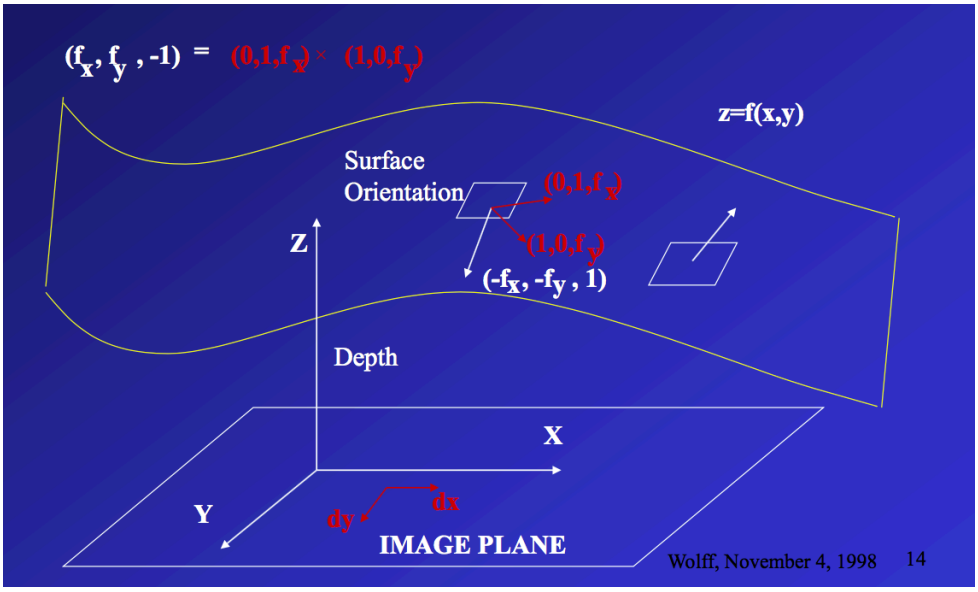
Assignment 3: Photometric Stereo
Practical Report

Chuan Long

The goal of this assignment is to implement an algorithm that reconstructs a surface using the concept of photometric stereo. You can assume a Lambertian reflectance function, but the albedo is unknown and non-constant in the images. Your program will read multiple images as input along with the light source direction for each image. All data sets for this assignment are provided are provided on canvas and the class website.

1. Theoretical Basis

1.1 Reflectance map



Reflectance map is a viewer-centered representation of reflectance.

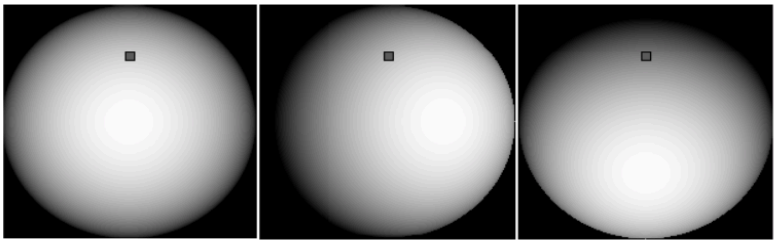
$$(-fx, -fy, 1) = (-p, -q, 1)$$

p, q comprises a gradient or gradient space representation for local surface orientation. Reflectance map expresses the reflectance of a material directly in terms of viewer-centered representation of local surface orientation.

1.2 Solve the Linear equation system

As what we have learned from the class, in order to solve the contour map arbitrary, we have to ***use at least three images*** to determine the value of a normal vector at one pixel. So, the following figure is the representation and the equation that we have to solve. Please note that: $\boldsymbol{Ii}$ represents the known light intensity of the pixel, $\boldsymbol{Si}$ represents the known scaled light source direction. $\boldsymbol{p}$ represents the unknown albedo, which is a lamebrain surface but not constant. And $\boldsymbol{N}$ represent the unknown normal that we have to solve.

Photometric Stereo



$I_1 = \rho \mathbf{S}_1.\mathbf{N}$ $I_2 = \rho \mathbf{S}_2.\mathbf{N}$ $I_3 = \rho \mathbf{S}_3.\mathbf{N}$

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} \mathbf{S}_1^T \\ \mathbf{S}_2^T \\ \mathbf{S}_3^T \end{bmatrix} \rho\mathbf{N}$$

$\rho\mathbf{N} = \mathbf{S}^{-1}\mathbf{I}$ Solve linear equation system to calculate $\bar{N}$.

As suggested, we used the solution from Forsyth & Ponce. Which is as following:

Solution Forsyth & Ponce:

For each point source, we know the source vector (by assumption). We assume we know the scaling constant of the linear camera (k). Fold the normal (**N**) and the reflectance (**ρ**(x,y)) into one vector **g**, and the scaling constant and source vector into another $\mathbf{V_j}$.

- Out of shadow:

$$\begin{aligned} I(x,y) &= kB(\boldsymbol{x}) \\ &= kB(x,y) \\ &= k\rho(x,y)\boldsymbol{N}(x,y) \cdot \boldsymbol{S}_1 \\ &= \boldsymbol{g}(x,y) \cdot \boldsymbol{V}_1 \end{aligned}$$

- In shadow:

$$I(x,y) = 0$$

where $\boldsymbol{g}(x,y) = \rho(x,y)\boldsymbol{N}(x,y)$ and $\boldsymbol{V}_1 = k\boldsymbol{S}_1$, where k is the constant connecting the camera response to the input radiance.

In this case, the intensity in shadow might be 0, and we can not get any derivative terms from the 0. So, in order to balance this equation, we used a matrix trick to stack these terms into a big matrix form, and solve the problem using lease square method.

Matrix Trick for Complete Shadows

- Matrix from Image Vector:

$$\mathcal{I}(x,y) = \begin{pmatrix} I_1(x,y) & \dots & 0 & 0 \\ 0 & I_2(x,y) & \dots & 0 \\ \dots & & & \\ 0 & 0 & \dots & I_n(x,y) \end{pmatrix}$$

- Multiply LHS and RHS with diag matrix

$$\mathcal{I}\boldsymbol{i} = \mathcal{I}\mathcal{V}\boldsymbol{g}(x,y)$$

$$\begin{pmatrix} I_1^2(x,y) \\ I_2^2(x,y) \\ .. \\ I_n^2(x,y) \end{pmatrix} = \begin{pmatrix} I_1(x,y) & 0 & .. & 0 \\ 0 & I_2(x,y) & .. & .. \\ .. & .. & .. & 0 \\ 0 & .. & 0 & I_n(x,y) \end{pmatrix} \begin{pmatrix} \mathbf{V}_1^T \\ \mathbf{V}_2^T \\ .. \\ \mathbf{V}_n^T \end{pmatrix} \mathbf{g}(x,y)$$

Known Known Known Unknown

2. Detailed Procedure

2.1 Pre-processing the data

For the ***images*** that are color image, I have to transform them (dogs image) to gray scaled image, otherwise just read the image as intensity matrix. Then, scale these intensity matrixes to [0, 1] range divided by 255. Please note that the original image data are in the form of unit8, I have to transform it to double format to avoid resulting zero when divide by 255.
For the ***light source vectors***, normalize them individually, that is divide each element by its own norm.

2.2 Create the matrix

I created one matrix for albedo map, two matrixes for two derivatives, three matrixes for the three components (x, y, z) of the normal vector, one matrix for the depth(height) map. All of them are the same size the corresponding images.

2.3 Solve the linear equation

In a two for-loops for each pixel, I stacked the corresponding values into the big matrix, and solve the equation by the suggested method $\boldsymbol{g} = \boldsymbol{TV} \setminus \boldsymbol{T_i}$. In this case, I used all the four images for each object. So, MATLAB solve the over-constrained linear equation for me automatically (this is the mechanism from MATLAB, when the matrix is not square, it will decompose it using SVD, that is Least Square Method). And then, I get the albedo value, three components of normal vector, two derivative terms, and stacked them into the corresponding matrix in current pixel coordinate.

2.4 Reconstruct depth map using integration

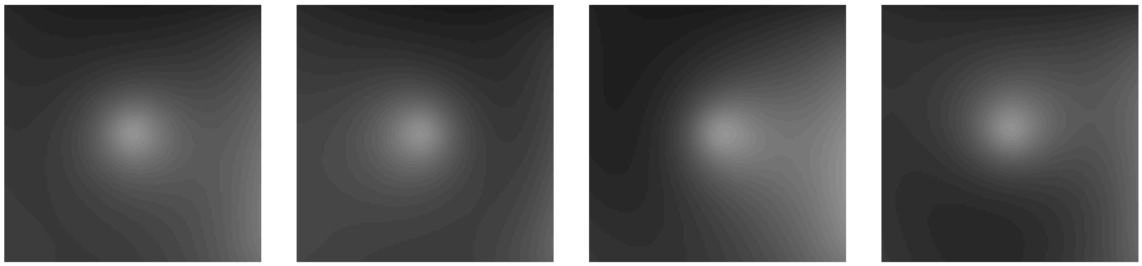
As suggested, I integrate the derivative terms to reconstruct the depth map. I integrate the derivatives not only from the top-left of the image but also the middle of the image as required for the synthetic images and sphere images. The dog images have bad results when integrate from the middle of the image. What is more, I also adopted the bonus method to reconstruct the depth map. I will give some more details about the discussion of this part later in this report.

2.5 Plot the results

I have plotted the results of all the forms that suggested by the assignment.
For the **albedo map**, I used **imshow()** to show the estimated albedo map, considering it as an intensity map.
For the **surface normal map**, I plotted the three components of the normal vector individually, and the needle map (spacing 1). Beyond that, I also reconstruct the new shaded image by different light source vector, and I will specify them in the following results.
For **the depth(height) map**. I plotted them in both the **surf()** and the **mesh()** method.

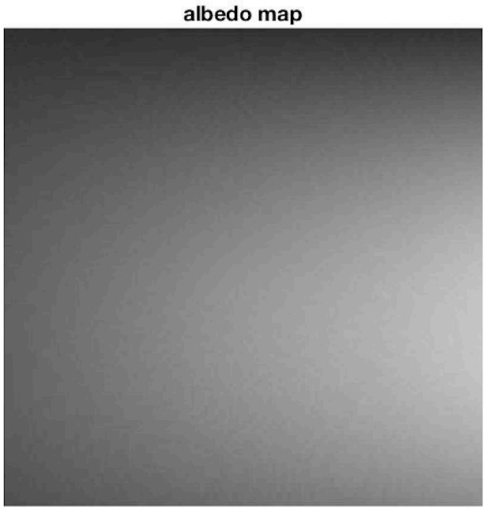
3. Results

3.1 Synthetic Images



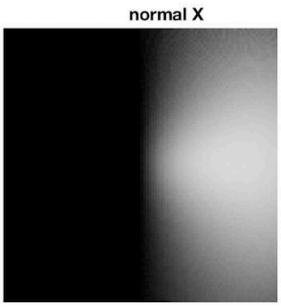
im1.png $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ im2.png $\begin{bmatrix} -0.2 \\ 0 \\ 1 \end{bmatrix}$ im3.png $\begin{bmatrix} 0.2 \\ 0 \\ 1 \end{bmatrix}$ im4.png $\begin{bmatrix} 0 \\ -0.2 \\ 1 \end{bmatrix}$

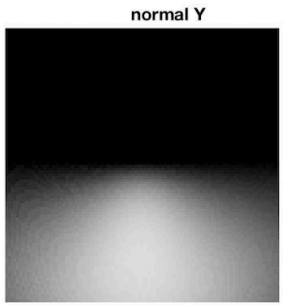
3.1.1 Albedo Map

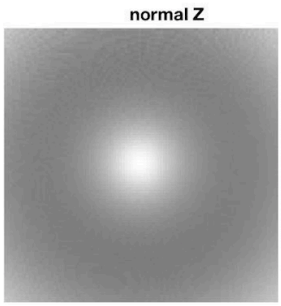


3.1.2 Estimated surface normal

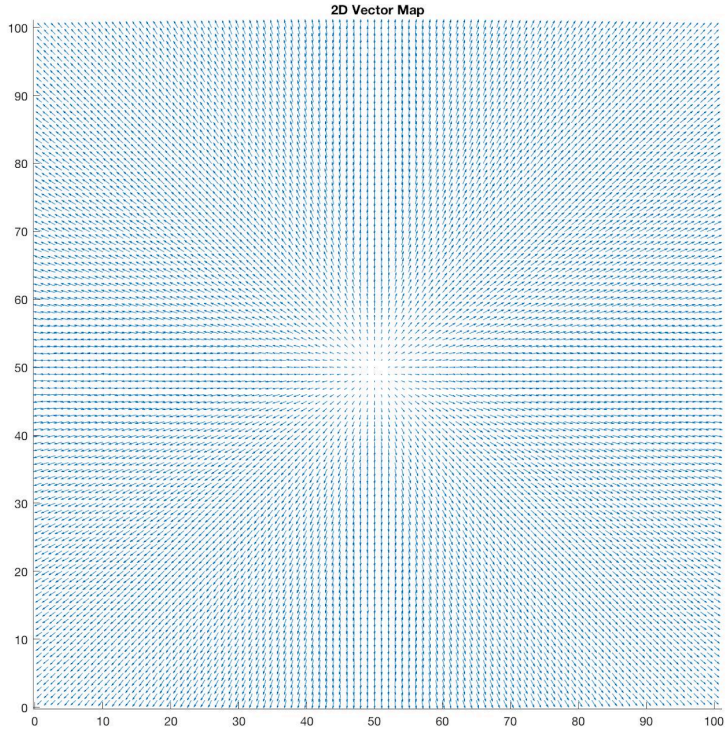
3.1.2.1 Three Components of surface normal



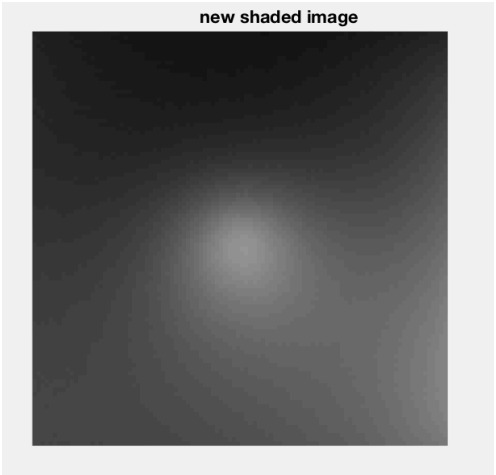




3.1.2.2 Needle map (2D vector map)



3.1.2.3 New shaded image (Light source : [0, 0.2, 1])



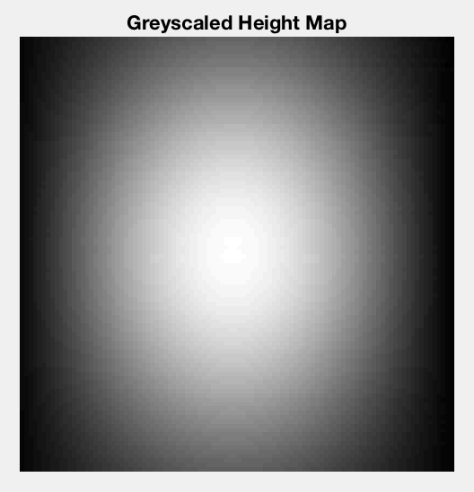
3.1.3 Reconstructed Height Surface

3.1.3.1 Gray-level depth image

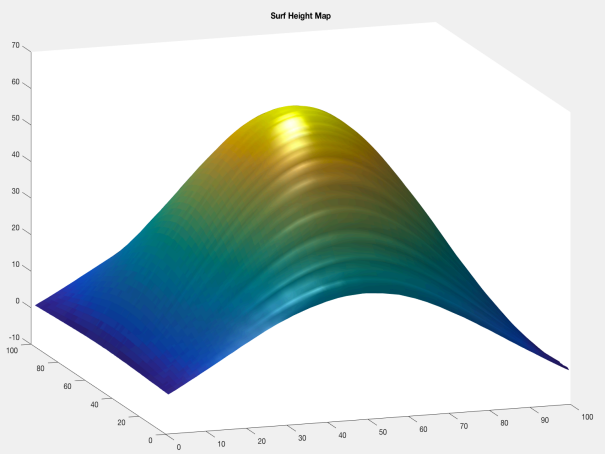
First, I used each depth value to represent the intensity of gray scale. Then, I have to scale the height value of each pixel into the intensity. So, I standardize the height map by using following method:

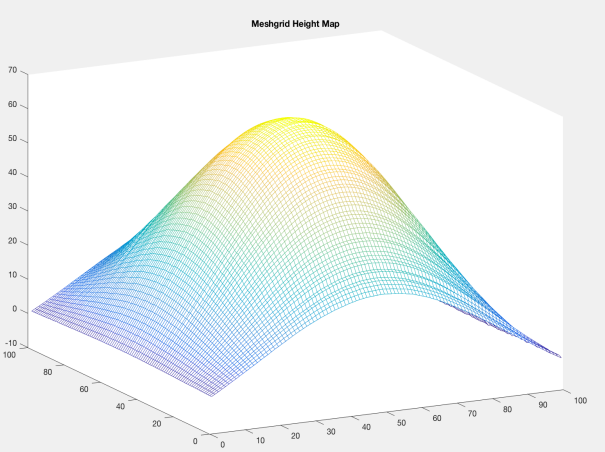
```
height_map_greyscaled = (height_map(:) - min(height_map(:)))/      (max(height_map(:)) - min(height_map(:)));
height_map_greyscaled = reshape(height_map_greyscaled, w, h);
```

Then I have the following image:

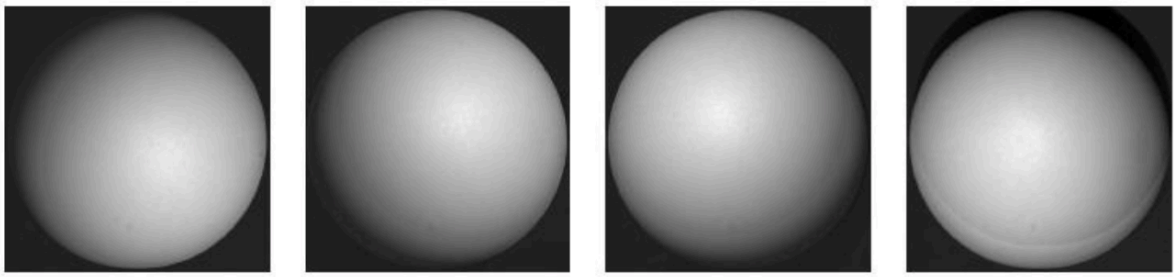


3.1.3.2 Depth map in surf() and mesh()





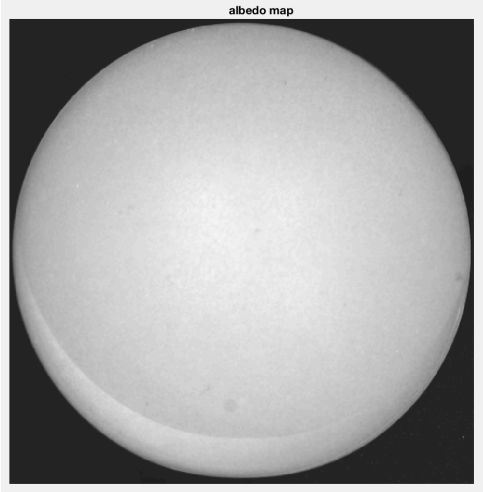
3.2 Sphere Images



real1.bmp real2.bmp real3.bmp real4.bmp

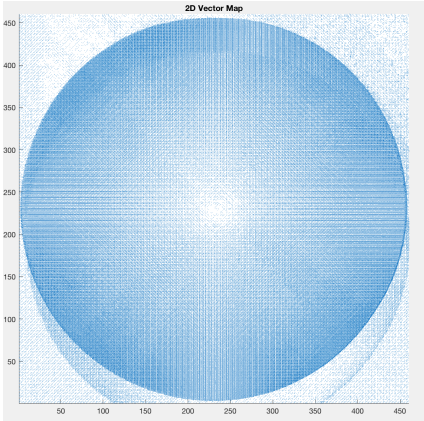
$$\begin{bmatrix} 0.38359 \\ 0.236647 \\ 0.89266 \end{bmatrix} \quad \begin{bmatrix} 0.372825 \\ -0.303914 \\ 0.87672 \end{bmatrix} \quad \begin{bmatrix} -0.250814 \\ -0.34752 \\ 0.903505 \end{bmatrix} \quad \begin{bmatrix} -0.203844 \\ 0.096308 \\ 0.974255 \end{bmatrix}$$

3.2.1 Estimated Albedo Map

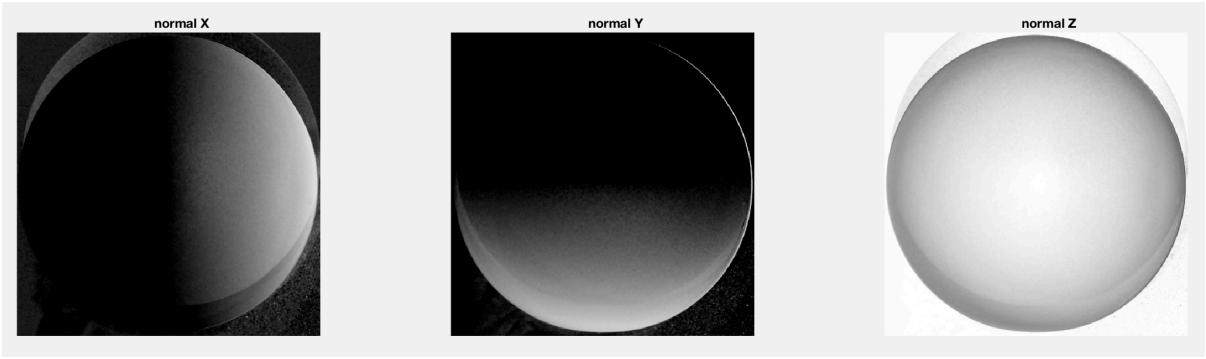


3.2.2 Estimated Surface Normal

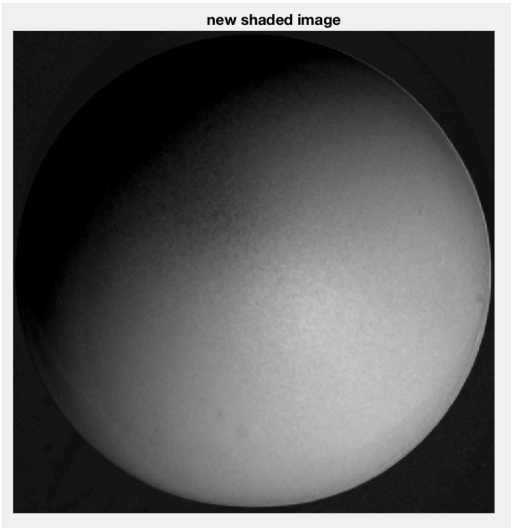
3.2.2.1 Needle Map



3.2.2.2 Three components of surface normal vector (normalized normal)

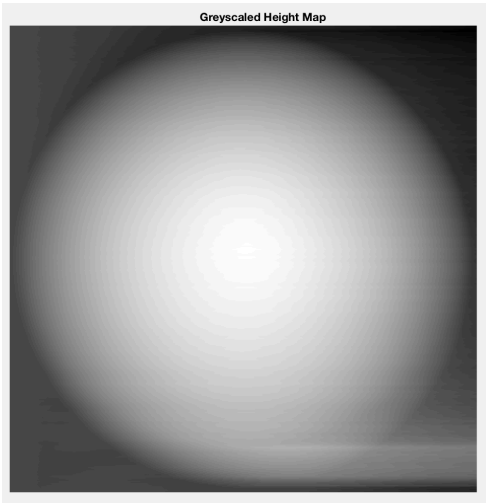


3.2.2.3 New shaded image (Light source:[1, 1, 1])

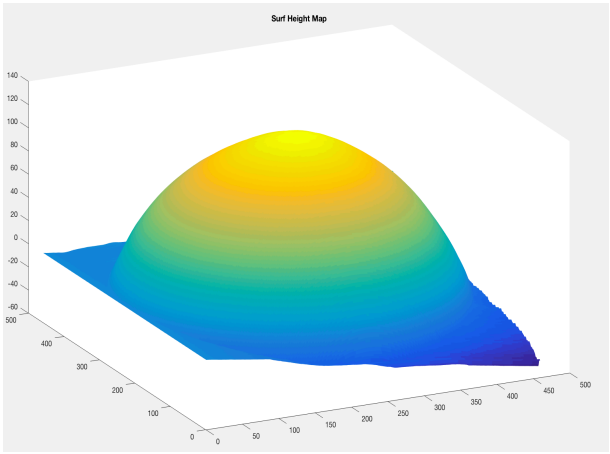


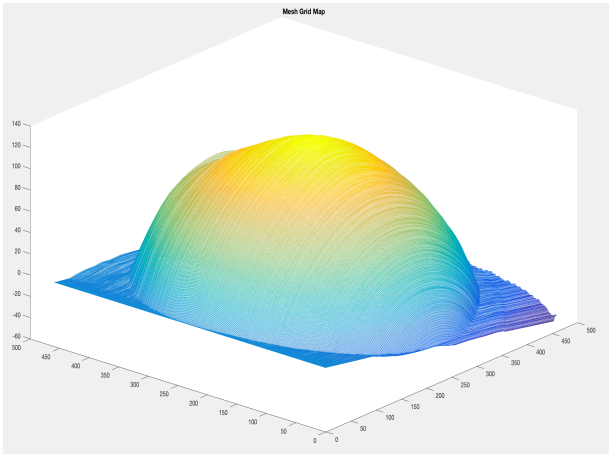
3.2.3 Reconstructed Height Surface

3.2.3.1 Gray-scaled Depth Image



3.2.3.2 Depth map in surf() and mesh()





3.3 Dog Images

dog1.tif

$$\begin{bmatrix} 16 \\ 19 \\ 30 \end{bmatrix}$$

dog2.tif

$$\begin{bmatrix} 13 \\ 16 \\ 30 \end{bmatrix}$$

dog3.tif

$$\begin{bmatrix} -17 \\ 10.5 \\ 26.5 \end{bmatrix}$$

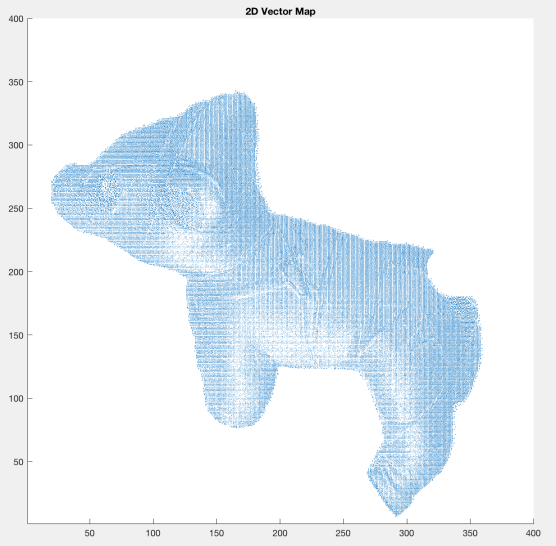
dog4.tif

$$\begin{bmatrix} 9 \\ -25 \\ 4 \end{bmatrix}$$

3.3.1 Estimated Albedo Map

3.3.2 Surface Normal

3.3.2.1 Needle Map



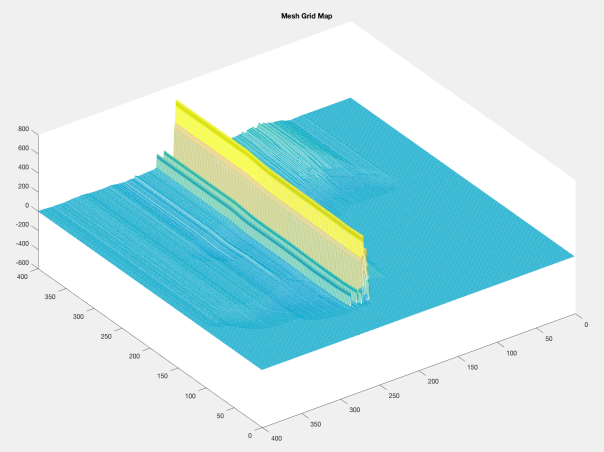
3.3.2.2 Three components of surface normal vector

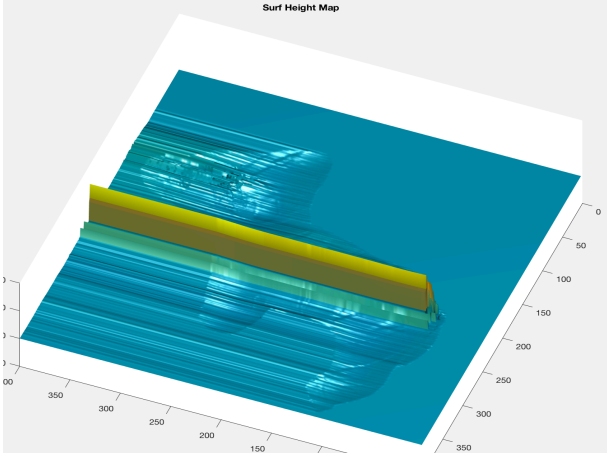
3.3.2.3 New Shaded Image (Light Source Vector[1, 1, 1])

3.3.3 Reconstructed Height Surface

3.3.3.1 Gray-scaled Depth Image

3.3.3.2 Depth Map with surf() and mesh() method





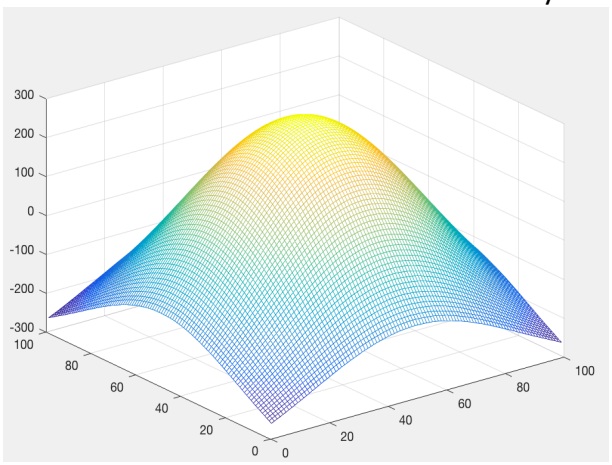
4. Bonus Part (Chellappa Method)

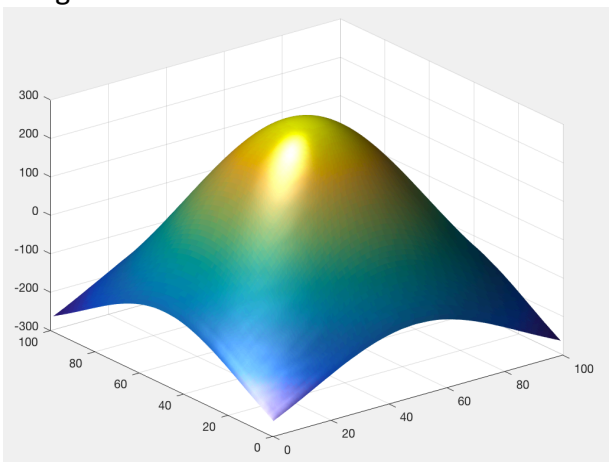
4.1 Chellappa Method

I checked out the Chellappa method and went through the MATLAB code about this method. And generate the depth map.

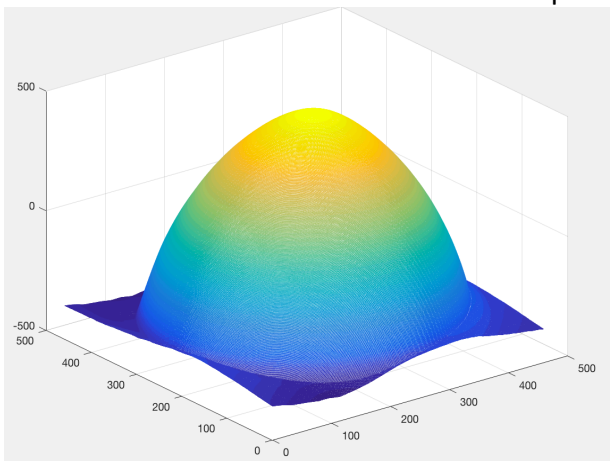
4.2 Results

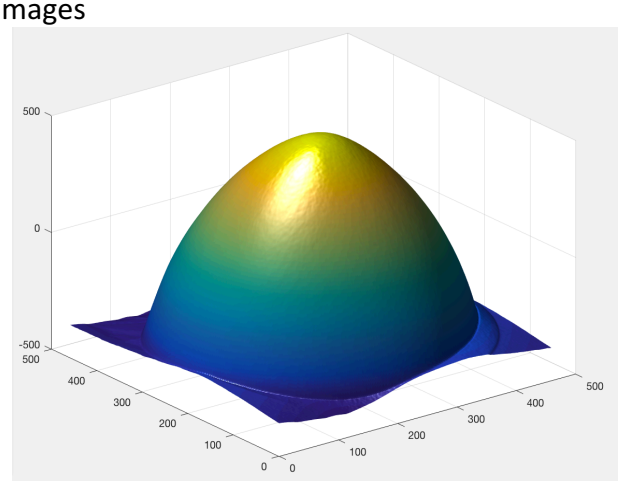
Synthetic Images



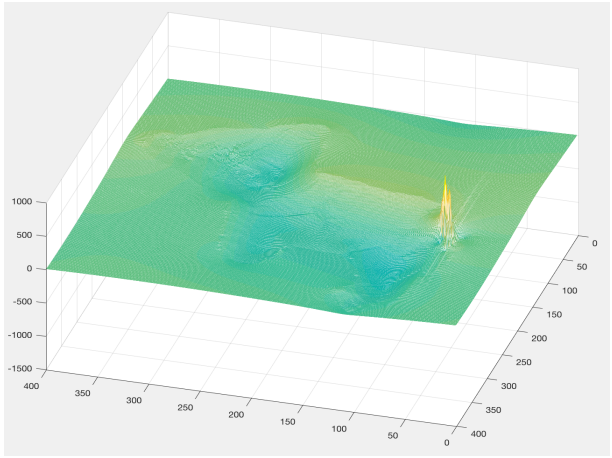


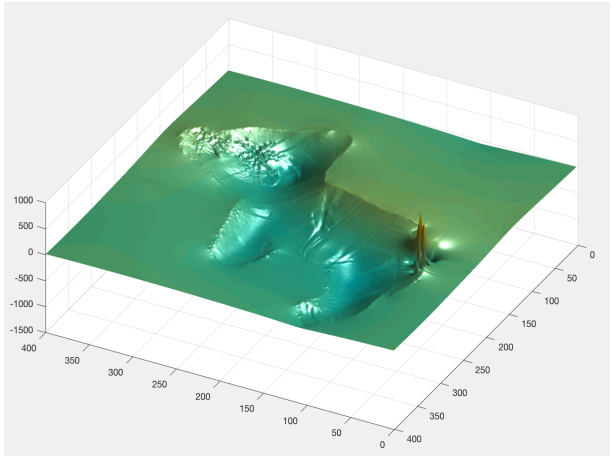
Sphere Images





Dog Images





5. Discussion

5.1 Issues and critical assessment

1. As we can see, the naïve integration method generates the depth map with noise from the left to the right of the image (see the images in the section 3.3.3.2). The reason why this would happen is that I integrate the noise from the top-left to the bottom right. For the dog images, **the dog tail raises many noise regarding the result of the Chellappa method**. And the naïve integration method just integrates these noises from the left to the right. So, I used another method that integrate the height from the middle of the image. And I got reasonable result for the sphere images and the synthetic images but the dog images are not so well (the result could be reproducible). And based on the results of the Chellappa method and the naïve method, we can verify my implementation is right.

2. Four of the intensity of the corresponding points of the images $\boldsymbol{V_i}$ could be zero at the same time. And it would raise a problem called **matrix rank deficiency**. So, in this case, it would apply the value of the normal Nan (not a value). And then I would not get any result when I want to generate the depth map. In this case, when the $\boldsymbol{V_i}$ equal to [0, 0, 0, 0], I just let the three components of the normal to be 0. An alternative method is to add ***eps*** value (which is a very small float number) to each element of the matrix so as to solve the problem of matrix rank deficiency.

5.2 Verification

1. The first method that I want to prove I get a reasonable result is that I have calculate the result of the equation:

$$\frac{\partial p}{\partial y} - \frac{\partial q}{\partial x} = 0$$

For each pixel, the values of this equation is either equal to 0 or very close to 0. I have stacked the values to a matrix called ‘checked’. This method could prove that I generate a reasonable derivative terms this time. All of the images and the values are reproducible.

2. Based on the results of Chellappa method and the naïve integration method, they are almost the same. So, to some extend, it can prove that my implementation is correct.

Note: All the images are attached. And the results and the images are reproducible. I annotate the code in a very detailed way, and you could reproduce the corresponding result as long as you took out the corresponding annotation.