CSC2503 Fall 2018: Homework 3

Niloufar Afsariardchi

December 4, 2018

1 Part A-1

I made a camera obscura using a box and aluminum sheet for making different apertures. The scene is a bright spiral lamp (see Figure 2 (a) and (b)). I set the scene fixed for parts A-1 and A-2 of this assignments. I took the images formed in the camera obscura using a DSLR Canon camera. The camera setting (iso, exposure time, camera position, etc) was also fixed for parts A-1 and A-2 of this assignments. I shaded the lamp scene using a tarp so that I could take a bright enough photo from the image (see Figure 2 (f)) formed inside the the camera obscura. Figure 2 (c) shows a very sharp and upside-down image of the spiral lamp formed inside the camera obscura.

2 Part A-2

The results of this part is presented in Figure 2. The setup, which is the same as part A-1, is shown in Figure 1. As can be seen, as be we increased the pinhole size the image became very blurry, but its brightness increased compared to small pinhole case. Similarly, different pinhole shapes also made the the image blurry but I believe with different blurriness pattern. Among Figure 1 images, the image with hexagon and cross pinhole are, respectively, the most and least blurry.

3 Part A-3

I noticed that the pinhole of the camera acts like a filter for the image: when the pinhole is very small, its shape is closest to delta-dirac function, and therefore it seems like the image is being convolved with a 2D dirac function, which results in the image itself. Although there needs to be constant term which sets the overal brightness of the image (because not all of the radiated light enters the pinhole). The convolution operation here makes sense because we can think of the large pinhole as a superposition small pinholes, each of these smaller pinholes would create an image inside the camera obscura that is shifted due to displacement of small pinholes. The final image is superposition of all these images. The described operation is convolution for a linear shift invariant (LSI) system. Therefore we can write:

$$I_S(x, y) \quad = \quad I(x, y) * S(x, y) \tag{1}$$

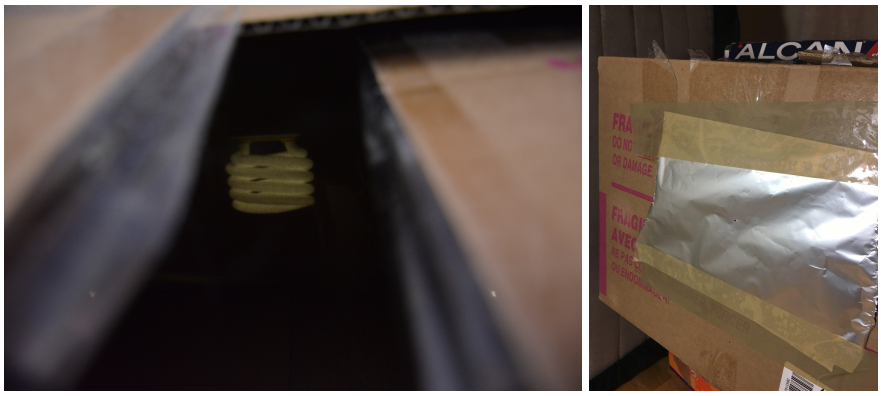
or equivalently:

$$\tilde{I}_S(u,v) \quad = \quad \tilde{I}(u,v).\tilde{S}(u,v) \tag{2}$$

in the frequency domain where the parameters with tilde are in Fourier transform of the corresponding parameter in spatial domain. When we use larger pinhole, it is like making a smaller Gaussian shape filter in the frequency domain whic filters out the high frequency signals (for example the edges), hence the image becomes more blurry. A similar operation is done in actual cameras as well as for telescope where the impulse response function of the camera called point spread function (PSF) depends on exit pupil shape [1]. I think for this question parallelism and lambertian relectance ensures the system to be LSI. Lambertian relectance is importante because as we increase the the pinhole size, each position on the image receives light from different direction and different scene locations, therefore if the relectance is not Lambertian we cannot simply assume that the formed image is the superposition of smaller pinhole sizes (the system won't be linear). Parallelism and planar scene on the other hand, I believe, ensure the system being shift invariant. If we shift the scene (or the pinhole) in a camera obscura that is not parallel, the formed image will be a function of spatial location as it should be projected on the back of the box. Without these assumption we cannot write equation (1) with convolution operation.

[1] `http://web.ipac.caltech.edu/staff/fmasci/home/astro_refs/PSFtheory.pdf`

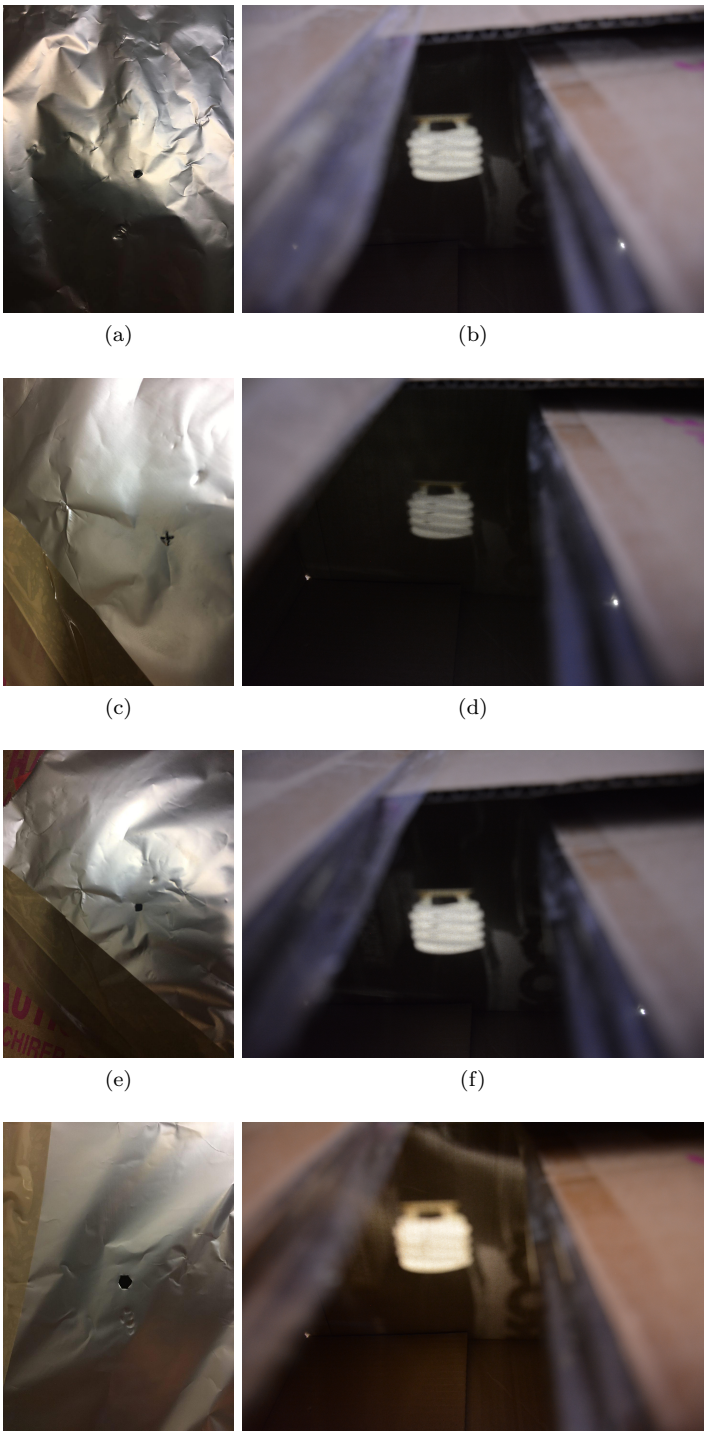
(a) (b)



(c) (d)

(e) (f)

Figure 1: (a) scene in front of the camera obscura (b) scene in front of the camera obscura (c) small pinhole of ∼2 mm diameter (d) an image formed inside the the camera obscura for a small pinhole (e) the setting (f) shading the scene for the the purpose taking photo



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 2: Left columns: the shape of pinhole (4mm disc, 4mm*4mm cross, 4mm*4mm square, 5mm hexagon diamter) Right column: the formed images. Note that the scene was exactly the same of part A-1

(a)

Figure 3: scene in front of the camera obscura which is the same as part A-1

4 Part B-1

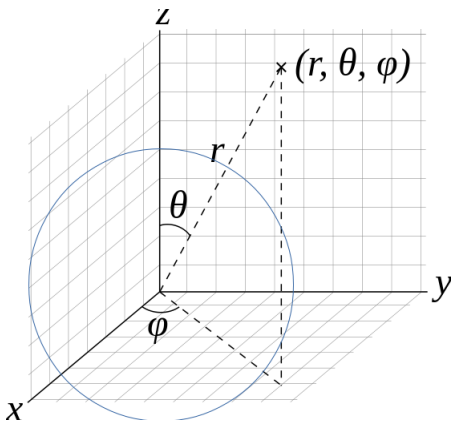


Figure 4: Spherical coordinates. In our case, coordinate z is along camera direction and (0,0) is the center of chrome sphere in the XY plane

In the spherical coordinates defined above, the normal to the sphere $\vec{n}$ is along $\hat{r}$ (both vectors are unitary) which could be written as

$$\vec{n} = \begin{pmatrix} \cos\phi \sin\theta \\ \sin\phi \sin\theta \\ \cos\theta \end{pmatrix}$$

If the brightest pixel in the image of chrome sphere has (x,y) location and r is radius of the sphere, then the term is the $\vec{n}$ are

$$\cos\phi = \frac{x}{\sqrt{x^2+y^2}} \tag{3}$$

$$\sin\phi = \frac{y}{\sqrt{x^2+y^2}} \tag{4}$$

$$\sin\theta = \frac{\sqrt{x^2+y^2}}{r} \tag{5}$$

$$\cos\theta = \frac{\sqrt{r^2-x^2+y^2}}{r} \tag{6}$$

Inserting equations 3–6, into the definition of $\vec{n}$,

$$\vec{n} = \begin{pmatrix} \cos\phi \sin\theta \\ \sin\phi \sin\theta \\ \cos\theta \end{pmatrix} = \begin{pmatrix} \frac{x}{r} \\ \frac{y}{r} \\ \frac{-\sqrt{r^2-x^2+y^2}}{r} \end{pmatrix} \tag{7}$$

The reason that there a negative sign behind the z component of $\vec{n}$ is because the camera is at (0,0,-1) and this the light sources should be on the negative side of the z-axis. In order to compute $\vec{n}$, we first need to measure the radius of the sphere, and the brightest pixel $(x, y)_i$ for each light source. Using 7, we can compute $\vec{L}$ assuming perfect specular reflectance (from class notes on

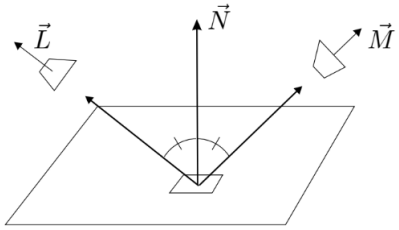


Figure 5: Perfect Specular Reflectance. Image taken from class notes. $\vec{n}$ is the surface normal, $\vec{L}$ light source direction, and $\vec{m}$ is the camera optical axis direction

Dropbox). Since we have the orthographic projection assumption, we know the camera direction is constant and equal to $\vec{m} = (0, 0, -1)$. According to Figure 5 we derive:

$$\vec{m} + \vec{L} = 2(\vec{m}.\vec{n})\vec{n} \quad (8)$$

$$\vec{L} = 2(\vec{m}.\vec{n})\vec{n} - \vec{m} \quad (9)$$

Altogether the algorithm to find $\vec{L}$ has these steps:

1. First, round the chrome sphere pixels to be either 0 or 1
2. Derive the center of chrome sphere $x_c = 0.5(x_{\min} + x_{\max}) = 254.5$ and $y_c = 0.5(y_{\min} + y_{\max}) = 149$, where $x_{\min}$ and $x_{\max}$ are the index of a pixel on the sphere that has minimum and maximum x value. The $y_{\min}$ and $y_{\max}$ are defined similarly
3. Compute sphere radius $r = \text{avr}(0.5(x_{\max} - x_{\min}), 0.5(y_{\max} - y_{\min})) = 118.75$
4. Measure the coordinates of the brightest pixel of each chrome sphere image using
```
siz=size(imData);
idx=find(imData==255);
[ xx,yy,z ]=ind2sub(siz,idx);
for n=1:nDir
x(n)=mean(xx(z==n));
y(n)=mean(yy(z==n));
end
```
This code takes the average of all brightest pixels with maximum value of 255.
5. Translate x and y of brightest pixel of each image so that the center of sphere becomes the origin of x- and y-axis and the compute $\vec{n}$ using Equation 7, we normalized it by its norm to get a unit length vector:
```
n=[((x-x_c)/r)',((y-y_c)/r)',-(sqrt(r^ 2-((x- x_c).^ 2+ (x-x_c).^
2))/r)'];
n=n./repmat(sqrt(sum(n.^ 2,2)),[1 3]);
```

6. Compute $\vec{L}$ using Equation 9

```
L=2*(repmat(dot(n,m,2),[1,3])).*n-m;
L=L';
```

In Figure 6, you can see the projection of light sources on sources, which seem to be consistent with the images provided to us. Here is the matrix of 8 light source directions:

$$\vec{L}^T = \begin{pmatrix} 0.4957 & -0.4721 & -0.7289 \\ 0.2404 & -0.142 & -0.9602 \\ -0.04256 & -0.1798 & -0.9828 \\ -0.09667 & -0.4376 & -0.894 \\ -0.3175 & -0.5041 & -0.8032 \\ -0.1109 & -0.5469 & -0.8298 \\ 0.2755 & -0.4238 & -0.8629 \\ 0.09541 & -0.4276 & -0.8989 \end{pmatrix} \tag{10}$$

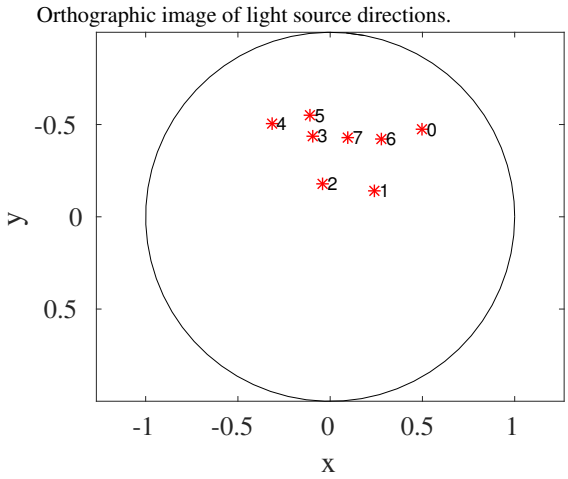


Figure 6: Image of light sources found from the function fitChromeSphere.m

Question B-2

We first formulate the optimization problem as:

$$e_{xy} = I_{xy} - \vec{g}_{xy} L$$

Where L is 3×8 matrix of lightsource directions and I_{xy} is 1×8 matrix that has the intensity of xy pixel for all 8 lightsources, $\vec{g}_{xy}$ is 1×3 vector where $a_{xy} = \|\vec{g}_{xy}\|$ and $\vec{n}_{xy} = \vec{g}_{xy} / \|\vec{g}_{xy}\|$

Therefore

$$O = \sum_x \sum_y e_{xy} \cdot e_{xy}^T = \sum_x \sum_y (I_{xy} I_{xy}^T + \vec{g}_{xy} L L^T g_{xy}^T - 2 g_{xy} L I_{xy}^T)$$

Taking derivative of O and equivalenting to zero,

$$\frac{\partial O}{\partial \vec{g}_{xy}} = 2 L L^T g_{xy}^T - 2 L I_{xy}^T = 0$$

$$\Rightarrow \boxed{g_{xy}^T = (L L^T)^{-1} L I_{xy}^T}$$

Now for all pixels N:

$$\boxed{(g^T)_{3\times N} = (L L^T)^{-1} L (I^T)_{8\times N}}$$

5 Part B-2

Here is my short code for fitReflectance.m:

```
g=inv(L*L')*L*im';
albedo=sqrt(sum(g.^ 2,1))';
n=g'./repmat(sqrt(sum(g'.^ 2,2)),[1,3]);
```

In Figure 7–10, I showed the results of fitReflectance.m that I got on a sample of images. I selected these images because they reflect the error types better. The results are fairly good, and the recovered image is *very* similar to the original image. However, the RMS plots show that the error is particularly high for the parts of the object that are in shade for certain light directions. This is particularly clear in the results of sphere images (Figure 9). We can see that *southern* edge of the sphere has very high RMS error, while the northern edge has not. I note in Figure 6 that all light sources are shining light from northern hemisphere, hence shading the southern edge of the sphere. Another sample can be seen in Figure 8, where the self-shadow is casted on the legs of the horse and results in high RMS in gray albeido. I also note that the error is most negative for each light source when dot(n,L) is negative. In another words, in these cases the light source is behind the the surface of object in that particular direction. This contradicts the assumption in the question that "the light source is positioned such that no visible piece of the surface is oriented away from the light source." Therefore we get high negative error where this does not hold as the code underpredict the albeido. In addition, we can see that error is highly *positive* for southern edge of the sphere even when dot(n,L)>0 (see Figure 9, panel (f)). These erroneous parts are located at the positions where the angle between the light source direction and surface normal is wide, which could be due to breakdown of Lambertian reflectance assumption when the normal and light direction make a wide angle. In addition to over- and under- estimation in albeido, the algorithm does not obtain very accurate surface normals either. This is particularly apparent in Figure 9(b). This object is spherically symmetric by the surface normal, especially n_y does not entirely look symmetric along y-axis.

Figure 7: Results of gray albedo on cat image

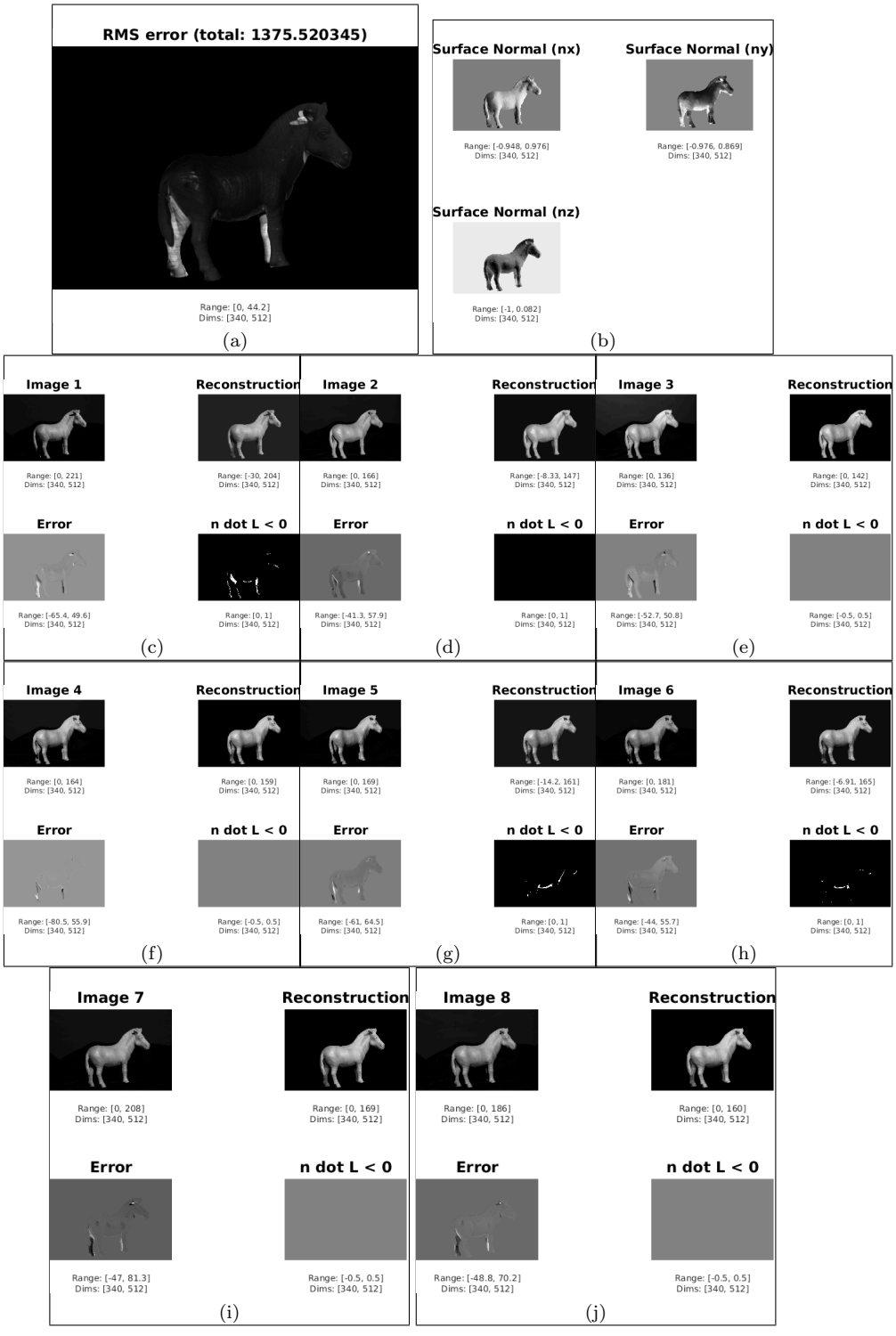


Figure 8: Results of gray albedo on horse image

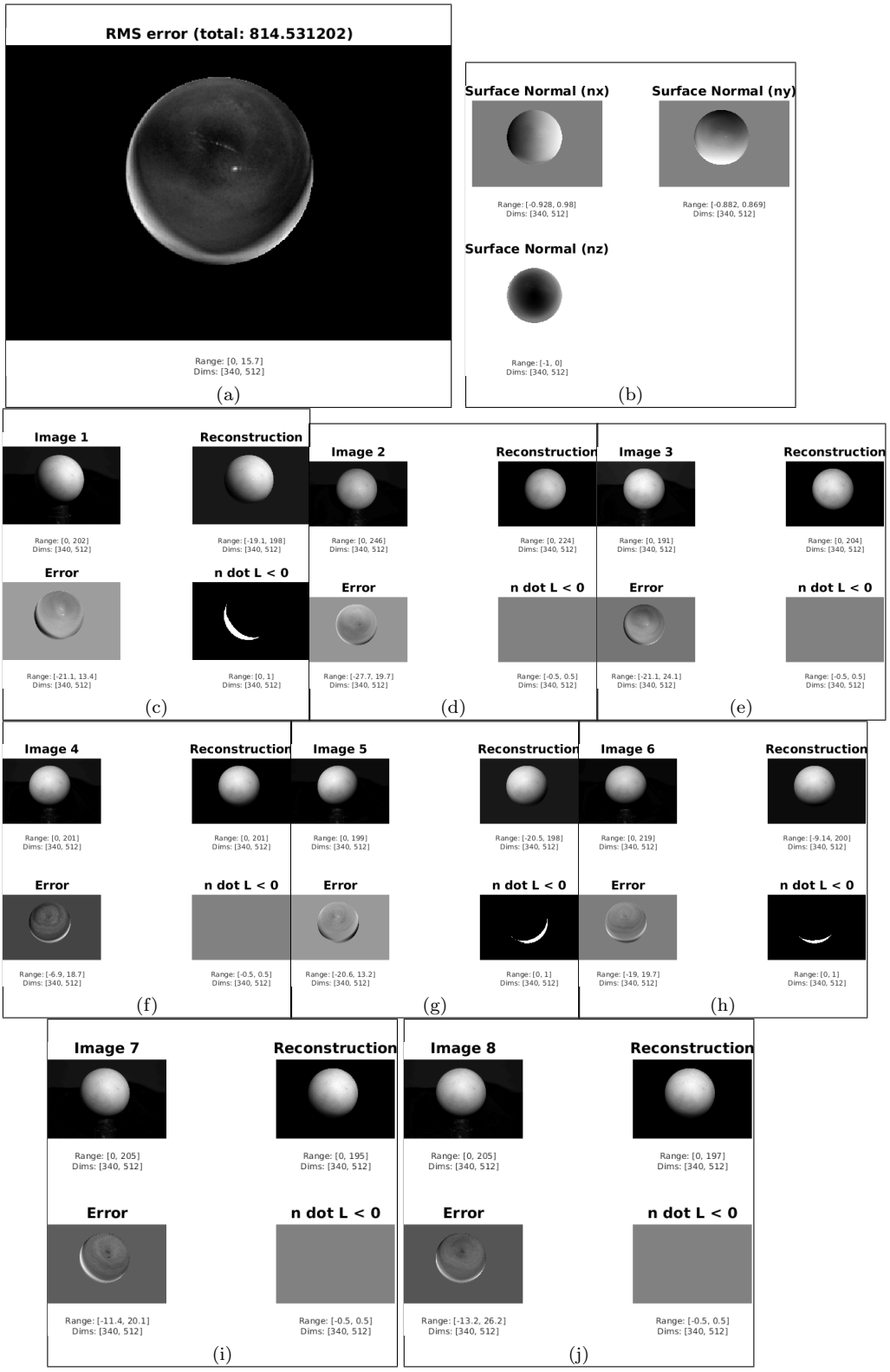


Figure 9: Results of sphere albedo on gray image

Figure 10: Results of gray albedo on owl image

Question B-3

We first write the error as:

$$e_{xy,v} = I_{xy,v} - a_{xy,v}\, g_{xy}$$

where $I_{xy,v}$ is a 1×8 array of intensities, $a_{xy,v}$ is a scalar value denoting albeido, $g_{xy} = n_{xy}.L$ is a 1×8 array for all light sources.

Therefore,

$$O = \sum_{v}\sum_{x}\sum_{y} e_{xy,v}\, e_{xy,v}^{T} = \sum_{v}\sum_{x}\sum_{y} I_{xy,v} I_{xy,v}^{T} + a_{xy,v}^{2}\, g_{xy} g_{xy}^{T} - 2a_{xy,v}\, g_{xy} I_{xy,v}^{T}$$

We take its derivative and equating it to zero:

$$\frac{\partial O}{\partial a_{xy,v}} = 2a_{xy,v}\, g_{xy} g_{xy}^{T} - 2 g_{xy} I_{xy,v}^{T} = 0$$

$$\Rightarrow \boxed{a_{xy,v} = \frac{g_{xy} I_{xy,v}^{T}}{g_{xy} g_{xy}^{T}}} \quad (1)$$

Therefor we can write (1) in terms of elemetwise operations in matlab without using a for loop on x & y

$$a_v = \frac{\text{dot}(g, I_v, 2)}{\text{dot}(g, g, 2)}$$

where g is $N_{pixel}\times 8$
I_v is also $N_{pixel}\times 8$
a_v is $N_{pixel}\times 1$

6 Part B-3

This my short code for this part:

```
for i=1:3
RGBimcrop(:,i,1:nDirChrome) = RGBim(mask,i,1:nDirChrome);
im=reshape(RGBimcrop(:,i,:),sum(mask),nDirChrome);
g=nCrop*L;
albedo(mask,i)= dot(g,im,2)./dot(g,g,2);
end
```

The results are shown in Figures 12–17. There recovered RGB albeido seems to fairly respresent the colors of the objects, however there seems to exhibit some over-estimations in the parts that were in shade perhaps to the simplifying Lambertian reflectance assumption. In particular, Figure 12 (a) shows very bright neck and left ear because these parts were mostly in shade according the directions of light sources. Similarly, the left owl wing joint shows higher albeido than expected. These over-estimations in albeido are also shown in the synthetically shaded images, especially in cases were the over-estimation parts should be in shade (for example Figure 12(b)), the over-estimation in albeido makes the parts so bright as if they are not in shade. The question asks about other sources of error assuming that the recovered albedos and surface normals are correct. It is hard to distangle the error in albedos and surface normals from other errors, but if we check the areas where albedo calculation does not have high error, we can tell if the synthetic image making works well in general. I believe the algorithm does a very good job in making synthetically shaded images assuming that the recovered albedos and surface normals are correct. It casted shadow with respect the shape of the objects and is even successful in making correct self-shadows, for example in Figure 12 (c) the algorithm successfully shaded the left leg of the cat as the light is shinning from left direction. The same could be seen for the ear of horse and the right leg of the owl. Note that the light direction for the shown synthetically shaded images are marked in Figure 11 with magenta color.

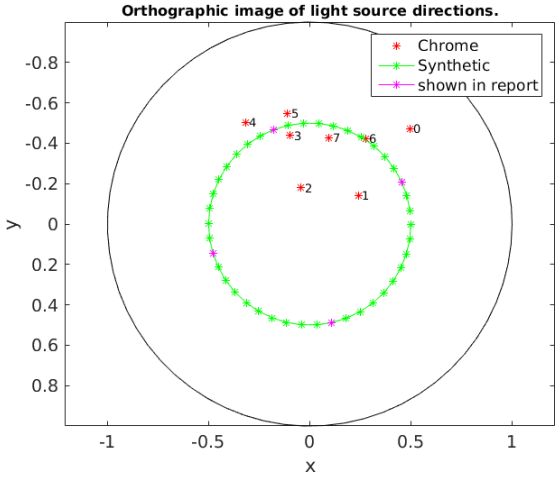


Figure 11: Synthetic light direction

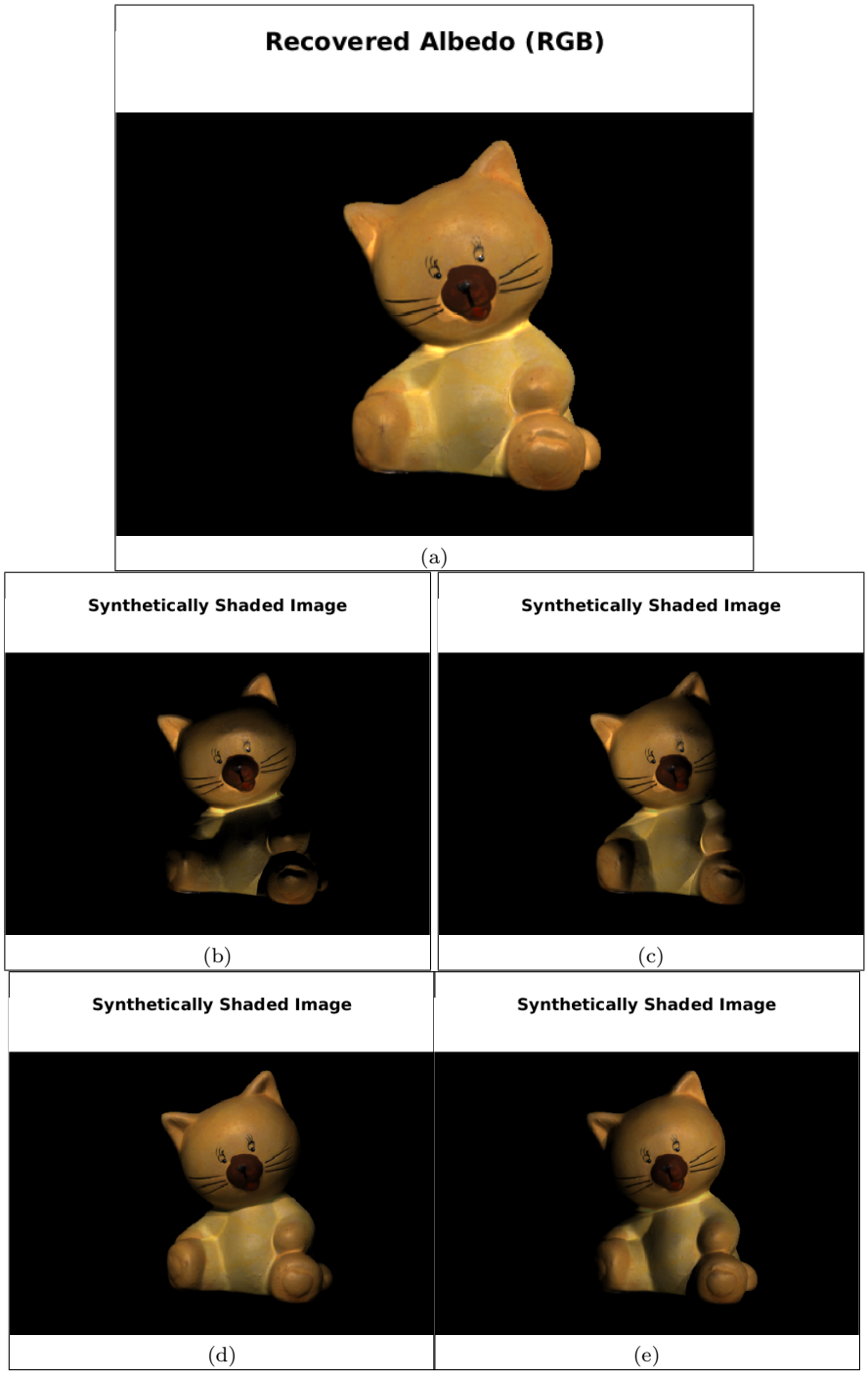


Figure 12: (a) Results of RGS albedo on cat image. (b) synthetic light from below (c) synthetic light from left (d) synthetic light from above (e) synthetic light from right

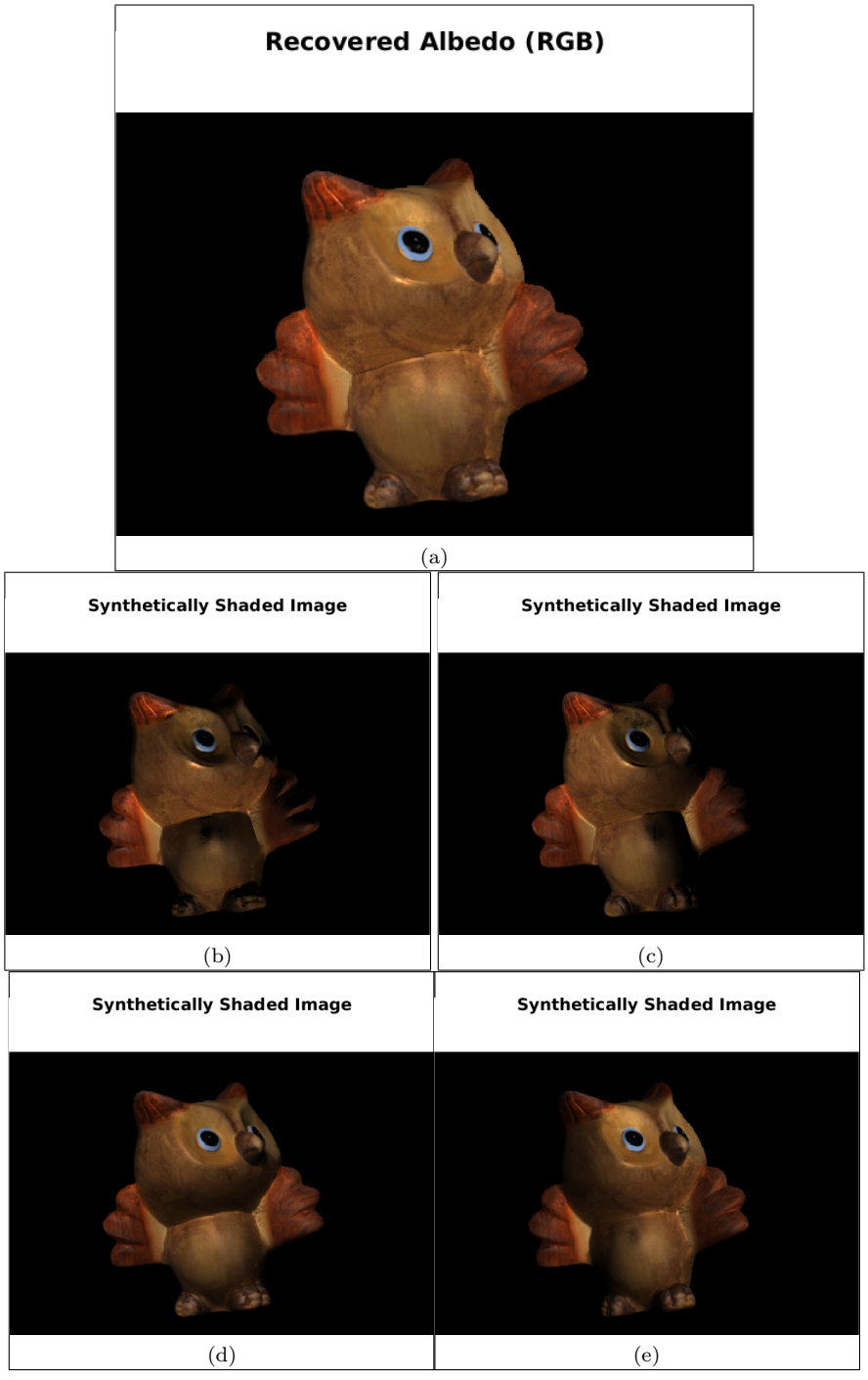


Figure 13: (a) Results of RGS albedo on owl image. (b) synthetic light from below (c) synthetic light from left (d) synthetic light from above (e) synthetic light from right

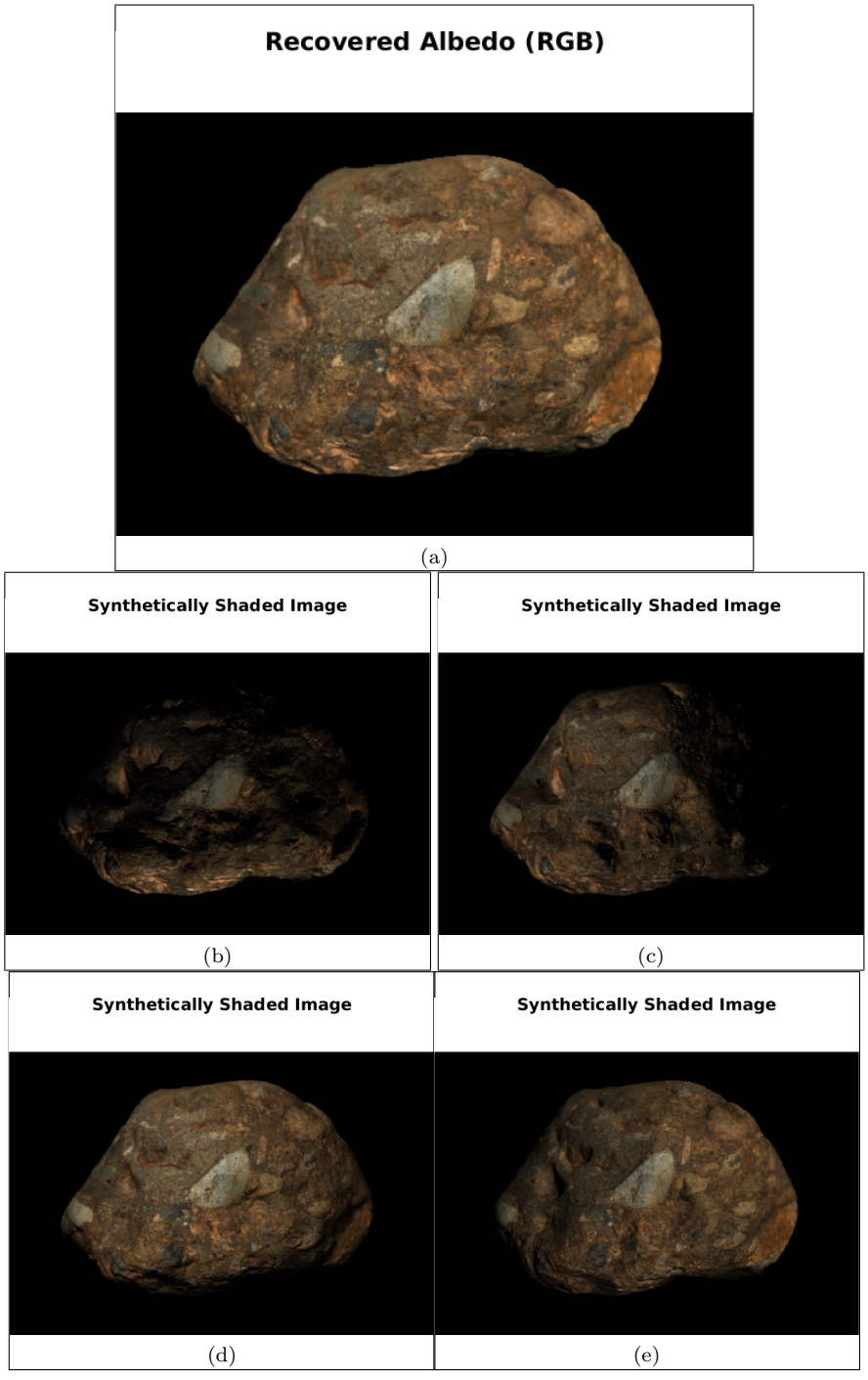


Figure 14: (a) Results of RGS albedo on rock image. (b) synthetic light from below (c) synthetic light from left (d) synthetic light from above (e) synthetic light from right

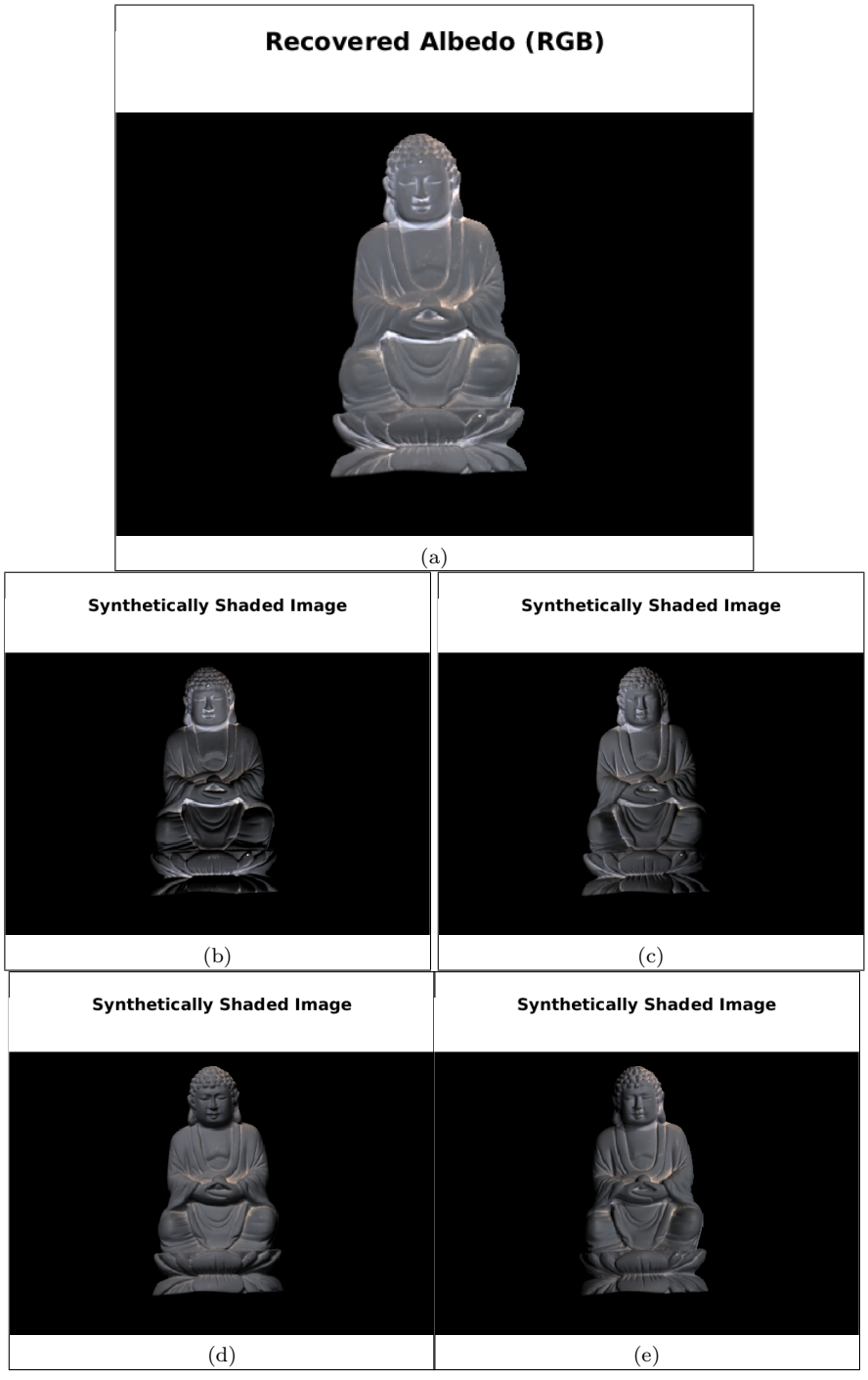


Figure 15: (a) Results of RGS albedo on buddha image. (b) synthetic light from above (b) synthetic light from below (c) synthetic light from left (d) synthetic light from above (e) synthetic light from right

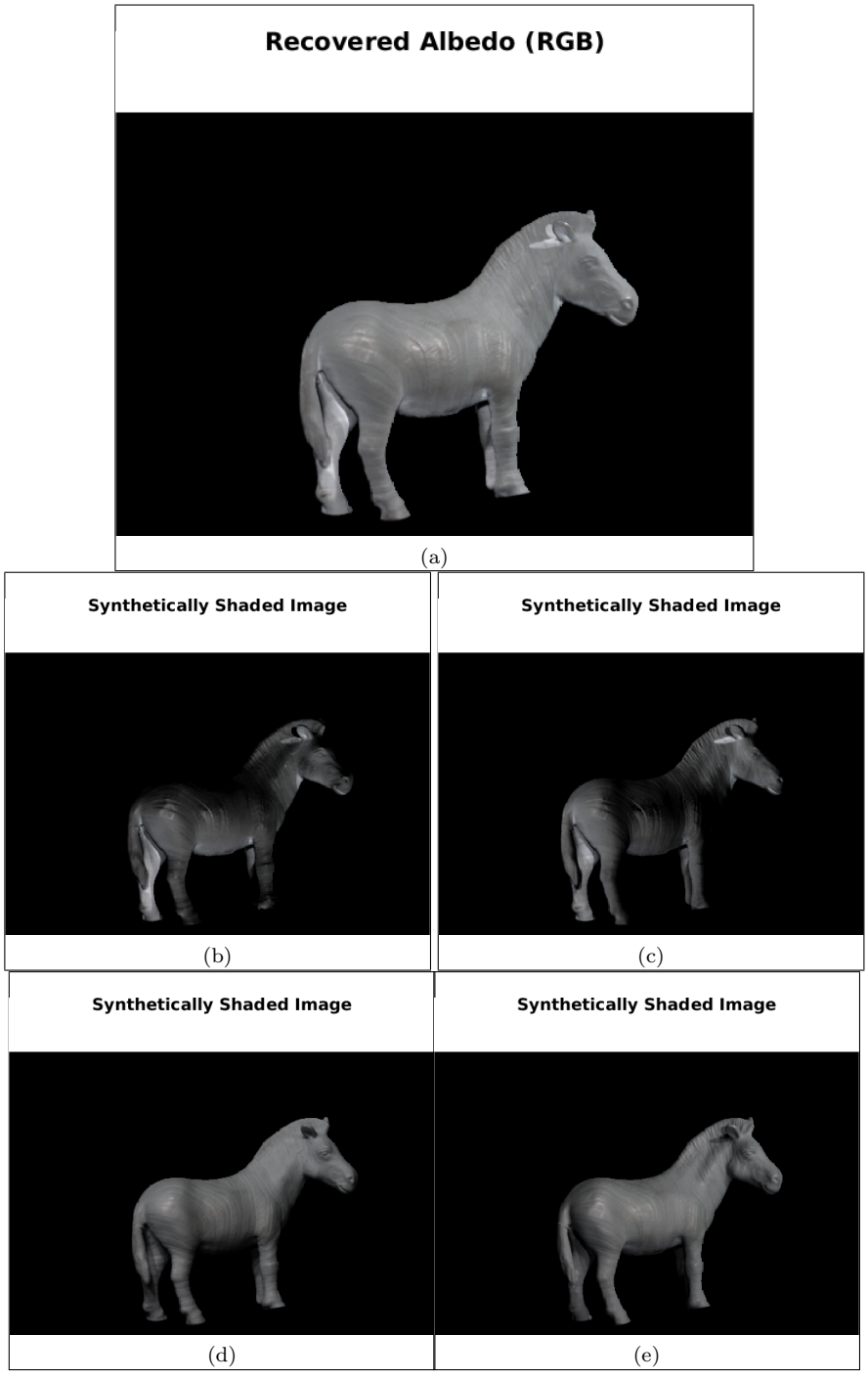


Figure 16: (a) Results of RGS albedo on horse image. (b) synthetic light from below (c) synthetic light from left (d) synthetic light from above (e) synthetic light from right

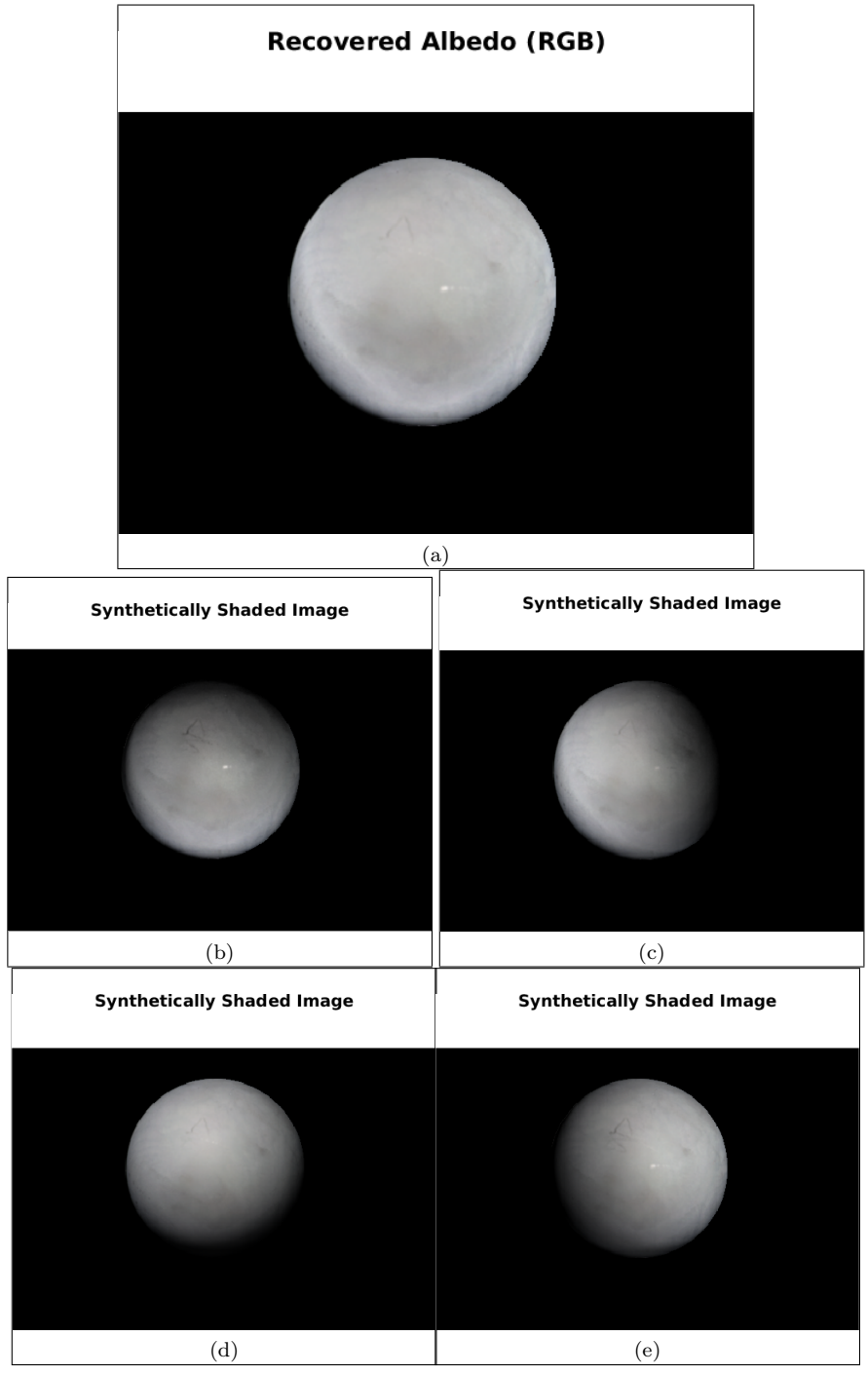


Figure 17: (a) Results of RGS albedo on gray image. (b) synthetic light from below (c) synthetic light from left (d) synthetic light from above (e) synthetic light from right

Question B-4

As formulated in the question, we need to solve following for $\vec{z}$:

$A\vec{z} = \vec{v}$ where A is $2MN \times MN$ (1)
$\vec{z}$ is $MN \times 1$
$\vec{v}$ is $2MN \times 1$

This is the matrix form of following equations:

$\vec{\tau}_x(x,y) \cdot \vec{n}(x,y) = 0 \Rightarrow n_x + n_z z(x+1,y) - n_z z(x,y) = 0$ (2)
$\vec{\tau}_y(x,y) \cdot \vec{n}(x,y) = 0 \Rightarrow n_y + n_z z(x,y+1) - n_z z(x,y) = 0$ (3)

So $\vec{z}$ is flattened vector of all x & y and is the vector we want to solve the equation for.
In order to do so, we need to convert the x & y indices to linear indices. This can be done using MATLAB's sub2ind function:

$id_\ell = \text{sub2ind}(n,m)$ where n & m are indecis of x & y pixels
& $id_\ell^x = \text{sub2ind}(n+1,m)$ & $id_\ell^y = \text{sub2ind}(n,m+1)$
According to (2):

(4)
$$\begin{cases} A(id_\ell, id_\ell) = -n_z(n,m) \\ A(id_\ell, id_\ell^x) = +n_z(n,m) \\ \vec{v}(id_\ell, 1) = n_z(n,m) \end{cases}$$

According to (3):

(5)
$$\begin{cases} A(id_\ell + MN, id_\ell) = -n_z(n,m) \\ A(id_\ell + MN, id_\ell^y) = n_z(n,m) \\ \vec{v}(id_\ell + MN, 1) = n_y(n,m) \end{cases}$$

Note that the constrains in (5) are added below the constrains in (4) and hence there is an offset of MN in row.
After forming A and $\vec{v}$, we solve (1) using $\vec{z} = \vec{v}/A$.
The matrix A should be sparse because only few of elements correspond to mask of object and the rest are zero.

7 Part B-4

For implementing this part, I define matrix A as a sparse matrix in MATLAB due to otherwise memory errors as well as its sparse nature. I first made matrices A and v for areas determined in mask matrix and then solve for z using z=A v. Finally I made the elements of the depth matrix zero for the areas that are not part of the object using mask matrix. By making useless elements in depth matrix zero, we can show the z component with z=0 origin that corresponds to minimum depth of the object.

The results are shown in Figure 18. Overal, the shape of the objects is captured very well using this algorithm. In Figure 19, I showed the depth estimation results where the performance is very good (panel (a)) along with some flaws I found (panels (b)–(d)). One particular type of flaws is shown in Figure 19 panels (d) and (e) where there seems to be an error, especially along the borders of the sphere due to inaccurate surface normals as also discussed in Part B-2. This object looks spherically symmetric, however the depth component of it shows some deviation from a sphere which can be explained by erroneous surface derivation due to shade and more importantly due to simplifying Lambertian reflectance. Perhaps a more sophisticated model for reflectance could improve the results for this type of error. Moreover, there are some high error in the some parts of the object as shown in Figure 19 panels (b), where the eye of the owl has dramatically more depth for only few pixels. This cannot be true as the surface must preserve the smoothness. This is also reflected in z component of surface normal in Figure 10(b). This can be easily fixed by using a smoothing kernel, or RANSAC algorithm for obtaining outliers and removing them (and replacing the surface estimate with the mean of depth from neighboring pixels).

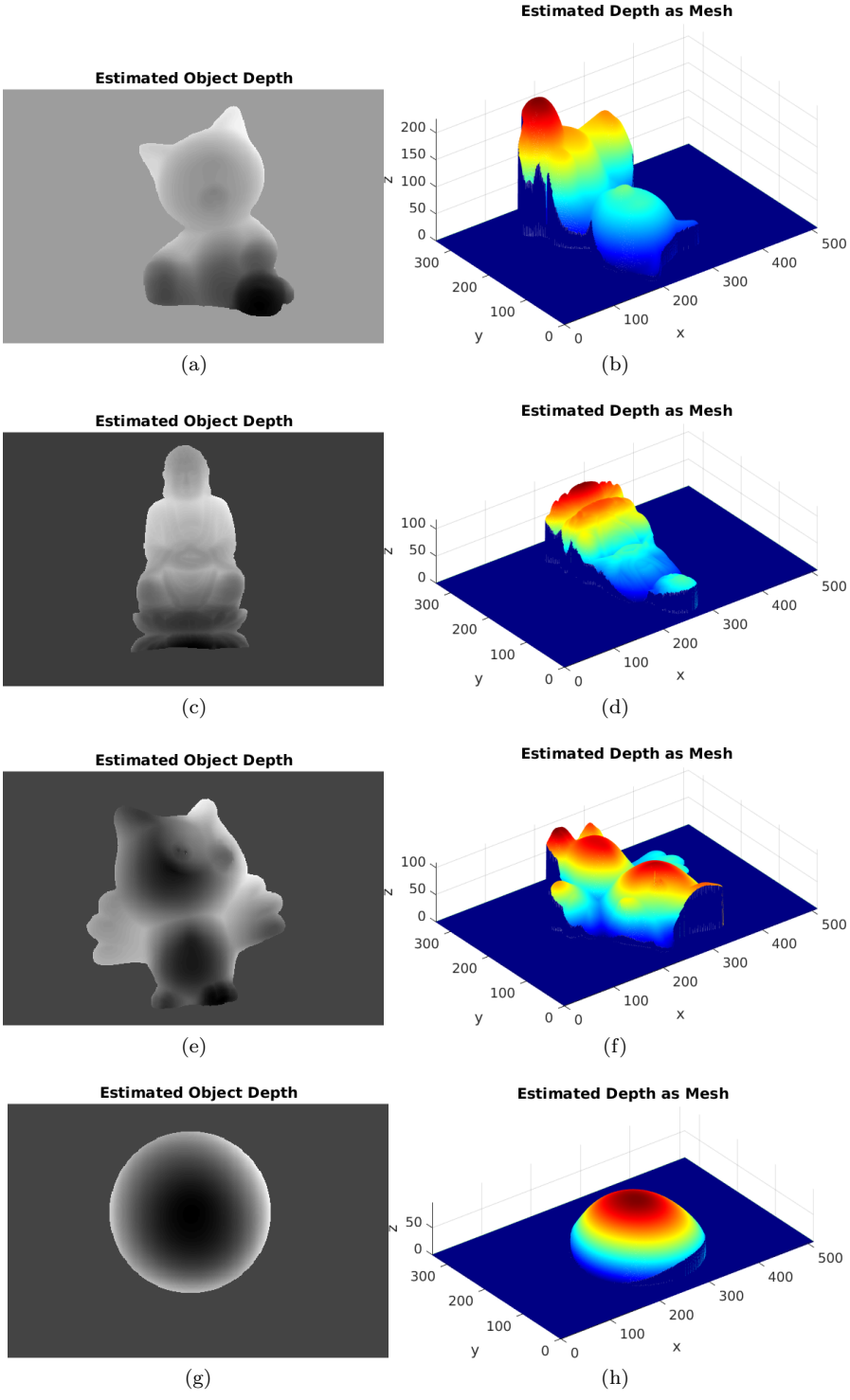
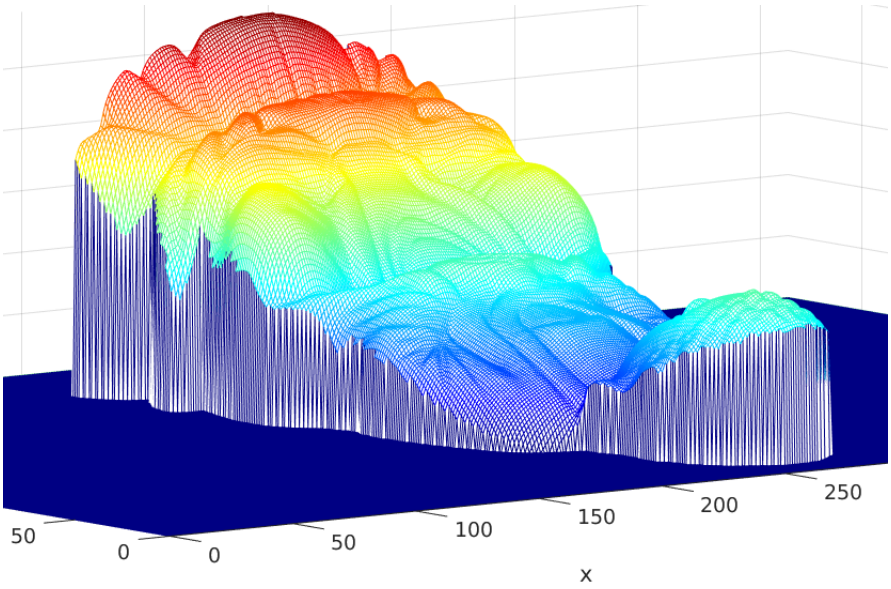
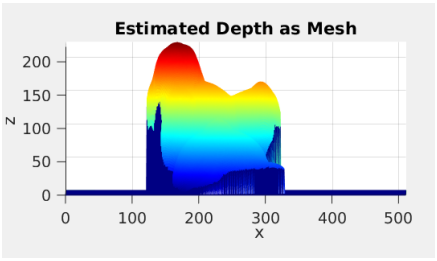


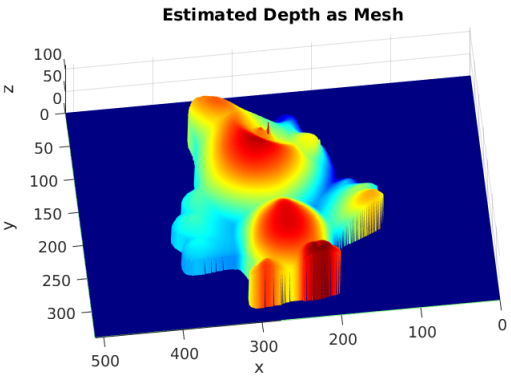
Figure 18: Depth Estimation results



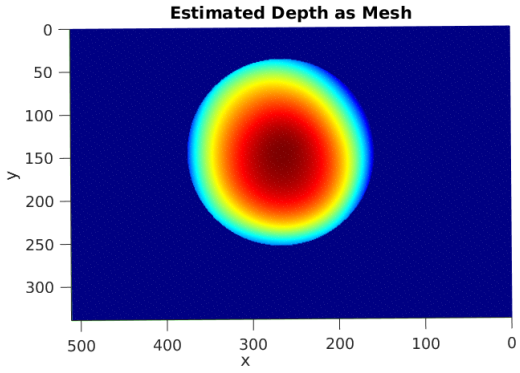
(a)



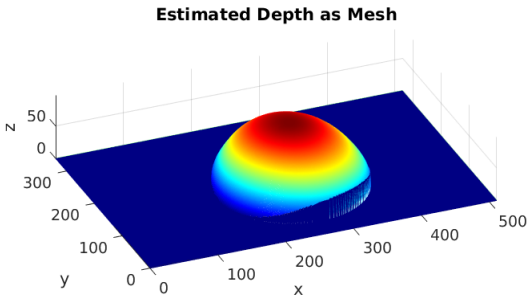
(b)



(c)



(d)



(e)

Figure 19: The performance of depth estimation (a) is one example of decent performance (b)-(e) some flaws in depth estimation algorithm

Question B-5

Similar to previous questions:

$$e_{xy,v} = (I_{xy,v} - a_{xy,v}\, n_{xy} L)$$

where I_{xy} is 1x4 & L is 3x4, n_{xy} is 1x3, $a_{xy,v}$ is scalar

$$O = \sum_{v}\sum_{x}\sum_{y} e_{xy,v}\, e_{xy,v}^{T} = \sum_{x,y} I_{xy} I_{xy}^{T} - a_{xy,v}^{2}\, n_{xy} L L^{T} n_{xy}^{T} - 2a_{xy,v}\, n_{xy} L I_{xy}^{T}$$

$$\frac{\partial O}{\partial L} = 0 \Rightarrow 2a_{xy,v}^{2}\, n_{xy}^{T} n_{xy} L - 2a_{xy,v}\, n_{xy}^{T} I_{xy,v} = 0$$

$$\Rightarrow \boxed{L = \frac{1}{a_{xy,v}} (n_{xy}^{T} n_{xy})^{-1} n_{xy}^{T} I_{xy,v}} \quad (*)$$

for each image with missing L information we need 3 constraints to find (L_x, L_y, L_z). Therefore $(*)$ needs to be solved for at least 3 pairs of (x, y).

8 Part B-5

I only assumed one light source in additional images. Therefore my alogorithm is simple:

```
Lrec = inv(g'*g)*g'*imData(mask,(nDirChrome+1):nDir);
```

The results are shown in Figure 20–24. The reconstructed image resembles the original image, but the error is relatively high, perhaps there are more than one light source in the images.

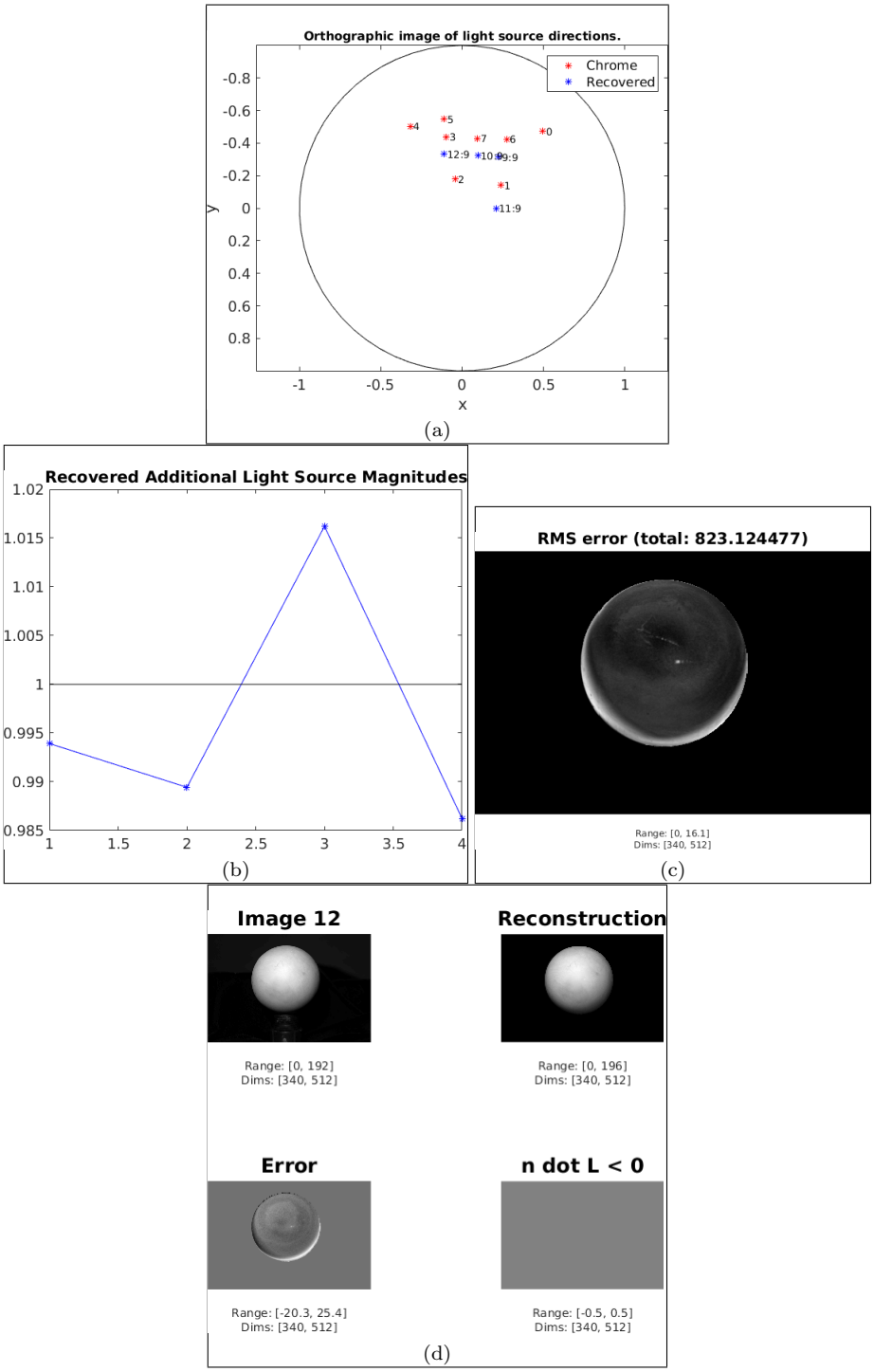


Figure 20: Results of part B-5

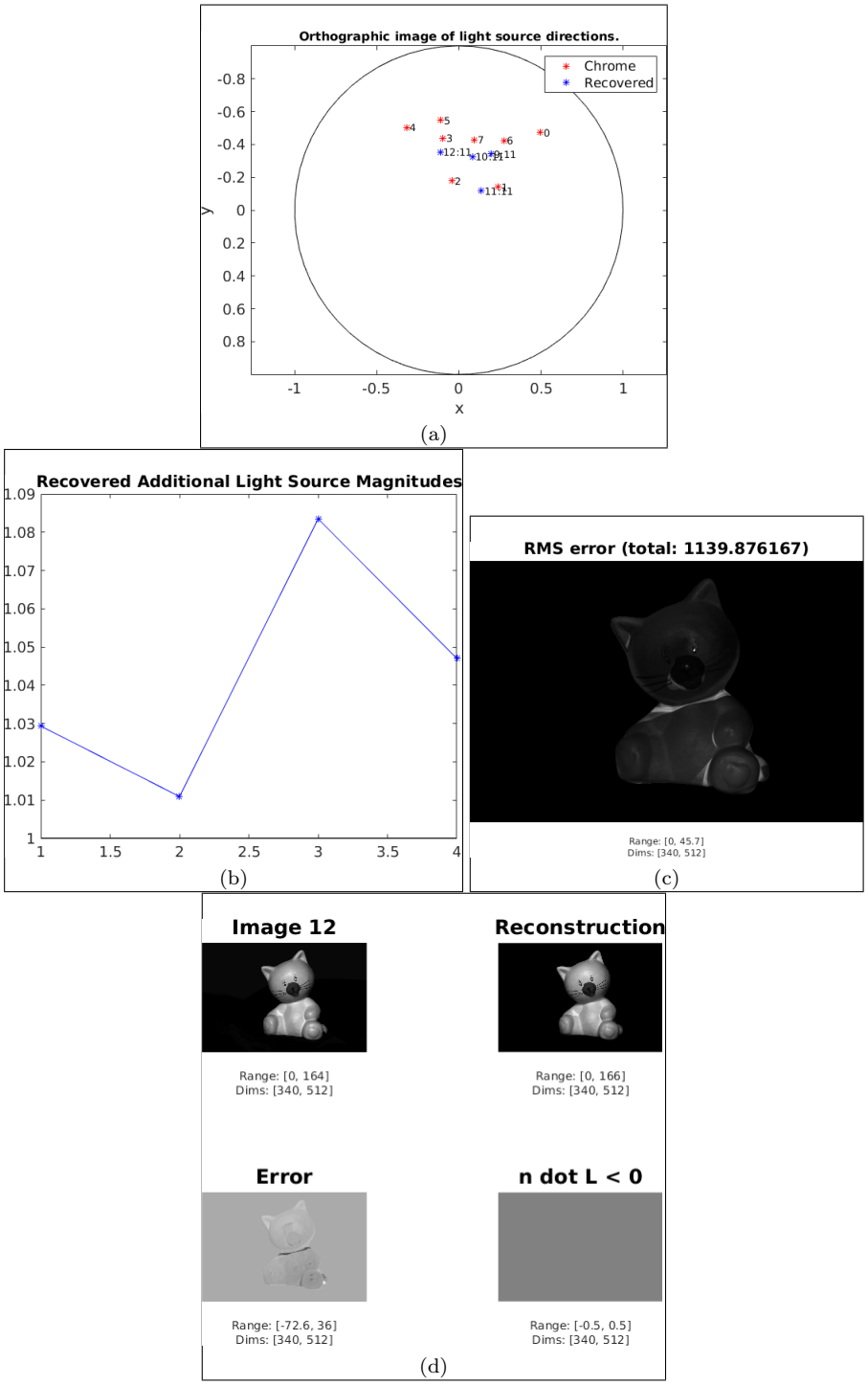


Figure 21: Results of part B-5

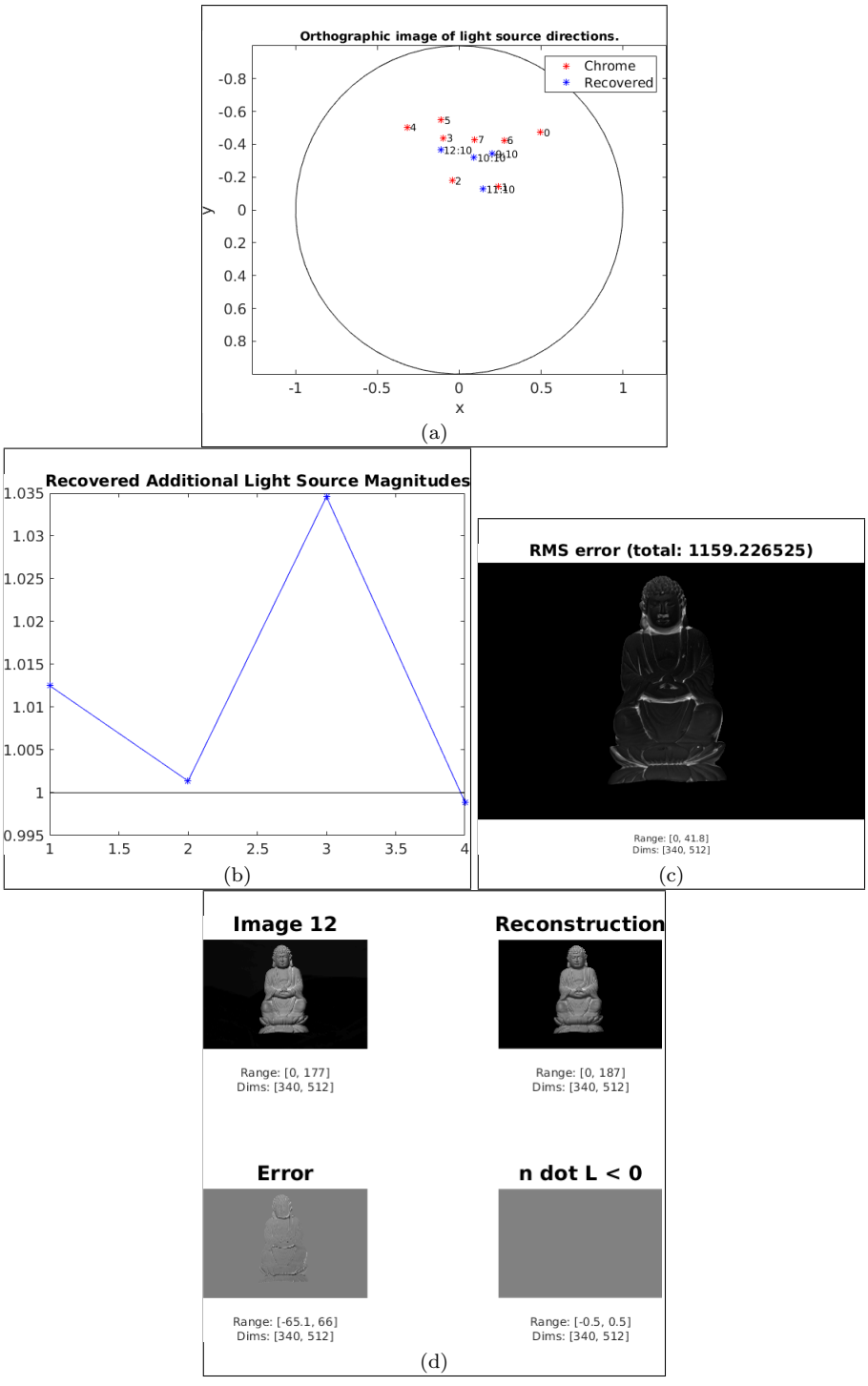


Figure 22: Results of part B-5

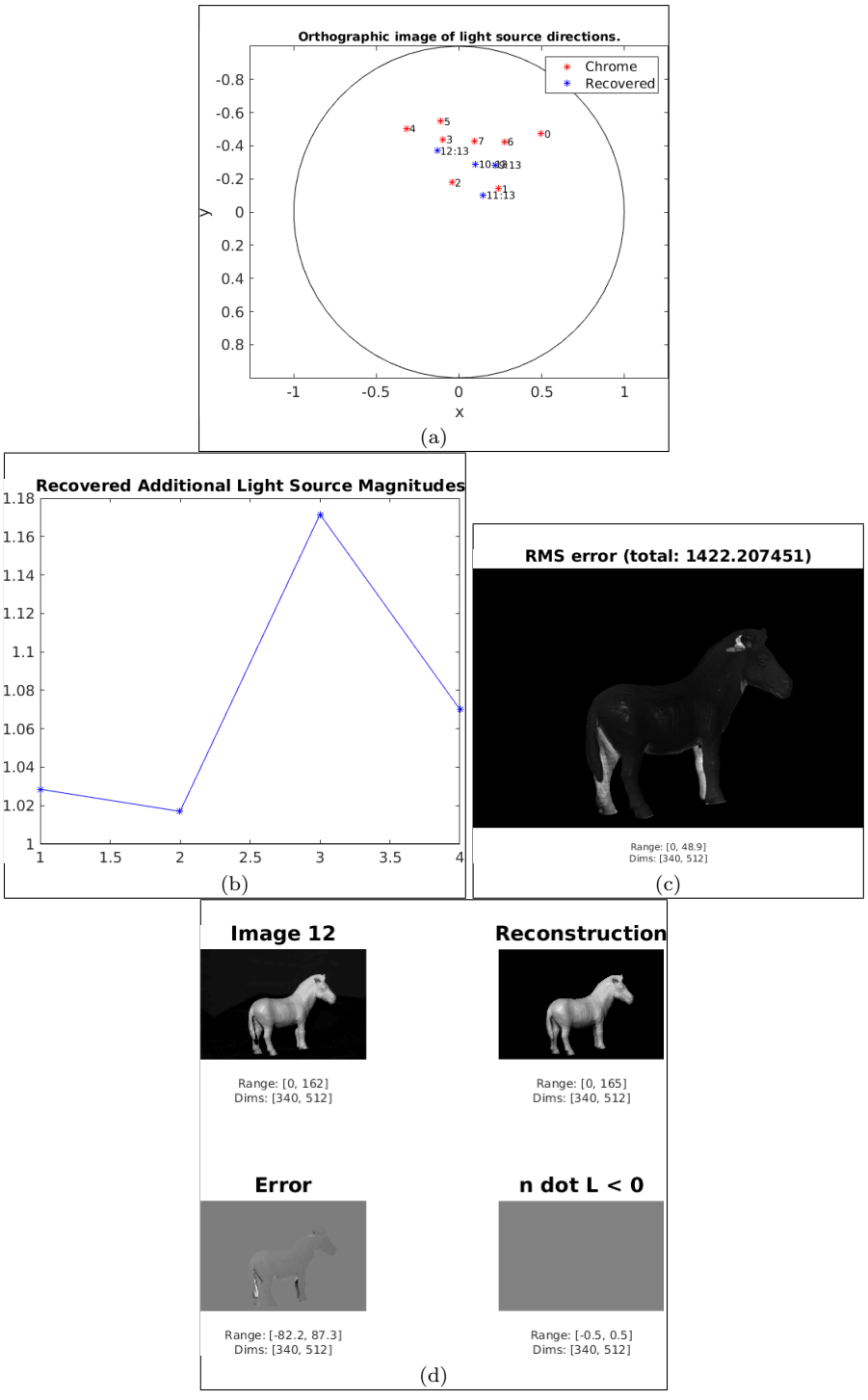


Figure 23: Results of part B-5

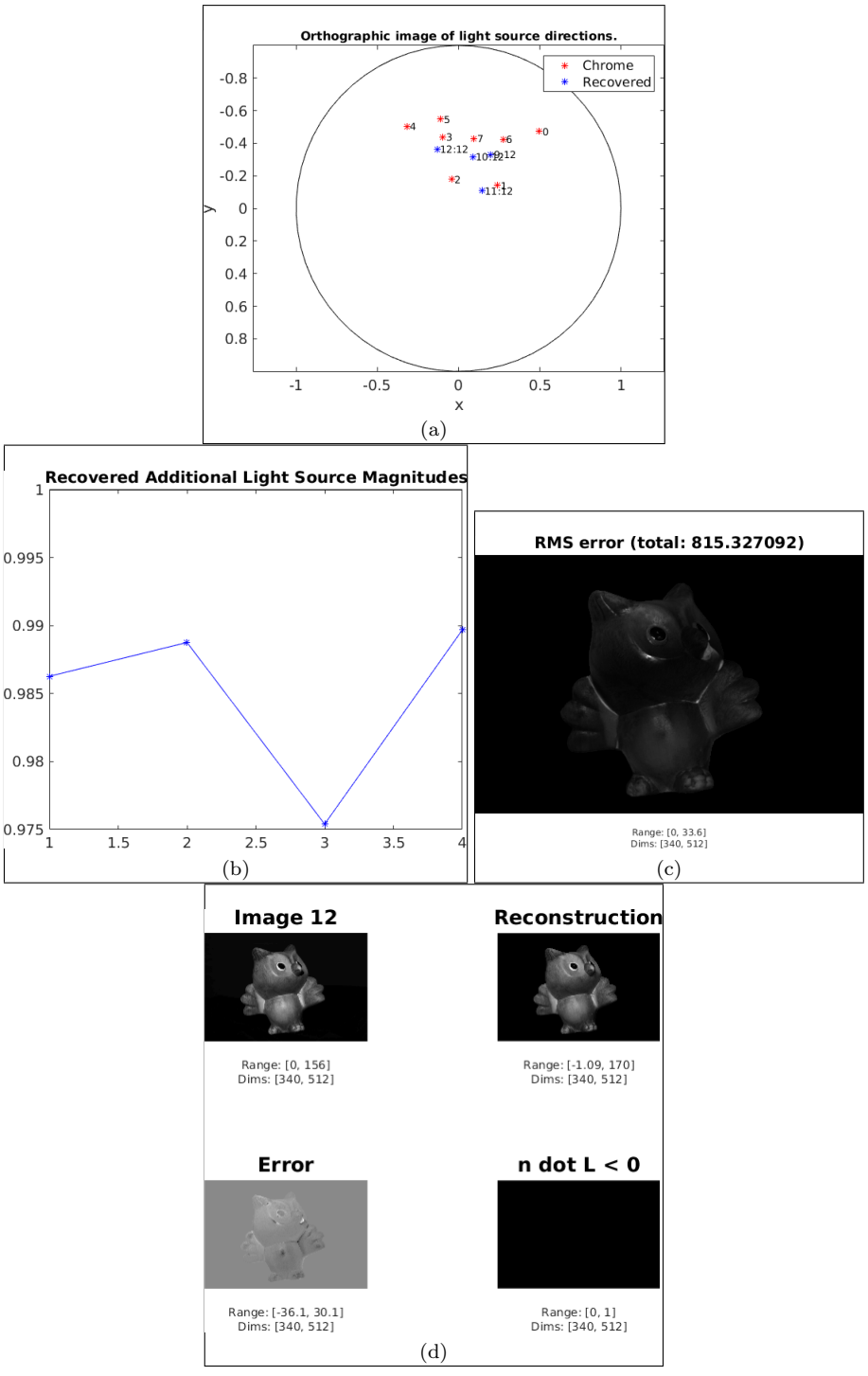


Figure 24: Results of part B-5