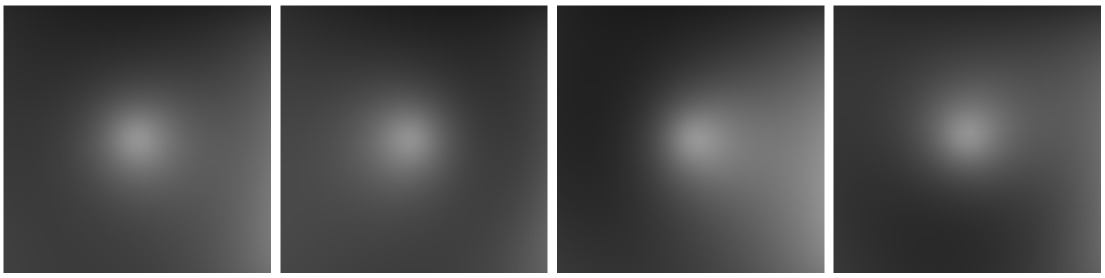
Assignment 3 - Photometric Stereo Report

Zebin Xu (zebinxu@nyu.edu)
Department of Computer Science and Engineering
NYU Tandon School of Engineering
April 11, 2017

II. Practical Problems

2.1 Photometric Stereo: Synthetic Images

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 nonconstant 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 on canvas and the class website.



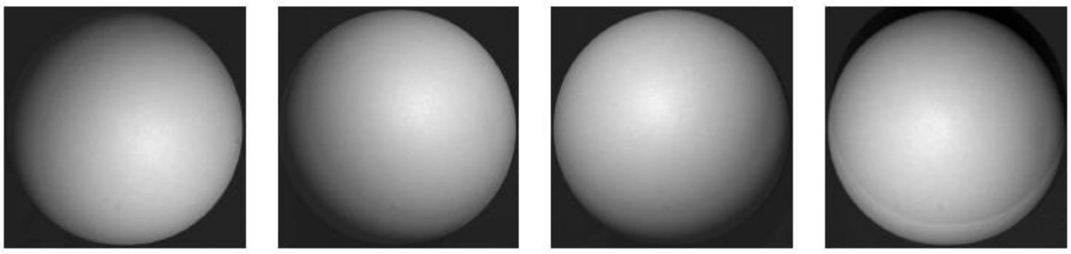
im1.png im2.png im3.png im4.png

$$\begin{bmatrix}0\\0\\1\end{bmatrix} \quad \begin{bmatrix}-0.2\\0\\1\end{bmatrix} \quad \begin{bmatrix}0.2\\0\\1\end{bmatrix} \quad \begin{bmatrix}0\\-0.2\\1\end{bmatrix}$$

Figure 1: Synthetic data: Images, file names, light source directions.

2.2 Shape from Shading from Real Images

Sphere Images:

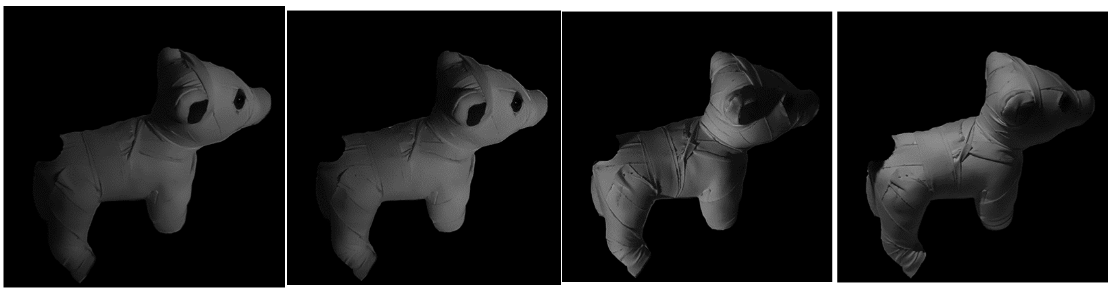


`real1.bmp` `real2.bmp` `real3.bmp` `real4.bmp`

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

Figure 2: Real data: Images, file names, light source directions.

Dog Images:



dog1.png	dog2.png	dog3.png	dog4.png
$\begin{bmatrix}16\\19\\30\end{bmatrix}$	$\begin{bmatrix}13\\16\\30\end{bmatrix}$	$\begin{bmatrix}-17\\10.5\\26.5\end{bmatrix}$	$\begin{bmatrix}9\\-25\\4\end{bmatrix}$

Figure 3: Dog image data: Images, file names, light source directions as non-normalized 3D vectors.

3. Theory: Photometric Stereo

Unlike multi-view stereo, which needs different views to reconstruct 3D information, photometric stereo allows us to reconstruct surface from multiple images taken under different illuminations, while fixing the camera and the surface in position. The camera model we choose here is an orthographic camera that maps (x, y, z) in space to (x, y) in the image plane (Figure 4).

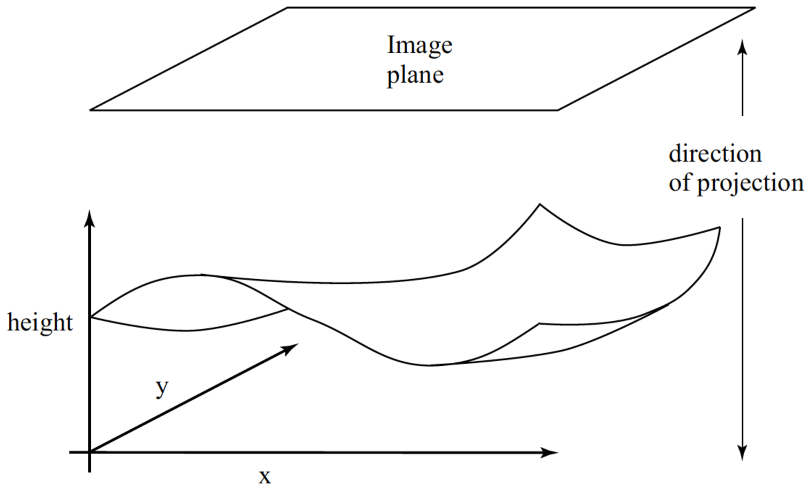


Figure 4: Orthographic camera model that maps (x, y, z) in space to (x, y). The camera is looking down from a positive z position.

Before measuring the shape of the surface, we need to introduce surface normal. Surface normal (Figure 5) is a vector that is perpendicular to the tangent plane to that surface at a point. Hence there is a surface normal at each point on a smooth surface.

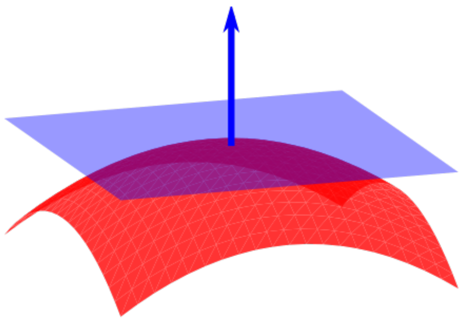


Figure 5: Surface normal (blue arrow) is perpendicular to the tangent plane (blue plane) to the surface (red) at a given point.

To measure the shape of the surface, we need to obtain the depth to the surface. Since the depth z can be related to each point (x, y), the surface can be represented as $s = [x, y, z(x, y)]^T$.
The partial derivative gives the rate of change in depth in either x and y direction:

$$s_x = [1, 0, p]^T$$

$$s_y = [0, 1, q]^T$$

where $p = \frac{\partial z(x,y)}{\partial x}$, $q = \frac{\partial z(x,y)}{\partial y}$.
Because s_x and s_y forms a tangent plane of the point on the surface, the normal can be obtained by calculating the cross product of s_x and s_y:

$$\begin{aligned} N &= s_x \times s_y \\ &= [1, 0, p]^T \times [0, 1, q]^T \\ &= [-p, -q, 1]^T \end{aligned}$$

From these we know we can recover depth via integration along x and y once we know the surface normal.

To obtain normal for each point on the surface, we need to further introduce the Lambertian model.
For simplicity, we only consider a local shading model, which means that shading is only caused by light that comes from the luminaire (no interreflections). Also we assume that the luminaire is an infinite distance source. Given these constraints, we adopt a Lambertian model such that the brightness at a point (x, y) on the surface is:

$$B(x, y) = \rho(x, y)N(x, y) \cdot S_1$$

where ρ is the albedo, N is the unit surface normal and S_1 is the light source vector. Albedo is used to describe the diffuse property of each point on the surface, ranging from 0 to 1. The higher the albedo, the brighter the point looks (reflects). S_1 tells the light source direction. We know from the dot product between N and S_1 that for each point on the surface the smaller the angle between its normal and light source vector, the brighter the point is. The brightest case for a point occurs when light source vector is parallel to its normal vector.
Assume the response of the camera is linear in the surface radiosity, the value of a pixel at (x, y) is:

$$\begin{aligned} I(x,y) &= kB(x,y) \\ &= k\rho(x,y)N(x,y)\cdot S_1 \\ &= g(x,y)\cdot V_1 \end{aligned}$$

where k is the constant connecting the camera response to the input radiance, $g(x,y) = \rho(x,y)N(x,y)$ and $V_1 = kS_1$. In this case, $g(x,y)$ describes the surface, and V_1 the property of the illumination and of the camera.

Given over three known light sources V_i and known pixel values $I_i(x,y)$, we can obtain $g(x,y)$ by solving an over-determined linear system using the least-square method. A more detailed solution will be explained in step 3.

Since $N(x,y)$ is the unit normal, we can extract albedo and surface normal from $g(x,y)$:

$$\rho(x,y) = |g(x,y)|$$

$$N(x,y) = \frac{g(x,y)}{|g(x,y)|}$$

According to previous deduction $N = [-p, -q, 1]^T$, we can obtain p and q:

$$p = -\frac{N_1}{N_3}$$

$$q = -\frac{N_2}{N_3}$$

Now we can obtain depth z by integration along x and y:

$$z(x,y) = \oint_C \left(\frac{\partial z(x,\ y)}{\partial x}, \frac{\partial z(x,\ y)}{\partial y}\right)\cdot d\boldsymbol{l} + \mathrm{c}$$

where C is a curve starting at some fixed point and ending at (x, y), and c is constant of integration, representing the height of the surface at the start point.

4. Practice

Step 1: Read in the images and light source directions.

We could have taken multiple images under different illuminations ourselves, but to simplify the process, we read in the images with existing light source vectors.

Besides the images, I've created a csv file named light_directions.csv in each subdirectory. This file contains the information about the light source direction vector for each image. For example, here is the csv file under res/synth-images/:

```
im1.png, 0, 0, 1
im2.png, -0.2, 0, 1
im3.png, 0.2, 0, 1
im4.png, 0, -0.2, 1
```

For each row, the first column is the name of the image, the second to fourth columns represent the x, y, z components of the light source direction vector.

Hence, we can first parse this csv file in our program so that we can easily know how many images we will read, and what is the light source vector corresponding to the image.

Step 2: Preprocessing

Once we read in the images and their corresponding light source vectors, we need to normalize the data to make sure our results are not affected by scaling.

1. Convert the images into grayscale if they are not (dog images are RGB) and normalize them into the range [0, 1].
2. Normalize each light source direction vector V so that |V| =1.

Step 3: Calculate albedo and surface normal

From previous theory explanation we know that to obtain albedo ρ and surface normal N, we need to calculate $g(x,y)$, and $g(x,y)$ is obtained by solving a linear system. Assume we have 4 images with different light source V_i (in fact we do have), we can stack these light source vectors into a known matrix $\mathcal{V}$:

$$\mathcal{V} = \begin{bmatrix} V_1^T \\ V_2^T \\ V_3^T \\ V_4^T \end{bmatrix}$$

For each image point (x, y), stack the pixel value of $I_i(x,y)$ in 4 images into a vector:

$$i(x,y) = \begin{bmatrix} I_1(x,y) \\ I_2(x,y) \\ I_3(x,y) \\ I_4(x,y) \end{bmatrix}$$

Now we have:

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

Before solving g, we need to take into account for the case when the image patch is in shadow (i.e. $I_i(x,y) = 0$).

We multiply left and right hand side with the following diagonal matrix:

$$\mathcal{T}(x,y) = \begin{bmatrix} I_1(x,y) & 0 & 0 & 0 \\ 0 & I_2(x,y) & 0 & 0 \\ 0 & 0 & I_3(x,y) & 0 \\ 0 & 0 & 0 & I_4(x,y) \end{bmatrix}$$

Now we can solve g for each point (x, y):

$$g(x,y) = (\mathcal{T}(x,y)\mathcal{V})^T\mathcal{T}(x,y)i(x,y)$$

The albedo and normal vector can be calculated by:

$$\rho(x,y) = |g(x,y)|$$

$$N(x,y) = \frac{g(x,y)}{|g(x,y)|}$$

We also store the p and q at each point for surface reconstruction in the next step.

We should also have another check that:

$$(\frac{\partial p}{\partial y} - \frac{\partial q}{\partial x})^2 \approx 0$$

This condition ensures that the surface is smooth.

Is is worth mentioning that in practice we have to consider the case when albedo is out of the range or is zero, as we will divide g by albedo to obtain the normal vector. I end up setting the normal vector to 0 if albedo equals 0 because very often this pixel is in the background, and setting albedo to 1 if it exceeds 1.

Step 4: Surface Reconstruction via Integration

From previous theory introduction, we know that the height of the surface can be recovered by integration.

The naïve direct integration approach can be found on page 50 in F&P textbook (2nd).

Here is the pseudocode:

```
First pre-allocate a 2D height map matrix.
Initialize height(1, 1) = 0 (Index starts at 1)
Integrate along y (vertical) direction only in the first column:
for i=2 : rows
    height(i, 1) = height(i – 1, 1)+ q(i, 1)
end
Integrate along x (horizontal) direction for each row:
for i=1 : rows
    for j=2 : columns
        height(i, j) = height(i, j – 1) + p(i, j)
    end
end
```

5. Improvements

Most of the improvements are done when dealing with dog images, as real images: (1) have noise; (20 the camera is not exactly an orthographic camera; (3) the Lambertian reflectance surface conditions may not fully be satisfied.

5.1 Integration using a mask

To avoid background noise and make reconstructed shape better, I create a binary image mask (Figure 6) to distinguish the object from the background. This is done using only 2 lines of code: convert the albedo map to binary image and then fill the holes if there is any (eye, ear).

```
bw = im2bw(albedo, 0);
mask = imfill(bw, 'holes');
```

During integration, I check if the the current location is in the mask (1) or not (0). If it's in the mask, then integration is performed; otherwise set the height to 0.

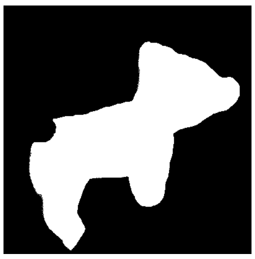


Figure 6: Dog images' binary mask

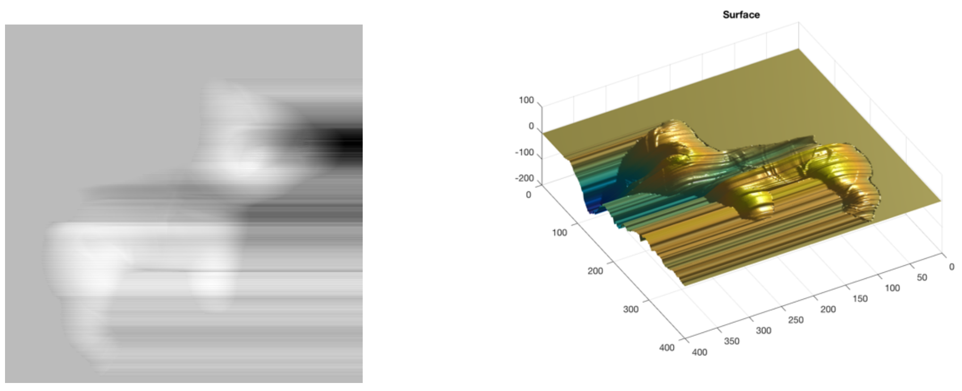
Figure 7 (a)(b) shows the improved result from height map and the reconstructed surface.

5.2 Reset background's height

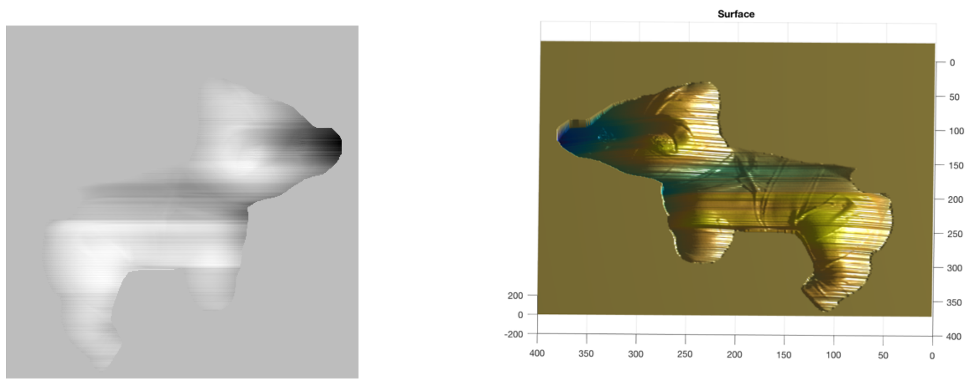
The top-left pixel value is initially zero. This means that the background (black) is zero. But I notice that part of the surface's height is below 0. So I reset the background's height to the lowest height of the surface. This better visualize the reconstructed surface (Figure 7 (c)).

5.3 Integration with the Chellappa's algorithm.

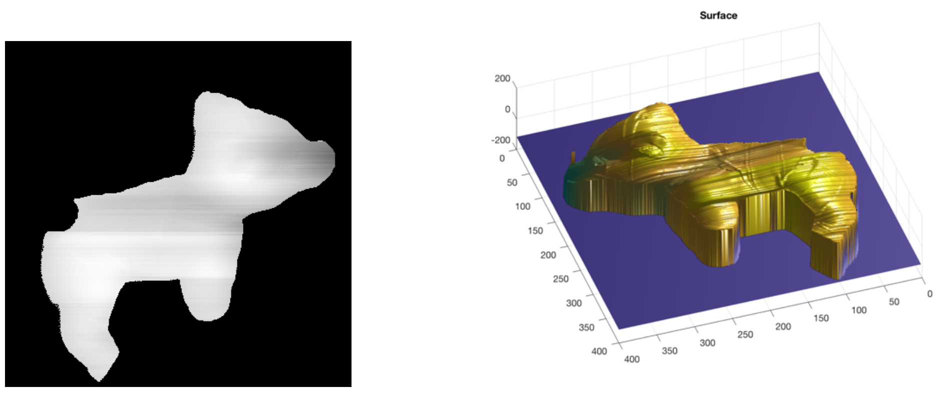
I also tried using Frankot and Chellappa's algorithm. The result can be seen in Figure 7 (d). The surface estimated by Chellappa's method is smoother.



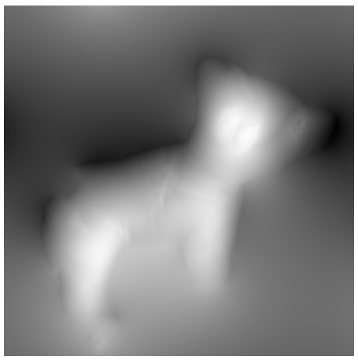
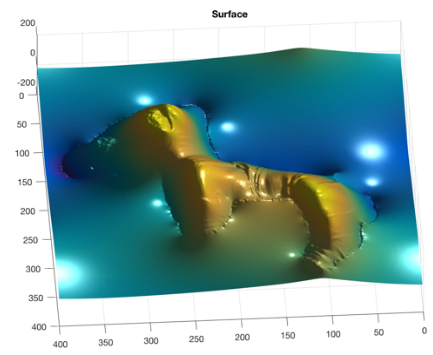
(a) Height map (left) and reconstructed surface (right) via Naïve integration



(b) Improved Height map (left) and reconstructed surface (right) using binary mask



(c) Improved Height map (left) and reconstructed surface (right) with reset background

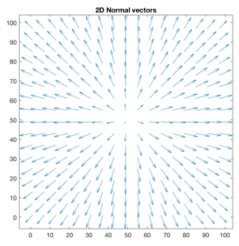
(d) Improved Height map (left) and reconstructed surface (right) with Chellappa's algorithm

Figure 7: A comparison in height map and surface between different methods

6. Results

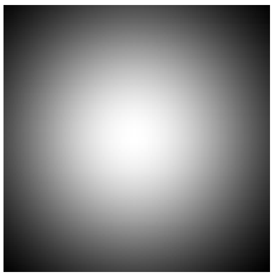
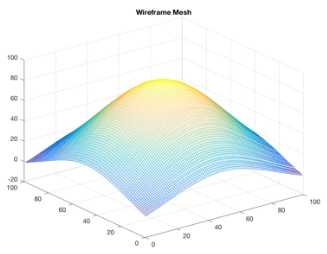
For this part I only show the final optimized results. The new shaded images are all under the light source $[1, 2, 3]^T$. Since Chellappa's result has been show for dog images before, I only show my results below.

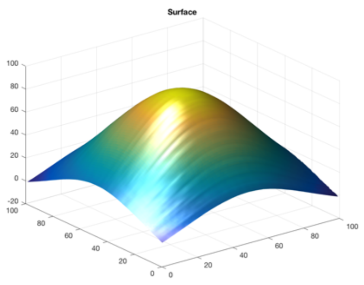
Synthesized images:



a. Albedo map b. 2D Normal vectors c. New shaded image

d. x, y, z components of surface normal

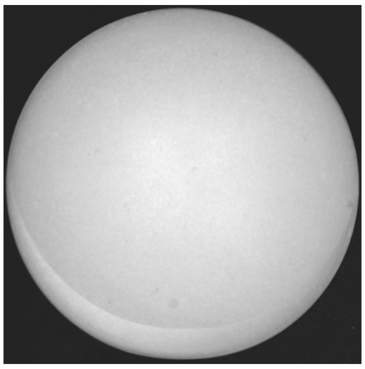





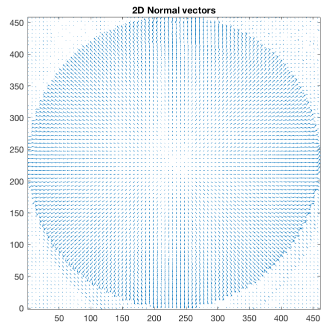
e. Height map f. Mesh g. Surface

Figure 8: Estimated albedo, normal, height map, and reconstructed surface for synthesized images

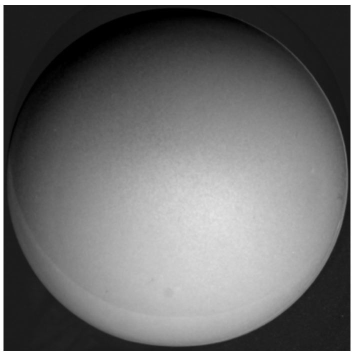
Sphere images:



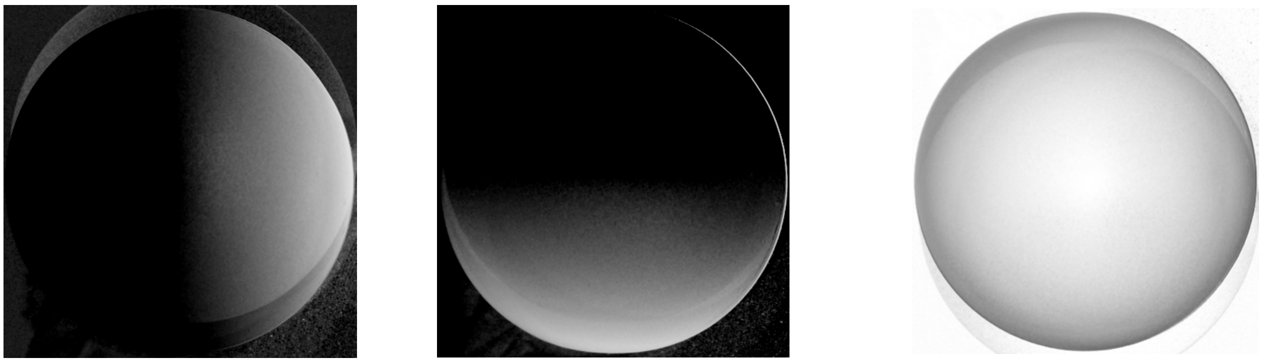
a. Albedo map



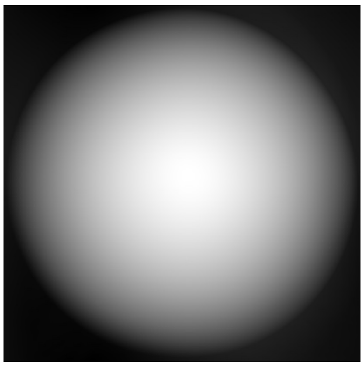
b. 2D Normal vectors



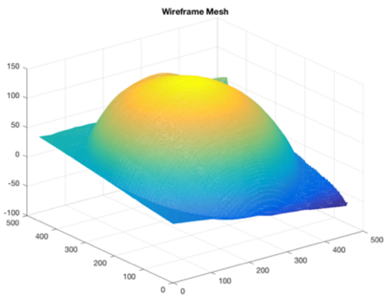
c. New shaded image



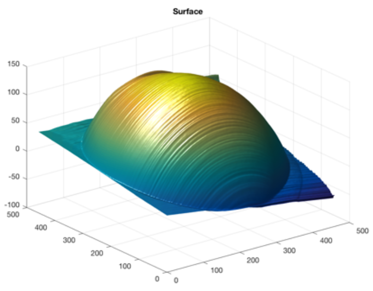
d. x, y, z components of surface normal



e. Height map



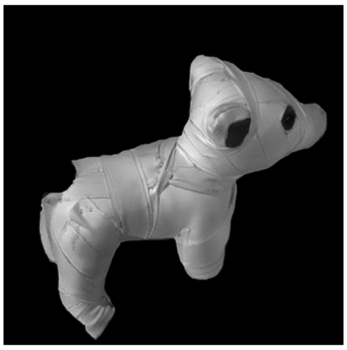
f. Mesh



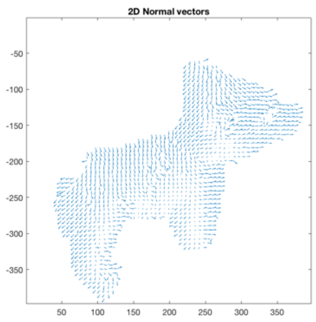
g. Surface

Figure 9: Estimated albedo, surface normal, height map and reconstructed surface for sphere images

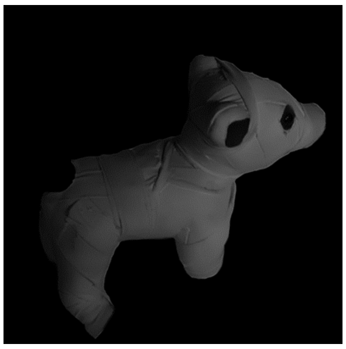
Dog images:



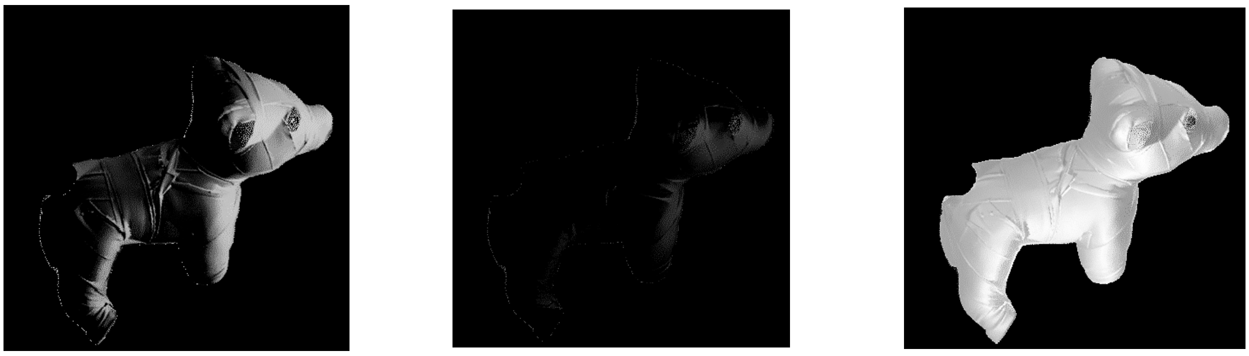
a. Albedo map



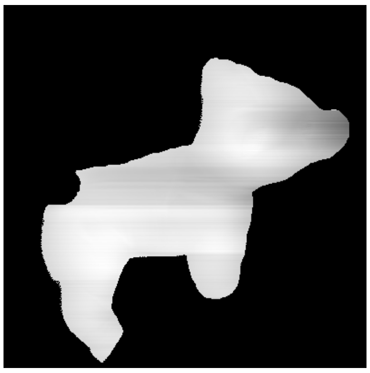
b. 2D Normal vectors



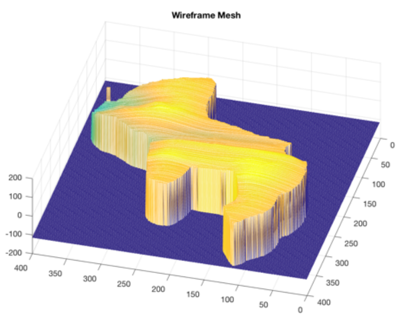
c. New shaded image



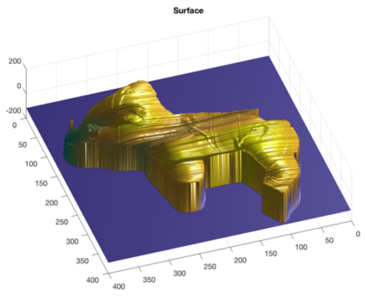
d. x, y, z components of surface normal



e. Height map



f. Mesh



g. Surface

Figure 10: Estimated albedo, surface normal, height map, and surface for dogs images

7. Issues

It is found that when solving g using least square method, it seems that removing the diagonal matrix T may have better result.

Solution 1:

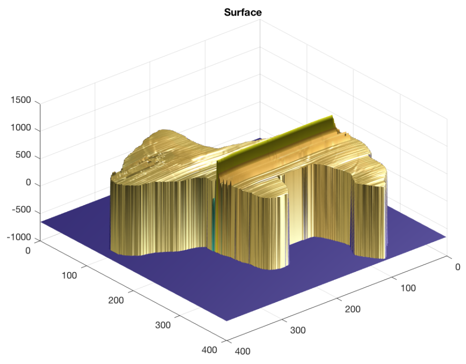
```
        T = diag(i);
        g = pinv(T * V) * (T * i);
```

Solution 2:

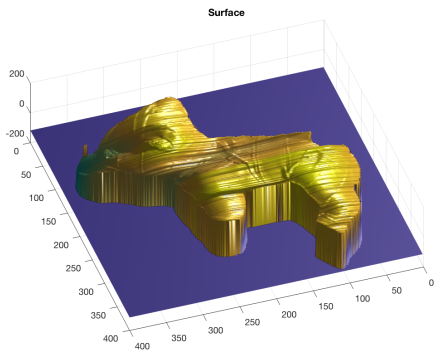
```
        g = pin(V) * i;
```

Solution 1 uses the matrix trick method mentioned in the class, by constructing a diagonal matrix of the stacked image vectors, and multiplying left and right hand side, and finally solving g using pseudo-inverse of matrix T*V (or performing SVD). Solution 2 does not apply the matrix trick.

From Figure 11 and Figure 12 we can see that without this diagonal matrix multiplied, the height is obtained more accurately. The diagonal matrix seems to reduce the difference between heights.

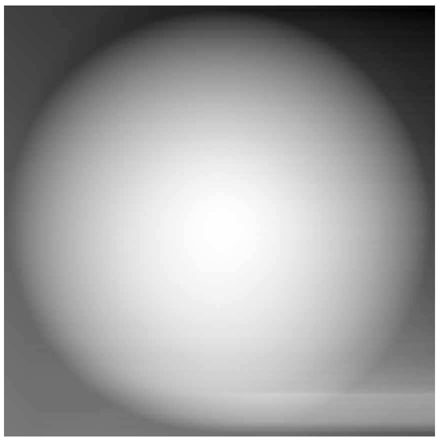


a. Solution 1

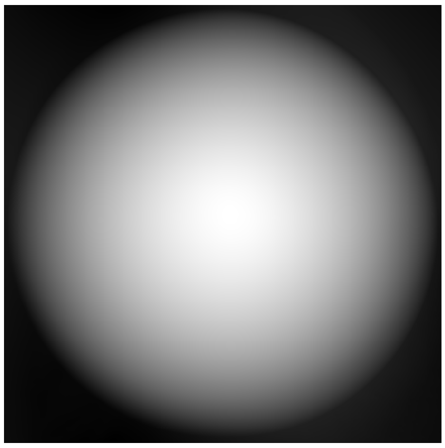


b. Solution 2

Figure 11: Reconstructed dog surface with and without T for solving g (normal * albedo)



a. Solution 1



b. Solution 2

Figure 12: Sphere's height map with and without T for solving g (normal * albedo)

File Structure

```
src
├──── computeAlbedoSurfNorm.m
├──── computeHeightMap.m
├──── frankotchellappa.m
└──── main.m
res
├──── dog-images
│     ├──── dog1.png
│     ├──── dog2.png
│     ├──── dog3.png
│     ├──── dog4.png
│     └──── light_directions.csv
├──── sphere-images
│     ├──── light_directions.csv
│     ├──── real1.bmp
│     ├──── real2.bmp
│     ├──── real3.bmp
```

```
|     └──── real4.bmp
└──── synth-images
      ├──── im1.png
      ├──── im2.png
      ├──── im3.png
      ├──── im4.png
      └──── light_directions.csv
```

Under the **src** directory, there are 4 files:
computeAlbedoSurfNorm.m computes the albedo and surface normal;
computeHeightMap.m computes height via naïve integration;
frankotchellappa.m computes height using Frankot and Chellappa's algorithm;
main.m combines the above functions and shows results.

Under the **res** directory, there are 3 different sets of images in separate subdirectories: **dog-images**, **sphere-images**, and **synth-images** as well as the corresponding light direction vectors mapping in their **light_directions.csv** file.