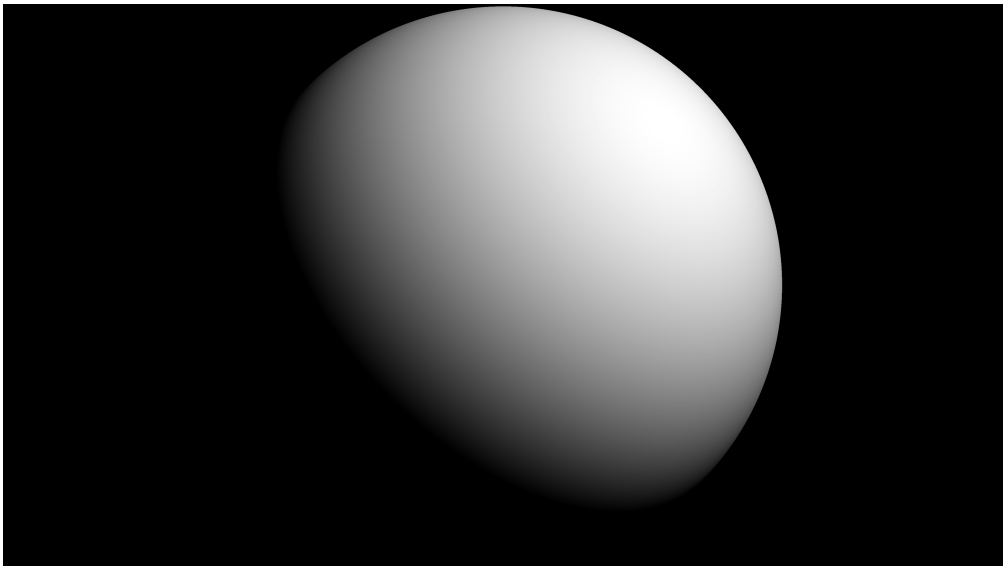


Figure 3: Lighting Direction: $(1, -1, 1)/\sqrt{3}$

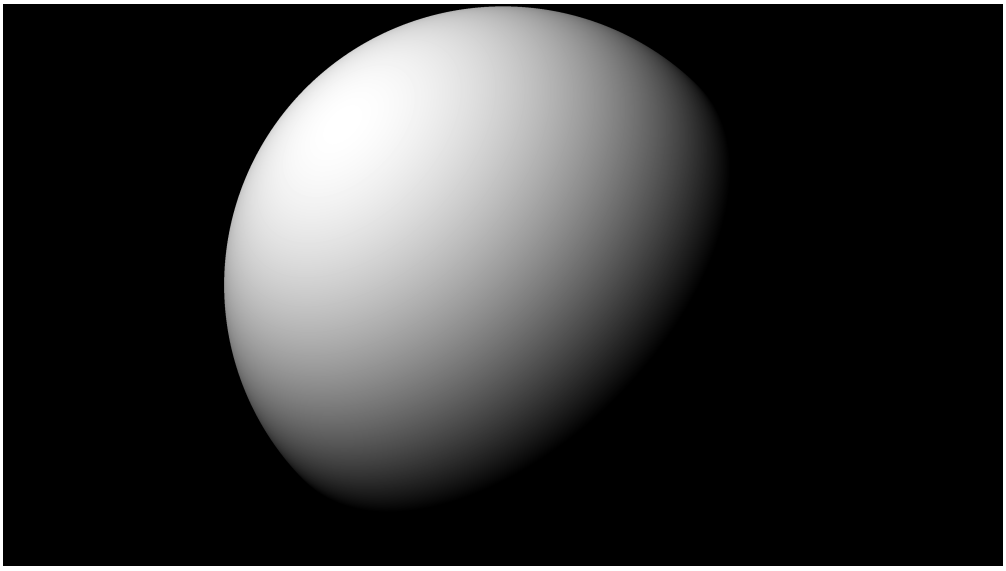


Figure 4: Lighting Direction: $(-1, -1, 1)/\sqrt{3}$

1.c Loading Data

Code:

```
    def loadData(path="../data/"):
        I = None
        L = None
        s = None
        # Your code here

        # Load images
        I = []
        for i in range(1, 8):
            im = skimage.io.imread(fname=path + "input_" + str(i) + ".tif"
    ).astype(np.uint16)
            s = im.shape[:2] # Image shape
            im = skimage.color.rgb2xyz(im)[:, :, 1].flatten()
            I.append(im)

        I = np.array(I)
        # print(I.shape)

        # Load light directions
        L = np.load(path + "sources.npy").T
        # print(L.shape)

        # print(s)

        return I, L, s
```

1.d Initials

Since the unknowns for each point i on the surface are the pseudo-normals which are represented as (X_i, Y_i, Z_i), the rank of the matrix I should be 3. That is, given at-least 3 unique, linearly independent lighting directions, the pseudo-normals can be uniquely estimated for each point X_i, Y_i, Z_i.

Using SVD, we get:

```
    if __name__ == "__main__":
        # Part 1(d)
        U, Sigma, VT = np.linalg.svd(I, full_matrices=False)
        print("Singular Values: ", Sigma)
```

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

Which is more than the required number of 3. Hence, we have more measurements than unknowns. This is probably because the measurements come with noise which makes them unique. Also, since there are seven unique lighting directions, the measurements would be unique.

1.e Estimating Pseudo-Normals

The problem was formulated as a least squares problem and solved using Numpy's least squares solver or by computing the pseudoinverse.

```
    def estimatePseudonormalsCalibrated(I, L):
        # Estimate pseudonormals
        # B = np.linalg.lstsq(L.T, I)[0]

        # Using pseudo-inverse
        B = np.linalg.inv(L @ L.T) @ L @ I

        return B
```

1.f Albedos and Normals

```
    def estimateAlbedosNormals(B)
        # estimate albedos and normals
        albedos = np.linalg.norm(B, axis=0)
        normals = B / albedos
        return albedos, normals

    def displayAlbedosNormals(albedos, normals, s):
        albedoIm = albedos.reshape(s)
        normalIm = normals.T.reshape(s[0], s[1], 3)
        scale = np.max(normalIm) - np.min(normalIm)
        normalIm = (normalIm - np.min(normalIm)) / scale
        return albedoIm, normalIm
```

Results:

Figure 5: Albedos

The albedo values are in the range [0, 1], where 0 is a perfectly black surface and 1 is a perfectly white surface. Hence, the albedos of the facial hair appear black, that of the lips and eyelids are close to black and the rest of the face is some shade of grey, which are all as expected. However, there are certain points such as the contour between the nose and face and the contours in the ears which appear sharp white, i.e., closer to 1. This is not what is expected. This albedo distribution describes these parts as highly reflective, which is incorrect, making this model slightly inaccurate.

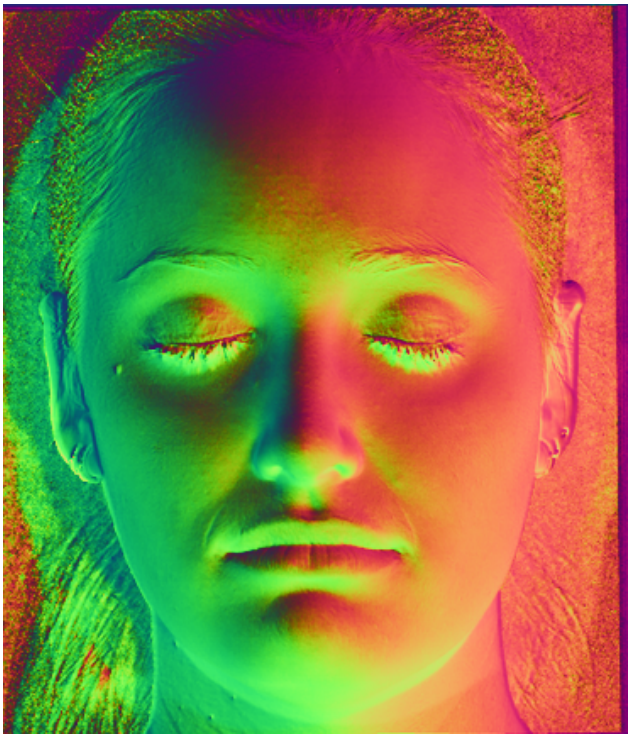


Figure 6: Surface Normals

This image matches the expected curvature of the face.

1.g Normals and Depth

Given a depth map $z = f(x, y)$ and representation of surface normal at any point as $n = (n_1, n_2, n_3)$.

By definition of a surface normal, it is perpendicular to the tangent at that point. Here, the tangent vectors describe how the surface changes.

Hence, tangent vector in x direction: $t_x = (1, 0, \frac{\partial z}{\partial x})$ and tangent vector in y direction: $t_y = (0, 1, \frac{\partial z}{\partial y})$

As defined above, the normal vector is obtained by: $n = t_x \times t_y \implies n = (-f_x, -f_y, 1)$.

Computing unit vector:

$$\hat{n} = \frac{n}{||n||} = \frac{(-f_x, -f_y, 1)}{||-f_x, -f_y, 1||}$$

Hence, from above definitions:

$$\begin{aligned} n_1 &= -f_x \\ n_2 &= -f_y \\ n_3 &= 1 \\ \implies f_x &= -\frac{n_1}{n_3} \\ \implies f_y &= -\frac{n_2}{n_3} \end{aligned}$$

1.h Understanding Integrability of Gradients

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

Computing Gradients:

$$g_x = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \; g_y = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \end{bmatrix}$$

To reconstruct, taking start point $g(0, 0) = 1$

1. Row 1 = $\begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$

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

Putting the rows together gives the original g

2. Column 1 = $\begin{bmatrix} 1 \\ 5 \\ 9 \\ 13 \end{bmatrix}$

Column 2, 3, 4 = $\begin{bmatrix} 2 & 3 & 4 \\ 6 & 7 & 8 \\ 10 & 11 & 12 \\ 14 & 15 & 16 \end{bmatrix}$

Putting the columns together gives the original g.

Hence, the final result obtained is the same with both methods and is also equal to the initial matrix from which the gradients were computed. Hence, the gradients are integrable in multiple ways to recover the original shape.

We can modify the gradients by adding some noise to them, i.e., slightly changing one or more of the values. This would result in the results of the two integration methods being different i.e., non-integrable.

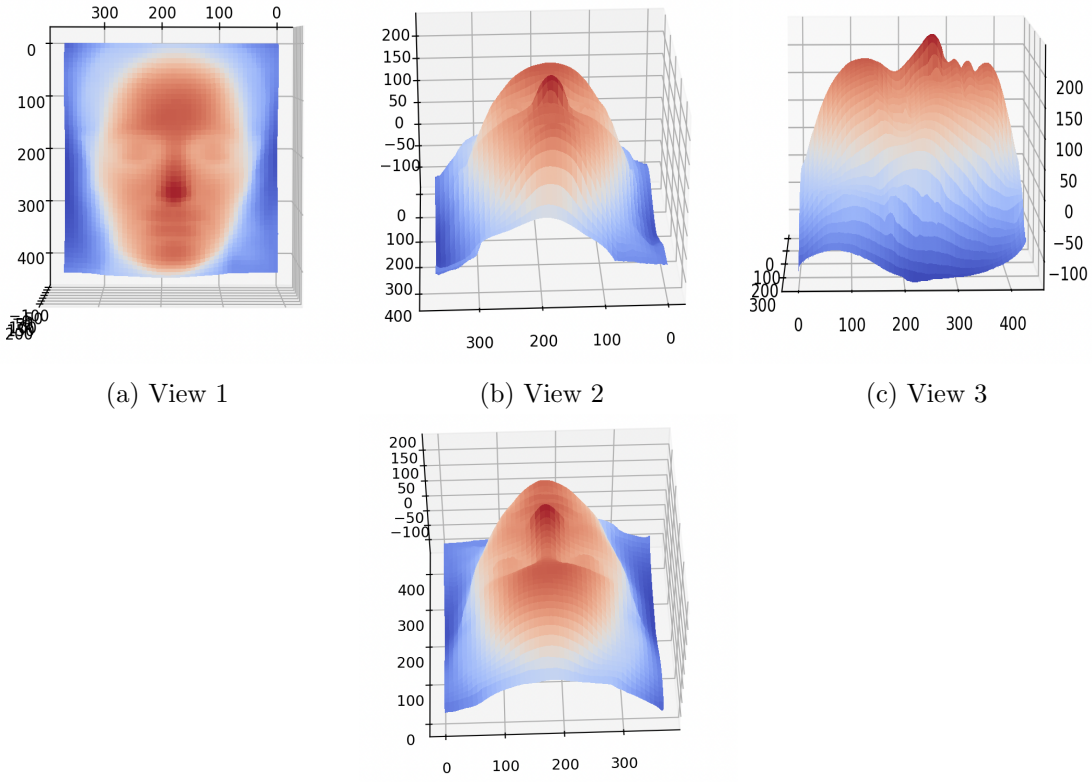
In a similar manner, the gradients estimated in (g) might be noisy and hence, non-integrable.

1.i Shape Estimation

Code:

```
    def estimateShape(normals, s):
        fx = -normals[0, :] / normals[2, :]
        fy = -normals[1, :] / normals[2, :]
        fx = fx.reshape(s)
        fy = fy.reshape(s)
        surface = integrateFrankot(fx, fy)

        return surface
```



(a) View 1 (b) View 2 (c) View 3

(d) View 4

Figure 7: Recovered Shape From Calibrated Photometric Stereo

2 Uncalibrated Photometric Stereo

2.a Uncalibrated Normal Estimation

Decomposing I using Singular Value Decomposition: $I = U\Sigma V$
Since we want to use only the 3 largest singular values, we set all values except for the first 3 in Σ to 0.

The new $\hat{\Sigma} = \begin{bmatrix} s_1 & 0 & 0 & ... \\ 0 & s_2 & 0 & ... \\ 0 & 0 & s_3 & ... \\ ... & ... & ... & 0 \end{bmatrix}$

Hence the estimated I, i.e., $\hat{I} = U\hat{\Sigma}V^T$

$$\implies \hat{I} = U \begin{bmatrix} \sqrt{s_1} & 0 & 0 & ... \\ 0 & \sqrt{s_2} & 0 & ... \\ 0 & 0 & \sqrt{s_3} & ... \\ ... & ... & ... & ... \end{bmatrix} \begin{bmatrix} \sqrt{s_1} & 0 & 0 & ... \\ 0 & \sqrt{s_2} & 0 & ... \\ 0 & 0 & \sqrt{s_3} & ... \\ ... & ... & ... & ... \end{bmatrix} V^T$$

Taking only first 3 columns of U and first 3 rows of V:

$$\implies \hat{L^T} = U[:, :3] \begin{bmatrix} \sqrt{s_1} & 0 & 0 \\ 0 & \sqrt{s_2} & 0 \\ 0 & 0 & \sqrt{s_3} \end{bmatrix} \implies \hat{B} = \begin{bmatrix} \sqrt{s_1} & 0 & 0 \\ 0 & \sqrt{s_2} & 0 \\ 0 & 0 & \sqrt{s_3} \end{bmatrix} V[:3, :]$$

2.b Calculation and Visualization

Code:

```
    def estimatePseudonormalsUncalibrated(I):
        U, S, V = np.linalg.svd(I, full_w   matrices=False)
        B = np.diag(np.sqrt(S[:3])) @ V[:3, :]
        L = U[:, :3]@ np.diag(np.sqrt(S[:3])).T

        return B, L

    if __name__ == "__main__":
        # Part 2 (b)
        I, L_0, s = loadData("../data/")
        B, L = estimatePseudonormalsUncalibrated(I)

        albedos, normals = estimateAlbedosNormals(B)
        # print("Albedos shape: ", albedos.shape)
        # print("Normals shape: ", normals.shape)
        albedos_image, normals_image = displayAlbedosNormals(albedos,
   normals, s)
        plt.imsave("2b-albedo.png", albedos_image, cmap="gray")
        plt.imsave("2b-normals.png", normals_image, cmap="rainbow")
```

(a) Albedos

(b) Normals

2.c Comparing to Ground Truth Lighting

```
Original Lighting Directions:
[[-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]]
```

```
Estimated Lighting Directions:
[[-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 ]]
```

These values are quite different.
A possible fix would be to avoid factorizing the Σ matrix into the square roots of the values and directly use the matrix obtained from singular value decomposition:
$\hat{B} = \Sigma[:3, :3]V^T[:3, :]$ and $\hat{L} = U[:, :3]$.

2.d Reconstructing the shape, attempt 1

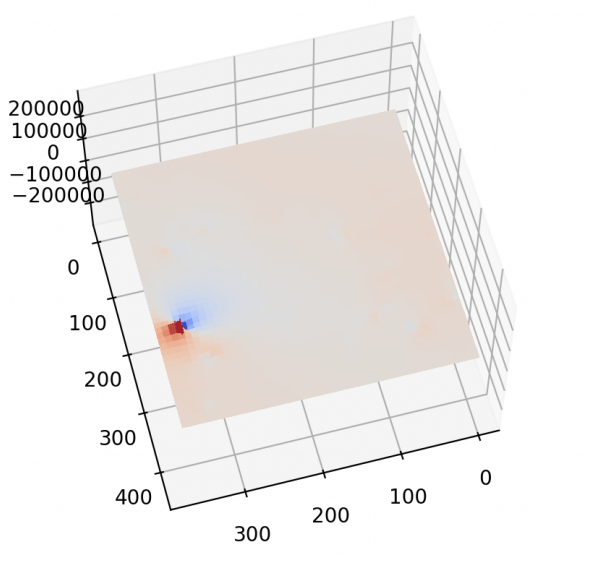
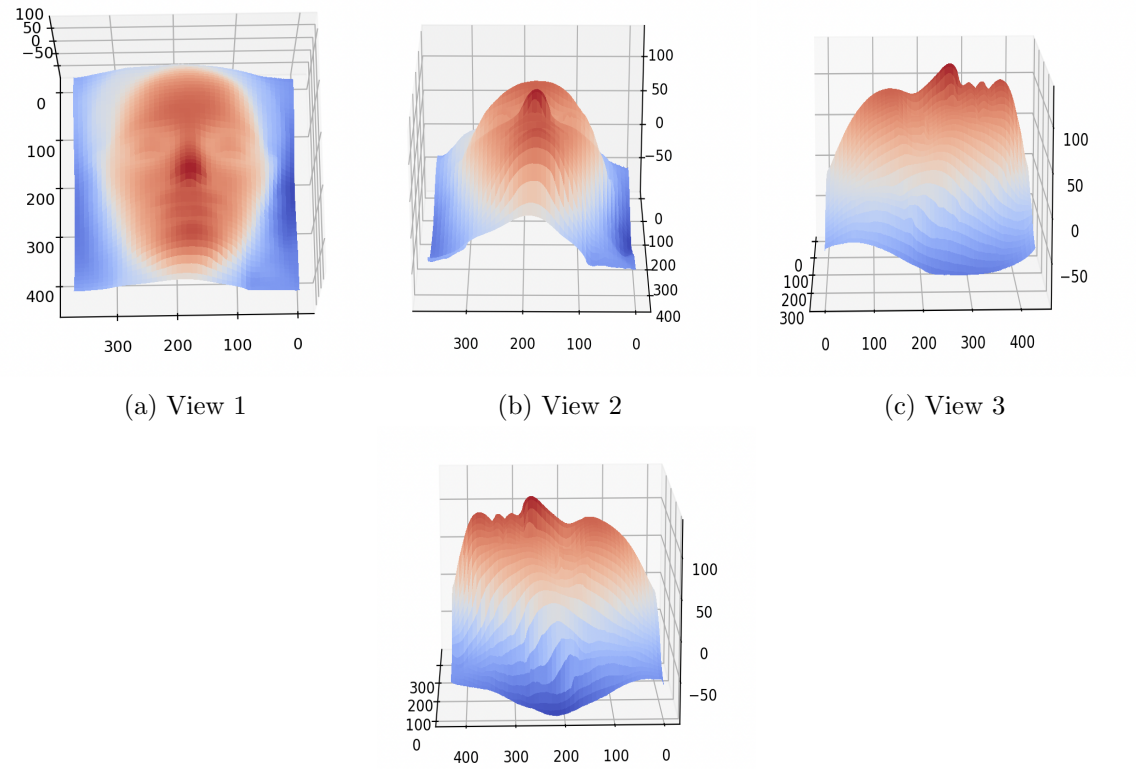


Figure 9: Attempt 1

This does not look like a face at all.

2.e Reconstructing the shape, attempt 2



(a) View 1

(b) View 2

(c) View 3

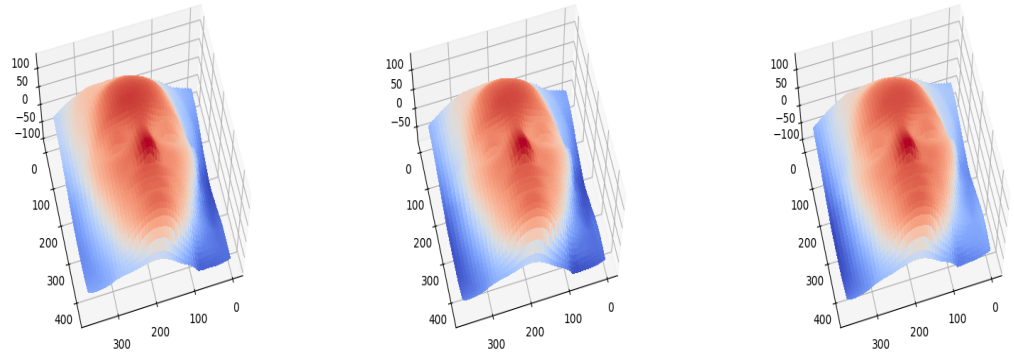
(d) View 4

Figure 10: Recovered Shape From Uncalibrated Photometric Stereo

While the shape is close to the shape obtained in calibrated photometric stereo, the z values differ and have lower variance and maximums than the result of the calibrated procedure.

2.f Why low relief?

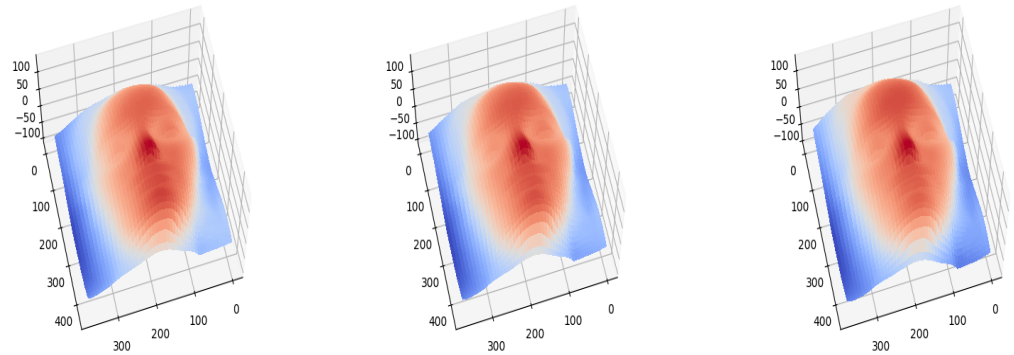
Varying μ:



(a) μ=0.1, ν=1, λ=1 (b) μ=0.5, ν=1, λ=1 (c) μ=1, ν=1, λ=1

Figure 11: Varying μ values

μ directly affects the left side of the shape, either flattening it or making it more pronounced. Decreasing it makes the shape flatter (less variance in and maximum of z) and increasing it makes the shape more pronounced (more variance in and maximum of z). It was also observed that the sign of the z values changed according to its own size.
Varying ν:

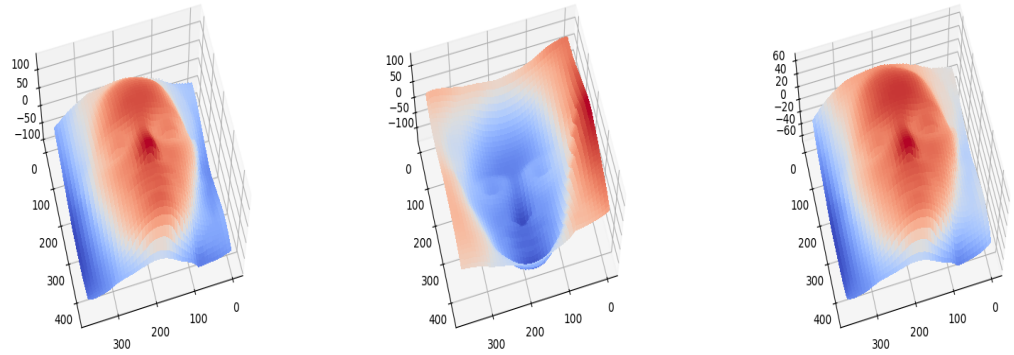


(a) μ=1, ν=0.1, λ=1 (b) μ=1, ν=0.5, λ=1 (c) μ=1, ν=1, λ=1

Figure 12: Varying ν values

ν directly affects the right side of the shape, either flattening it or making it more pronounced. Decreasing it makes the shape flatter (less variance in and maximum of z) and increasing it makes the shape more pronounced (more variance in and maximum of z). It was also observed that the sign of the z values changed according to its own size.

Varying λ:



(a) μ=1, ν=1, λ=1 (b) μ=1, ν=1, λ=-1 (c) μ=1, ν=1, λ=0.5

Figure 13: Varying λ values

λ affects the entire shape, increasing or decreasing the variance of z and changing the sign of the values depending on its own size.

These transformations can flatten or exaggerate the object's depth while preserving its visual appearance, much like how bas-relief sculptures create the illusion of depth on a relatively flat surface.

The name "bas-relief ambiguity" thus captures the essence of this phenomenon - the inability to distinguish between different 3D structures that appear visually similar when viewed from a particular angle, analogous to how bas-relief sculptures create the illusion of 3D forms on a mostly flat surface.

2.g Flattest Surface

This can be obtained by setting large, equal and opposite μ and ν values and making λ as low as possible:
With μ = -10; ν = 10; λ = 0.1:

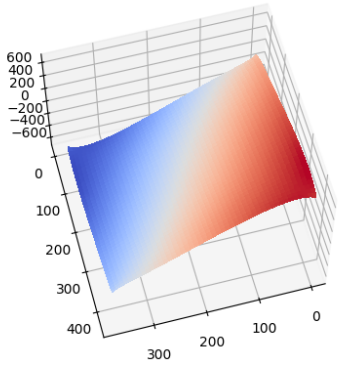


Figure 14: μ=-10, ν=10, λ=0.1

2.h More Measurements

Adding more measurements won't necessarily improve the results in this case. As discussed earlier, 3 linearly independent light sources are sufficient to compute pseudo-normals. Hence, more measurements would not necessarily reduce the ambiguity that comes with the noise in each new measurement.