Photometric Stereo

Abstract: this report shows the process of photometric stereo. Basically, the whole process can be divided into 2 parts, the first step is to calibrate the light and the second is to recover the object in the picture, like Albedo, norm map and depth value.

I. Light Calibration

Since light direction is not calibrated, we need to calibrate light direction first.
For each folder of an object, we totally have 4 types of images, the object under different illumination condition, the specular sphere, the Lambertian sphere and their corresponding mask images.

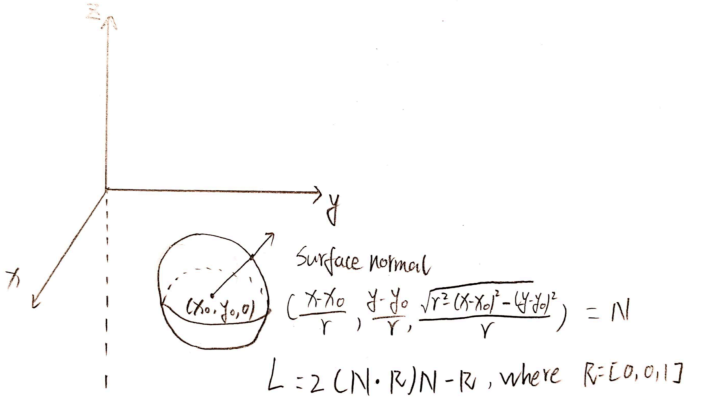
1. Light Intensity

In order to determine the intensity for each image, we need to implement the information provided by Lambertian model. Since the observed intensity is up to $L_o = L_i\rho(N * L)$, where N, L represent the unit vector of the surface norm and incoming light direction. Since it is Lambertian model, which also means that the pixel brightness is not related to the viewing angle, when $(N * L) = 1$, L_o should be the incoming light intensity L_i with a scalar ρ. Since ρ is just a constant. Therefore, the light intensity can be solved by finding the brightest point on the Lambertian model. The below one is the example of elephant light intensity. Below is 18 light intensity from the first 18 image in apple. In case of redundancy, I just show 16 result.

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0.0268	0.0254	0.0270	0.0294	0.0291	0.0187	0.0275	0.0260	0.0106	0.0102	0.0174	0.0230	0.0260	0.0221	0.0200

2. Light direction

For light direction, it is a little bit complex since it requires some geometry knowledge. The following figure shows the way how I solve the light direction. Below is 18 light direction from the first 18 image in apple. In case of redundancy, I just show 18 result.



	1	2	3
1	-0.3058	0.3738	0.8757
2	0.0795	0.4545	0.8872
3	-0.0272	0.9791	0.2016
4	-0.2591	0.2246	0.9394
5	0.2626	0.3311	0.9063
6	0.4042	0.3537	0.8435
7	-0.3110	0.1382	0.9403
8	0.2078	0.2424	0.9477
9	0.4389	0.2870	0.8515
10	-0.3229	0.0577	0.9447
11	0.2772	0.1386	0.9508
12	0.0652	-0.6845	0.7261
13	-0.3783	-0.0860	0.9217
14	0.1392	-0.2088	0.9680
15	0.3748	-0.2726	0.8861
16	-0.2391	-0.3757	0.8954
17	0.1701	-0.4423	0.8806
18	0.3699	-0.4035	0.8369

Suppose the specular sphere lies on the $X - Y$ plane. Therefore, the relationship of surface norm can be obtained by the formula $(x - x_0)^2 + (y - y_0)^2 + z^2 = r^2$, so the surface norm is

$(\frac{x-x_0}{r}, \frac{y-y_0}{r}, \frac{\sqrt{r^2-(x-x_0)^2-(y-y_0)^2}}{r})$. With surface norm, we can simply derive the relationship between light direction and surface norm which is $L = 2 * (N * R) * N - R$, where R is [0,0,1].

II. Recover albedo map, normal map and rendered image

After calibrating light intensity and light direction, we have all prerequisites to build a linear equation system to get Albedo and normal map.

The intensity of a pixel is determined by the Lambertian model I mentioned before $I = \rho(N * L)$. We need to recover the Albedo ρ and the surface normal N, that is to say, we totally have 3 unknows for each pixel. In order to solve this equation, we at least need 3 pictures under different illumination condition. However, in order to minimize error as much as possible, we should use way more than 3 images. In my project, I used all images.

To be more specific, for each pixel in the image, we can build up a linear equation system, in my case, every three unknows can be solved by 21 equations, it is an over-determined problem and can be solved using linear least square.

After having albedo map and normal map, the re-rendered image is pretty easy to recover. Besed on $I = \rho(N * L)$, we just need to plug in $L = [0,0,1]$, so basically, the re-rendered image is $\rho * N_z$.

III. Recover surface of the object and height map

In this part, I use the method II to recover the surface of the object. The basic idea of method II is that the surface norms are perpendicular to the surface. To be exact, normal are perpendicular to any vectors on the surface.

In mathematics, for each pixel(x, y) on the image, there are two neighboring pixel containing tangent vectors, so basically, we can model 2 equations to one pixel, which are $N_z(z_{xy} - z_{x+1,y}) = N_x$ and $N_z(z_{xy} - z_{x,y+1}) = N_y$. For example, if the dimension of the image is 600*800, we totally have 600*800 = 480000 unknows to recover, meanwhile, we have 480000*2 = 960000 equations. (actually, it is not exactly correct and I will discuss it in the next section.). One point that need to be specified is this is going to be a super big matrix and generally speaking, our computer cannot handle such size of matrix. Then, how should we solve it? This will be discussed in the next section.

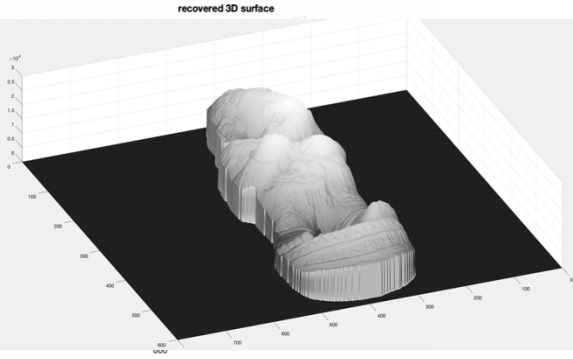
IV. Results and discussions

For each object, there are five results (from left to right, from top to bottom): the normal map encoded in RGB, the albedo map, and the re-rendered picture of the object using the recovered normal and albedo under illumination direction that is the same as the viewing direction ([0, 0, 1]), the recovered surface and the height map.

Normals

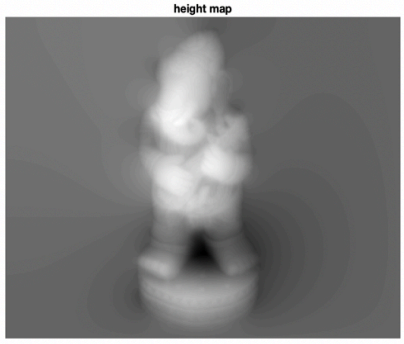
Albedo

recovered 3D surface

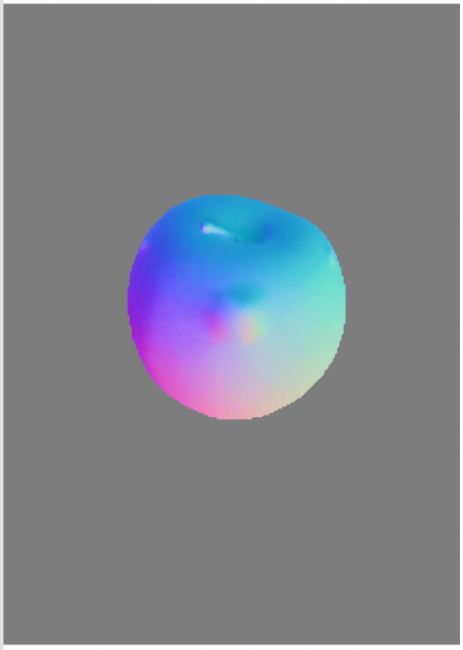


rerendered image with the same direction of light and viewing

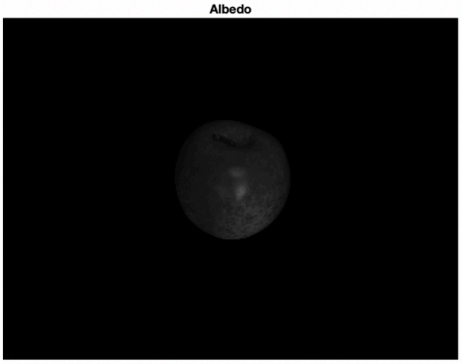
height map



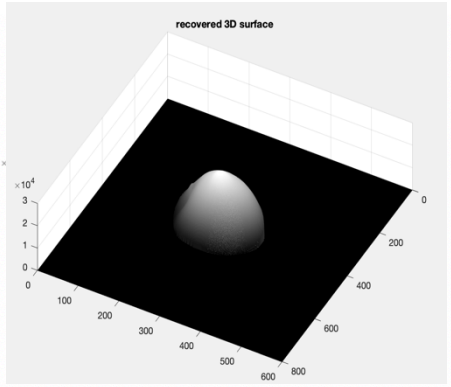
Normals

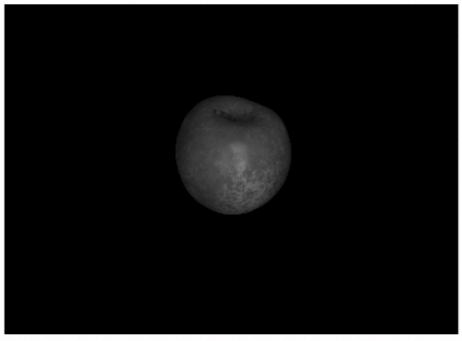


Albedo

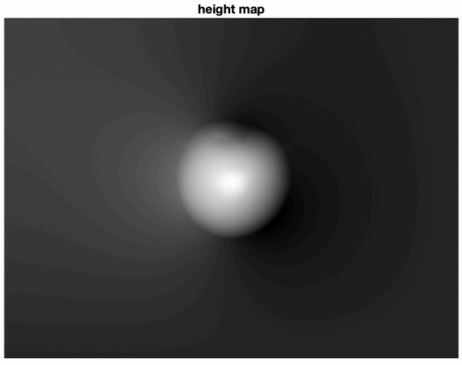


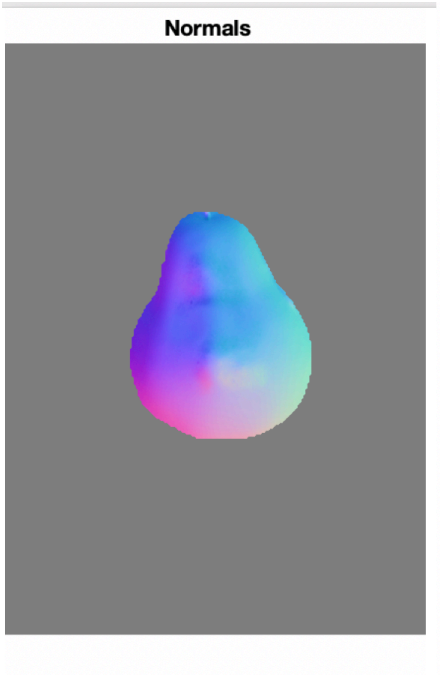
recovered 3D surface

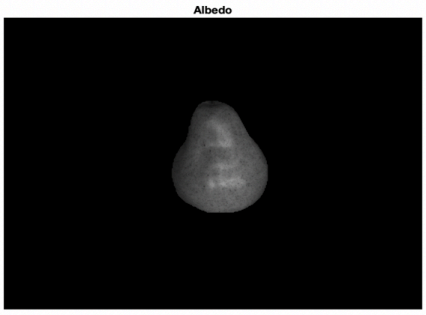


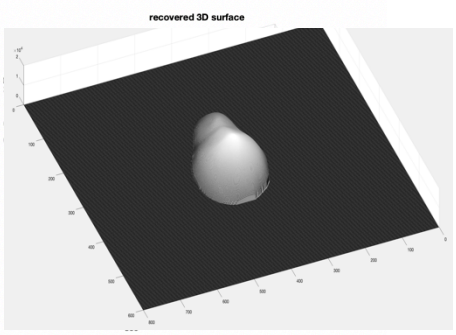


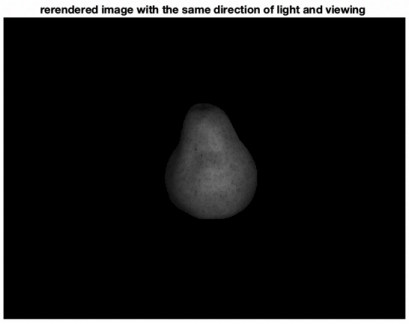
height map

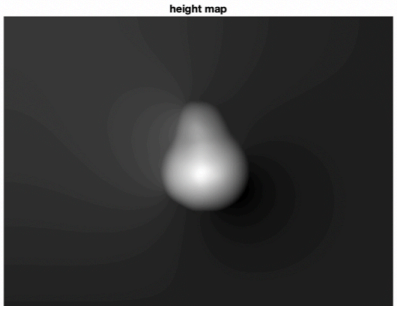












1. What makes trouble to photometric stereo?

   Actually, in this project, all conception, mathematics and formula are pretty simple. I got so many technical problems and i also think it is worth mentioning that.

   However, the first and the most difficult problem is how to build the linear equations in recovering height map part. Basically, I need to build a linear equation system in the following way:

$$\begin{bmatrix} a_{1,0} & a_{1,1} & \cdots & a_{1,w\times h} \\ a_{2,0} & a_{2,1} & \cdots & a_{2,w\times h} \\ a_{3,0} & a_{3,1} & \cdots & a_{3,w\times h} \\ a_{4,0} & a_{4,1} & \cdots & a_{4,w\times h} \\ \vdots & \vdots & \ddots & \vdots \\ a_{w\times h\times 2,0} & a_{w\times h\times 2,1} & \cdots & a_{w\times h\times 2,w\times h} \end{bmatrix} \begin{bmatrix} H(1,1) \\ H(1,2) \\ \vdots \\ H(w,h) \end{bmatrix} = \begin{bmatrix} N_x(1,1) \\ N_y(1,1) \\ \vdots \\ \vdots \\ N_x(w,h) \\ N_y(w,h) \end{bmatrix}$$

   Like I mentioned before, the first is to find the pattern of the matrix, the relationship of the height and norm is Nx(i, j) = H(i, j) − H(i, j + 1) and Ny(i, j) = H(i, j) − H(i + 1, j). However, this pattern does not hold for pixels at the right borders and the bottom border since pixels at these two positions does not have the right neighbors or the downside neighbors or neither. Instead, Nx(i, j) = H(i, j) or Ny(i, j) = H(i, j) works for these specific position.

   After finding the pattern, another problem is how to handle such a big dimension matrix. In our case, the image size is 600*800 or 800*600. That is to say, the size of the coefficient matrix is 960000*480000 which is out of computer memory. After I went through some stuff, I noticed, in MATLAB, sparse matrix could help me out. Since there are only two values, 1 or -1 in each row of our coefficient matrix.

Therefore, only 1920000 numbers are non zeros (actually less than this number). Sparse matrix is like a dictionary, so it only stores values with its index when the value is non zeros, that is why it can handle a large size of matrix.

I thought I solved all problems when I got sparse matrix. However, things are out of my expectation, since it took almost 10 minutes to run generate the coefficient matrix. Followed by the reminder from MATLAB, I realized that if we update the sparse matrix every time, it is going to be very time-consuming. So, the best way is to store the value and index first, then plug in to the sparse matrix at one time.

2. What kind of data works best/ worst?

   Typically, when we calculate the normal map and albedo map, we assume that the object that we recover is absolutely Lambertian model, however, in reality, it is not exactly correct since objects are more like the linear combination of Lambertian model and specular model. In other words, if we just recover objects based on the Lambertian model, then if the object itself is more Lambertian than phong, then the result is better. The example is that the result of elephant is better than apple. Sicne apple's texture is more like a phong model.

   And another thing is that, if we notice the recovered surface of the apple and pear, we can find that the height of brightest surface is really high, that is because the texture of apple and pear is more like phong model, so that brightest spot's reflected light is super high and the recovered surface is stretched to a relatively high level.

3. How the implemented algorithm can be improved?

   I think a way to improve the performance is to use more complicated model especially for non-lambertian model. Like using linear combination to put Lambertian and non-lambertian together. We can also add term to albedo which make albedo spatially variant.