CSC 2503
Joel Cheverie
1002924393

Assignment 1: Photometric Stereo

Q. 1: Calibration

To calibrate the system, I am going to use the direct specular component of the Phong model. As this is a chrome sphere, we can assume it will act similarly to a perfect mirror, so we can ignore the other components of the Phong model during calibration. To actually estimate the light source directions, I do the following:

A) Determine the centre and the radius the Chrome Sphere.

The mask is assumed to be a near-perfect sphere. Hence the centre is simply the mean of the row and column lengths. We can find the pixels that represent the sphere in the chrome.mask.ppm file by determining which pixels have values that are non-zero (the white parts).

These indexed pixels are found through the command:
find(mask>0);

This gives us our indexed dimensions of the sphere (ie. Total row count and total column count of the sphere). From here, simply take the mean of each.
i.e. If the indexed sphere dimensions are X and Y,
centre = mean(X), mean(Y);

Then the diameter is simply:
diameter = double(max(X) – min(X));

So the radius is:
radius = double(diameter /2);

B) Determine the brightest spot in the chrome sphere images.

This is done similarly to how the centre and radius of the chrome sphere are found. For each small spot mask, I find all of the pixels in the chrome spot file where the pixel values are 255 (white spots). This is done through:
[bright_x, bright_y] = find(imData(:,:,i) == 255);

Then take the mean of each indexed dimension to get the brightest pixel (the centre of the bright spot):
reflection_point = [mean(bright_x), mean(bright_y)];

C) Determine the surface normal for each bright spot on the chrome sphere.

To determine the normal, the following are computed:

norm_x = reflection_point(1) – centre(1);
norm_y = reflection_point(2) – centre(2);
norm_z = -sqrt(radius^2 -norm_x^2 – norm_y^2);

So then the unnormalized normal is:

unnormalized_normal = [norm_x, norm_y, norm_z];

And finally the normalized normal is:

normal = unnormalized_normal/norm(unnormalized_normal);

Note: Here I am subtracting the centre from the reflection points as I am taking the centre of the sphere as the origin. Also, the norm_z component is negative due to the direction of the camera being (0,0,-1).

D) Determine L (the Light Source Directions)

Finally, as I am assuming perfect specular reflection, I can recover d_i (the incident light direction) from d_e (the emittant light direction) and the normalized surface normal at the bright spot as follows:

*d_i = 2 * dot (normal, d_e) * normal – d_e; (the perfect specular reflection equation)*

Where d_e = [0,0, -1] due to where the camera is positioned.

So as d_i is already normalized, for each spot, we now have its light source direction L(:,i) :
L(:, i) = d_i;
We can then loop through the full set of 8 spots to recover L.

These are my results:

L' (transpose for display purposes) =

0.4960	-0.4642	-0.7338
0.2424	-0.1360	-0.9606
-0.0378	-0.1750	-0.9838
-0.0939	-0.4412	-0.8925
-0.3178	-0.5074	-0.8010
-0.1101	-0.5605	-0.8208
0.2820	-0.4226	-0.8613
0.1015	-0.4304	-0.8969

The MSE of this value with the default Light Dir matrix is: 0.0144. Thus it is actually fairly different from the provided default values (as expected).

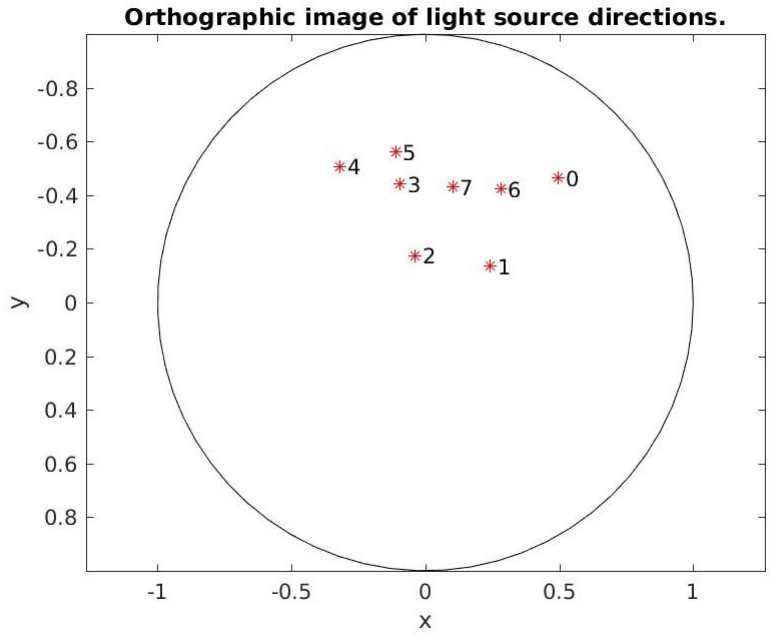


Figure 1: Recovered L

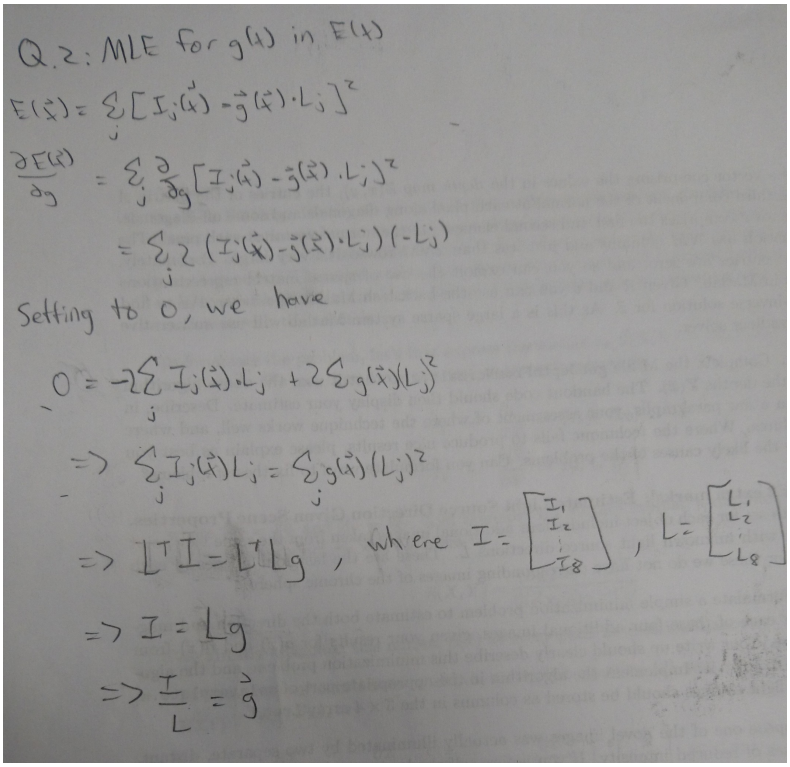
Q.2. Surface Normals and Grey Albedo

So as per the definition of g in the assignment, we have:

$a(x) = ||g(x)||$

$n(x) = g(x) / ||g(x)||$

where a(x) is the grey-level abledo and n(x) is the surface normal. It is then straightforward to solve for a(x) and n(x) by estimating g(x) trhough by differentiating E(x) w.r.t. g, and setting the derivative to 0 (computing the MLE of g):



So, I simply need to compute:

g = L’\ im’;

See below for the recovered grey albedo, surface normal, image reconstruction and rms error of the Cat and Buddha objects. I chose these objects as they were the easiest to visualize in grayscale.

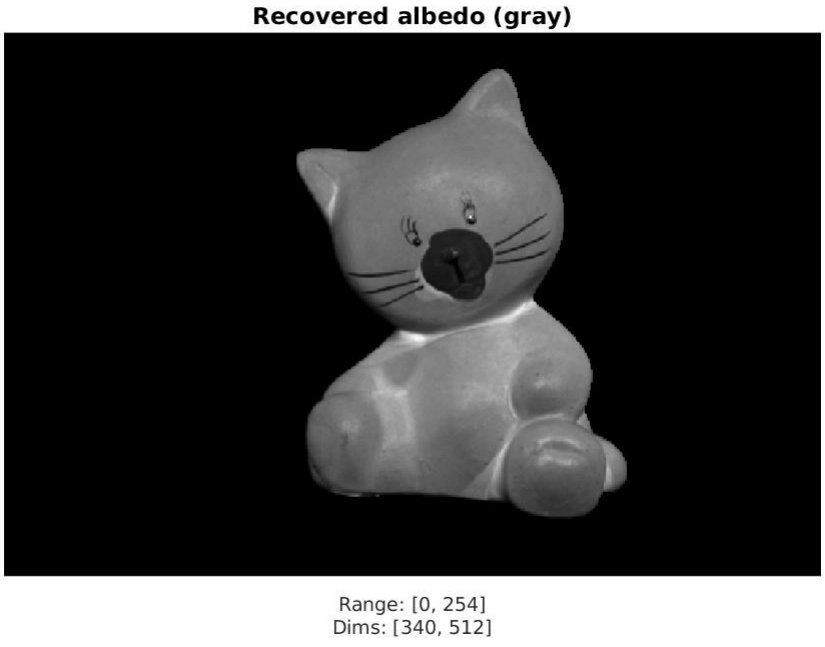


Figure 2: Recoved grey albedo on the Cat Object

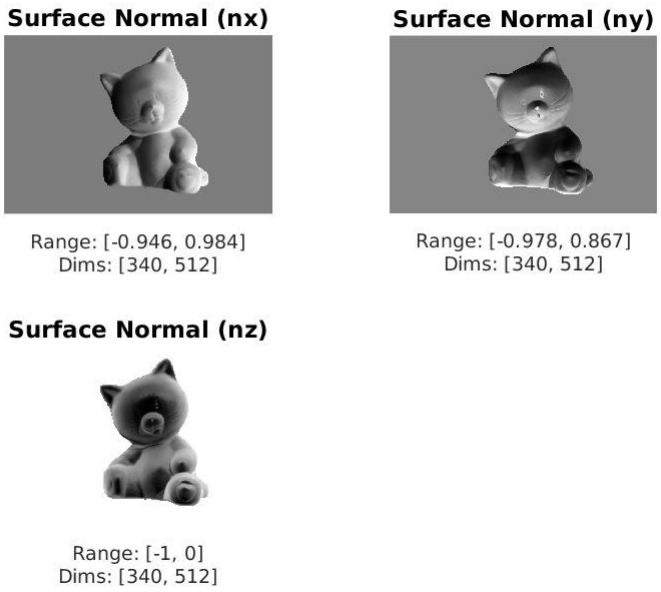


Figure 3: Estimated surface normals for the Cat Object

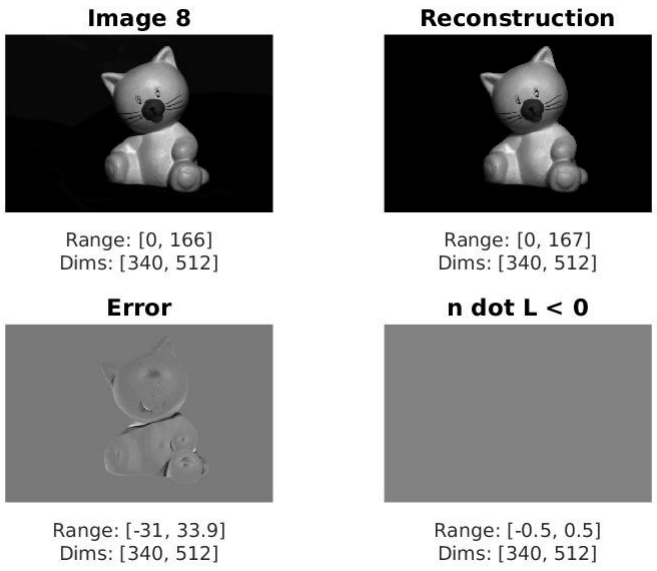


Figure 4: Detailed Reconstruction Error for the 8th image of the Cat Object

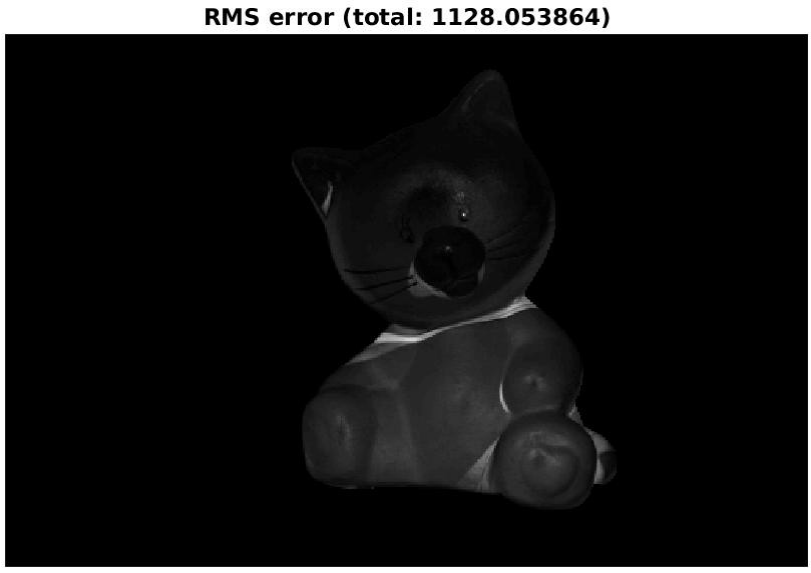


Figure 5: Total RMS Error for the Cat Object

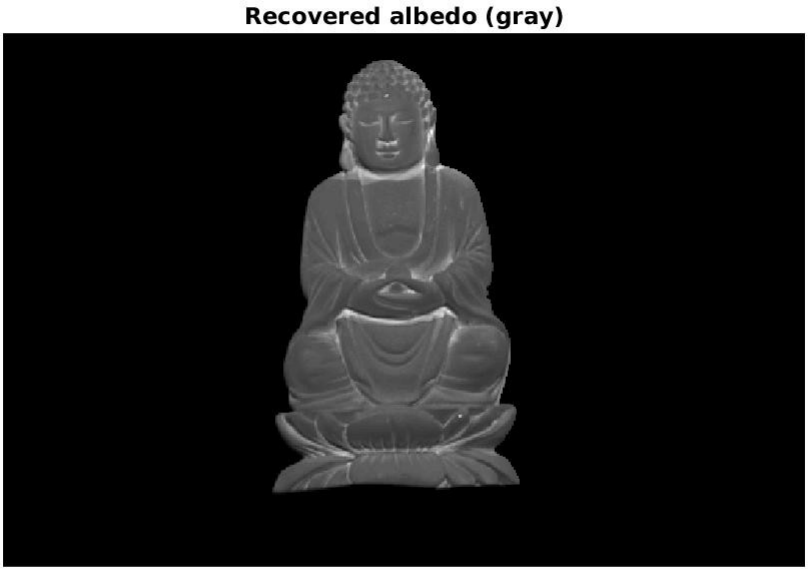


Figure 6: Recoved grey albedo on the Buddha Object

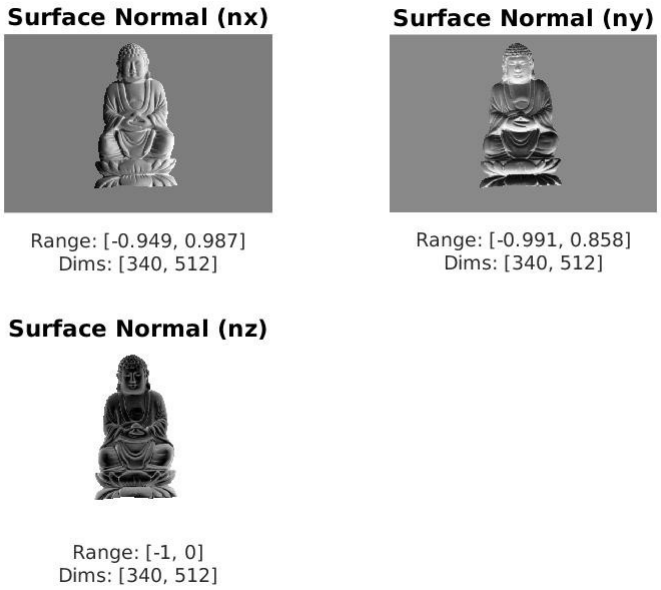


Figure 7: Estimated surface normals for the Buddha Object

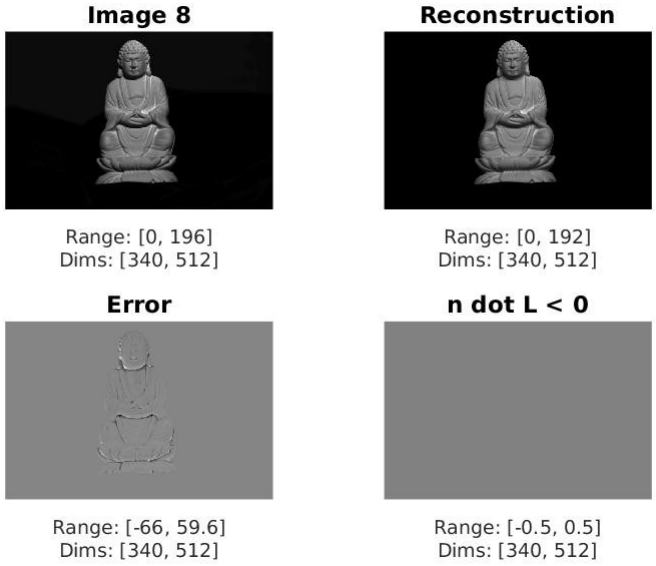
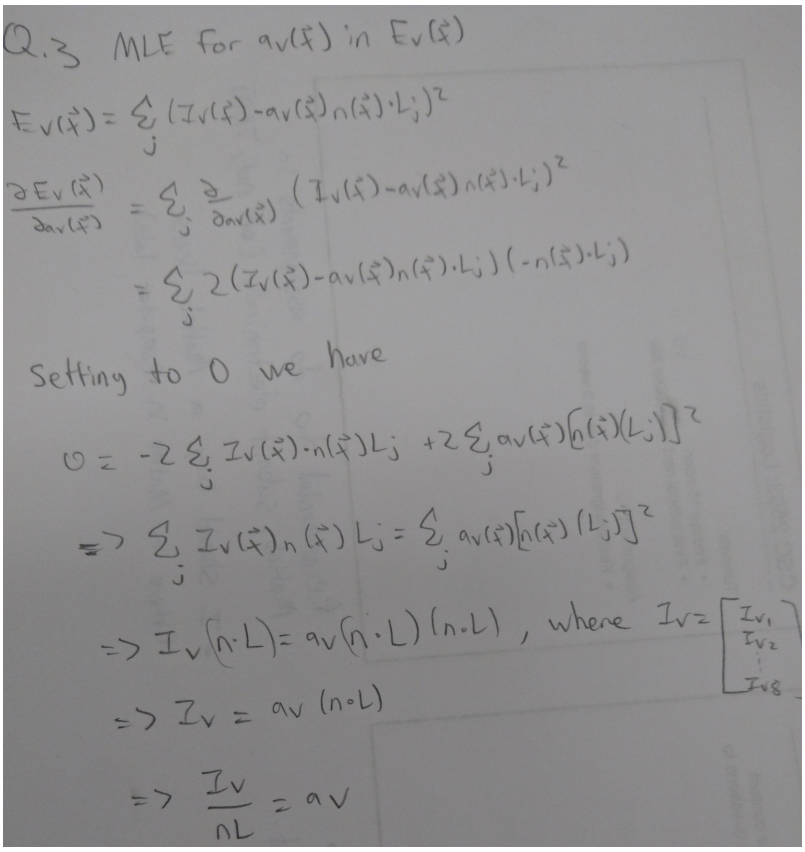


Figure 8: Detailed Reconstruction Error for the 8th image of the Buddha Object

Figure 9: Total RMS Error for the Buddha Object

The quality of the recovered grey images are actually fairly decent. It is obviously not perfect however. For instance the neck areas of both of the estimated reconstructed images have a higher value on the RMS error figures. Similarly the left leg and right ear of the cat object have higher RMS error values, as do the shadowy creases on Buddha's clothing. These areas seem to correspond to where there were shadows in the original images. I'm presuming these concentred error areas are due to the assumptions made in our model about a Lambertian surface. In particular, this model is focusing on the diffuse reflection from a given surface, hence there is error where the light is directly reflected. More specifically, in this type of model, shadows are not accounted for as light is assumed to reflect off of a surface directly into the camera. Hence the shadowy areas have higher estimated albedo than they do in reality.

Q. 3 RGB Albedo and Relighting



Thus, we have the same result as in the previous question.

So as computed, we have:
a_v(x) = I_v / (nL) , or g = I_v/L

The idea here is similar to what was done in the previous question. We are now simply calculating albedo in the different colour channels. Consequently I am able to resue the code I implemented for the second part. More specifically:

I first read the image file into three separate colour channels:

```
rgb_data(:,:,i) = reshape(rgb(:), [numPixels, 3]);
```

Then I crop it as was done in part 2 to work within the mask area:

```
rgb_crop = rgb_data(mask,:, 1:nDirChrome);
```

Finally I am able to get the different albedo estimations for each colour channel by looping through the individual colour channels with the fitReflectance function. I am not computing the the surface normals again however as they are the same as in the grey case. Hence it is simply:

for i = 1:3
[~, albedo(mask,i)] = fitReflectance(squeeze(rgb_crop(:,i,:)), Lchrome);
end
(where i indexes the colour channels in the cropped rgb data.)

See below for the estimated colour albedo on the cat and horse objects, a synthetically shaded image of the cat and horse objects and the orthographic image of the light source directions. I chose these objects as they give meaningful observations for rgb albedo estimation errors.

Figure 10: Recovered Albedo for 3 Colour Channels of Cat Object

Figure 11: Synthenically Shaded Image of the Cat Object

Figure 12: Recovered Albedo for 3 Colour Channels of Horse Object

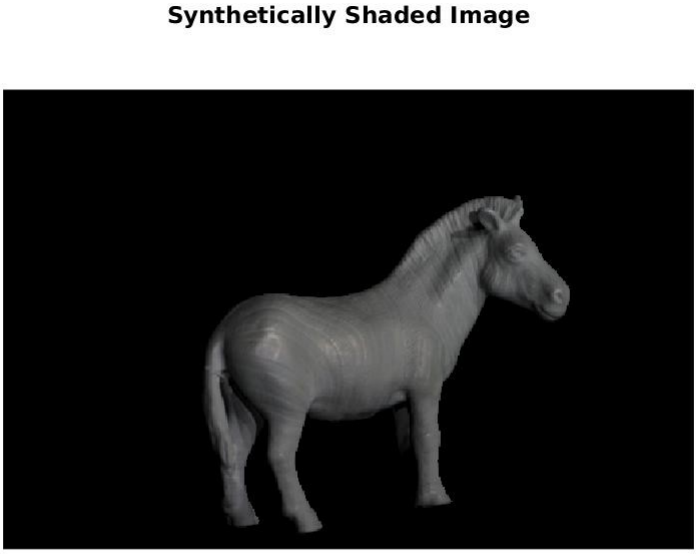


Figure 13: Synthenically Shaded Image of the Horse Object

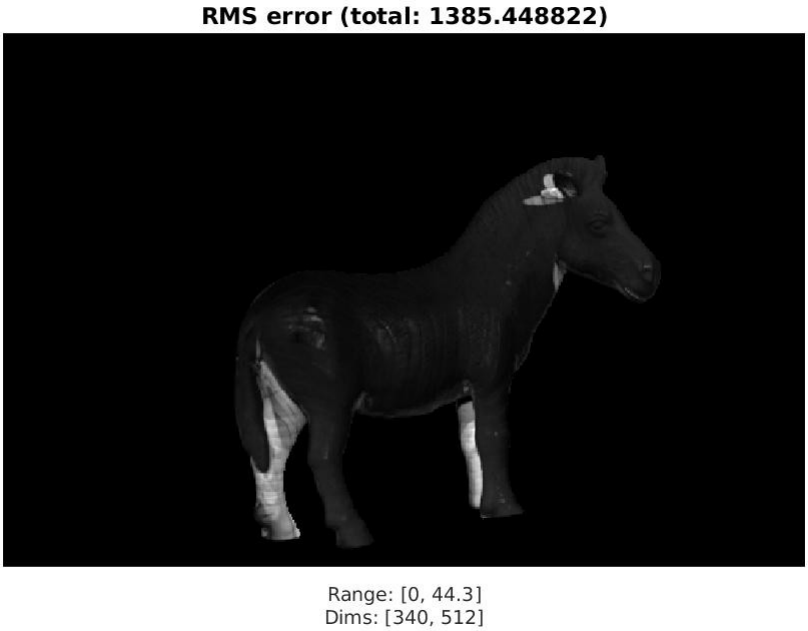


Figure 14: Total RMS Error Computed in Q2 for Horse Object

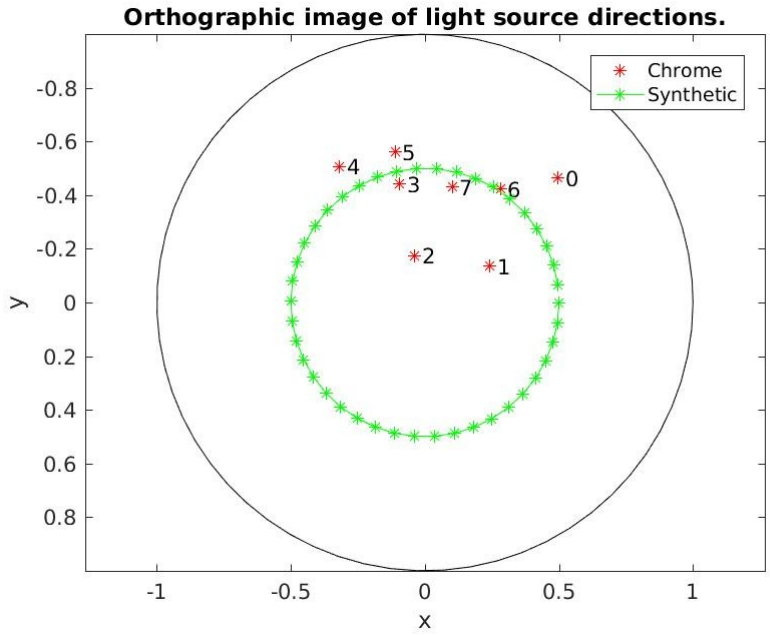


Figure 15: Orthographic Images of Light Source Directions

Here again, the neck of the cat appears bright, when it should be darker from shadows cast by its head and face. Likewise the back left leg and right ear of the horse are much brighter than they should be. These areas along with the left foot and right ear of the cat were where there was higher RMS error, hence the errors are coming from the actual albedo calculation, just like in the greyscale case. The Lambertian assumption is the underlying cause of this again as it simplifies the model. We are not assuming non-linear components (which contribute to shadows) in the model, as it is based simply on a linear diffuse reflection component. It therefore doesn't look entirely realistic in these shadowy areas, but approximates decently.

Q.4 Surface Fitting

To start, we have to reconstruct A and v, from the constraints in the assignment handout.

A: A is a large 2NM x NM matrix. It is sparse in that most of the entries are 0. Those that aren't 0 are either positive or negative z components of the normal.

V: This is going to be a vector that is 2MN x 1, which are the x and y components of the normal for each pixel.

As outlined in the assignment, we then have to solve Az = v. As we are concerned with z, we are able to use:

z = A\v;

This finds the pseudo-inverse solution for z. It is ideal here as it uses built-in optimizations to solve for the sparse matrix (which is massive here).

A higher level overview of my algorithm here is as follows:

1. Loop through the indexed pixels within the masked region of the image
2. Index the current row and column of the 3 different N x M normal components to construct A and V, as outlined in the assignment.
3. Solve z = A\v;
4. Set depth = reshape(z,M,N); to transform the depths from a vector to an M x N matrix.
5. Remove the components of depth outside the mask as they are not useful and clutter up the plot.

This technique works pretty well for all of the objects. It is pretty easy to identify each object, its outline and the depth that is being expressed in a given result (though the reconstruction). In the case of the cat object, it is possible to infer that the left foot is closer to the camera than than anything else for instance. It is also possible to deduce relative depth too, like that the nose is the closest component of the face, as expected.

For the owl object, it does not appear to work as well. Particularly, it seems to fail at reconciling that the beak of the owl is closer than the rest of the face, although due to the angle of the owl's face, this may simply be due to the Lambertian diffuse surface assumption again. There are no shadows or other real indicators to allow the beak to have a different depth than the rest of the face. It also seems to lose the lower albedo area corresponding to the owl's left ear in the depth map, making for a less accurate depth estimate here. There is also a large spike on the owl's face, which seems to correspond to another area like the left ear with very low albedo. To fix a spike like this, an averaging of neighbouring albedo values could be taken. This would involve some type of bluring technique on a specific region of the depth meshes.

Here are some results of surface fitting for the cat and owl objects:

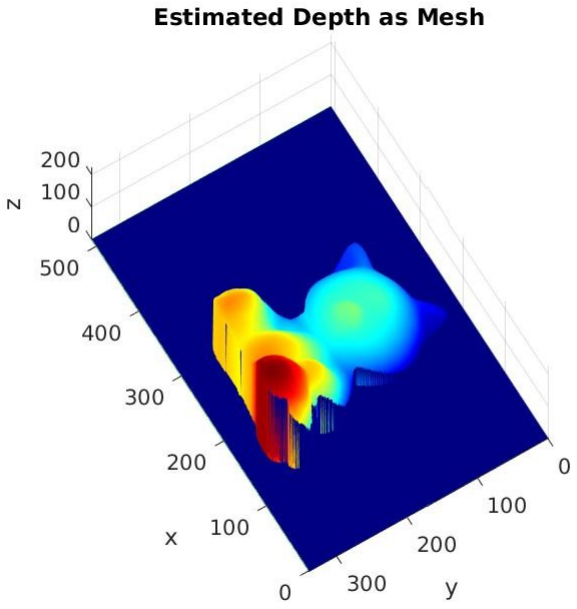


Figure 16: Estimated Depth as Mesh for Cat Object

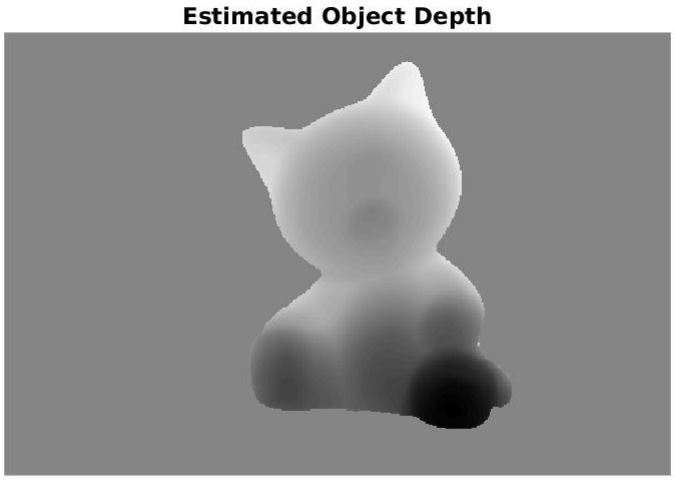


Figure 17: Estimated Object Depth for Cat Object

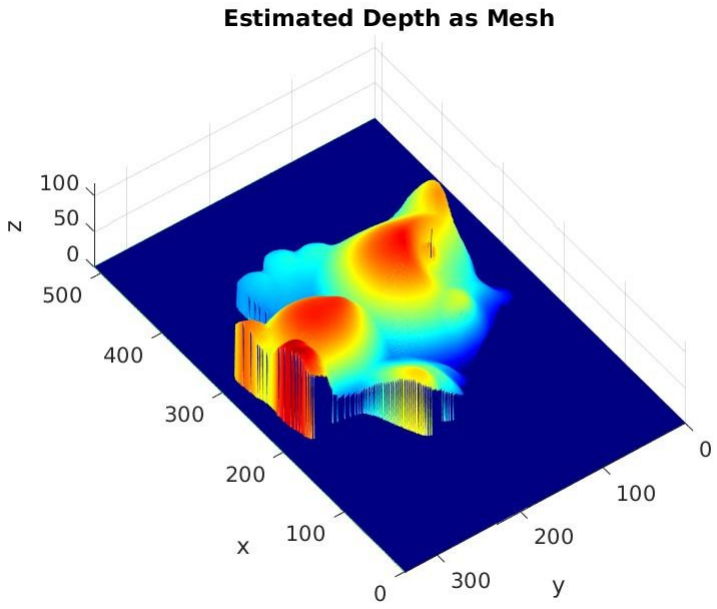


Figure 18: Estimated Depth as Mesh for Owl Object

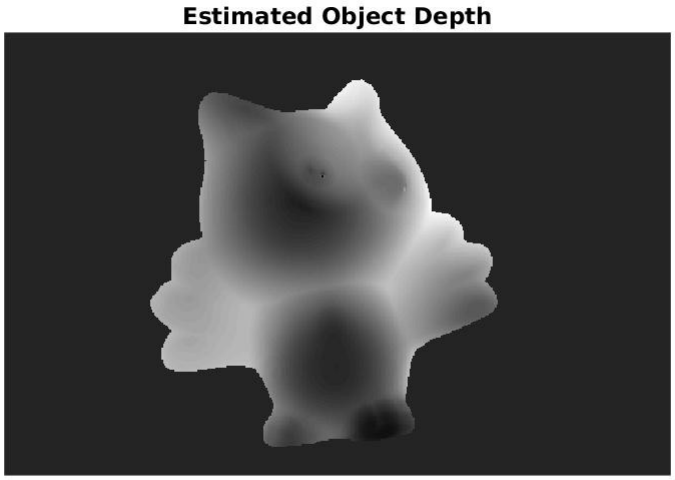
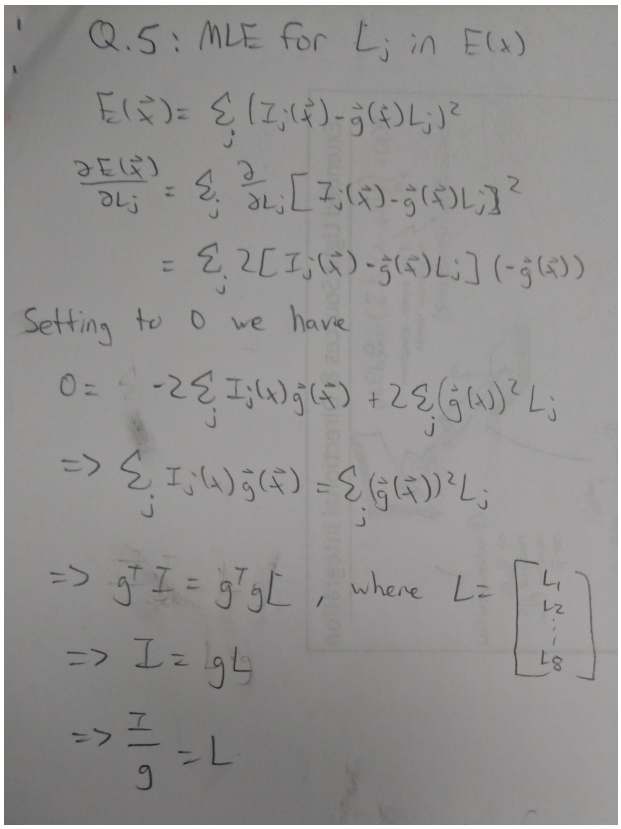


Figure 19: Estimated Object Depth for Owl Object

Q.5 (Bonus) Estimate Light Source Direction Given Scene Properties

Like in questions 2 and 3, we can use MLEs with E(x). This time, we do so with respect to L, as that is the error we are trying to minimize.



Notice here that the result is identical to section 2. It is thus easy to solve for L using the albedo constant a(x) and the normal n(x) as before:

I(x)/ g(x) = L

If there are two light source directions, then our model becomes more complicated. Specifically, we get I(x) = g(x) (L_1 + L_2). It is impossible to solve this as there is no way to break down this equation to solve for the individual uknown variables required to reconstruct both L1 and L2 individually. We simply don't have enough information here to break down total illumination values at every pixel into their different light source components. To get around that, some type of inference or approximation algorithm could be used to try to split up the light sources. This would be difficult if the light sources are in approximately the same direction however. We would need a specific breakdown of lighted pixels by light source to actually solve this problem.

Here are the results for recovered light directions and updated RMS errors for the extra cat and buddha object images:

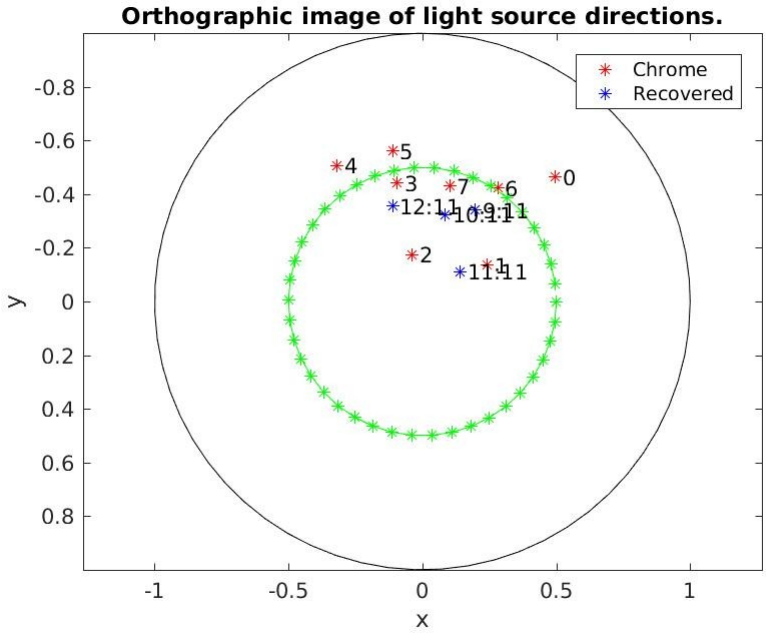


Figure 20: Orthographic Image of Recovered Cat Object Light Source Directions

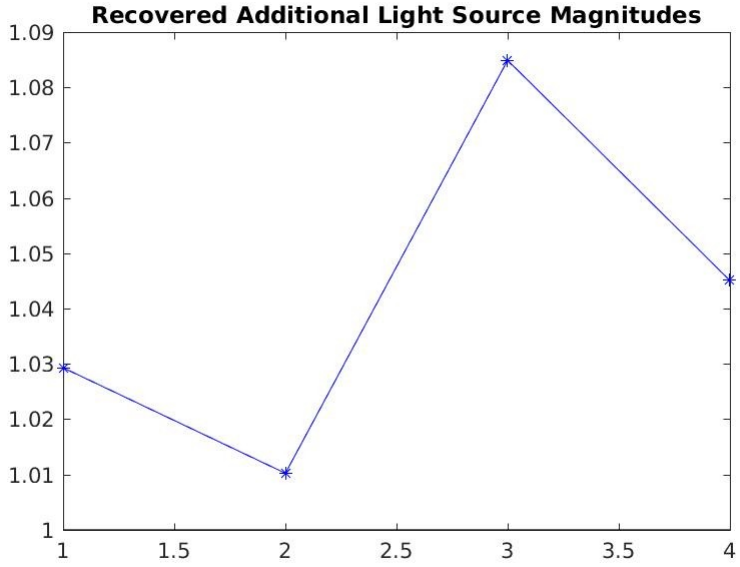


Figure 21: Recovered Additional Cat Object Light Source Magnitudes

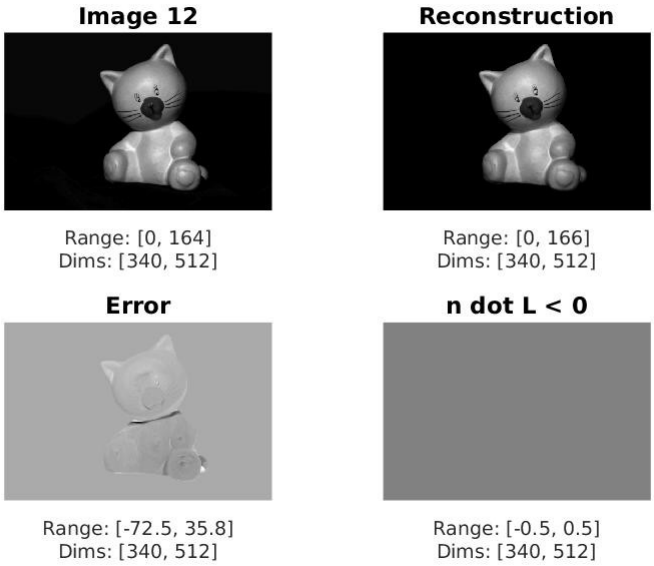


Figure 22: Detailed Reconstruction Error for the 12th image of the Cat Object

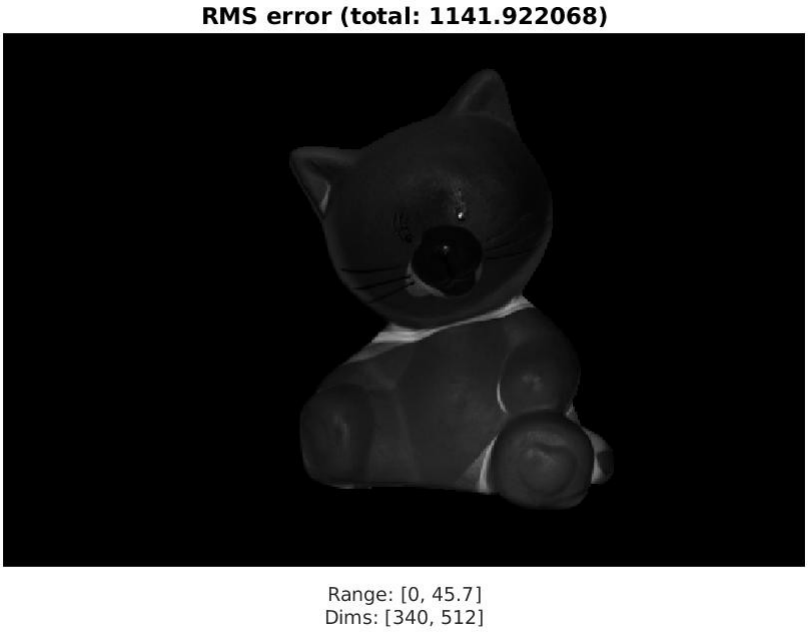


Figure 23: Total RMS Error for the Cat Object with additional light sources

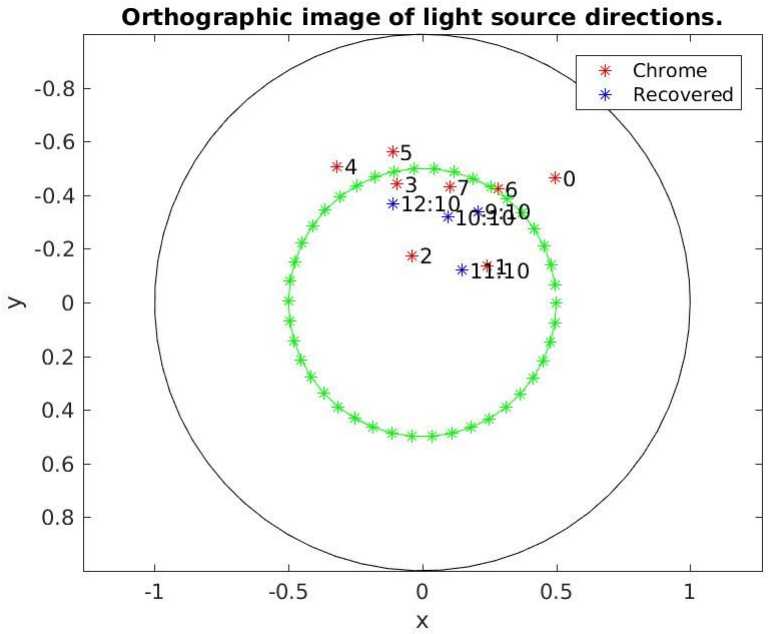


Figure 24: Orthographic Image of Recovered Buddha Object Light Source Directions

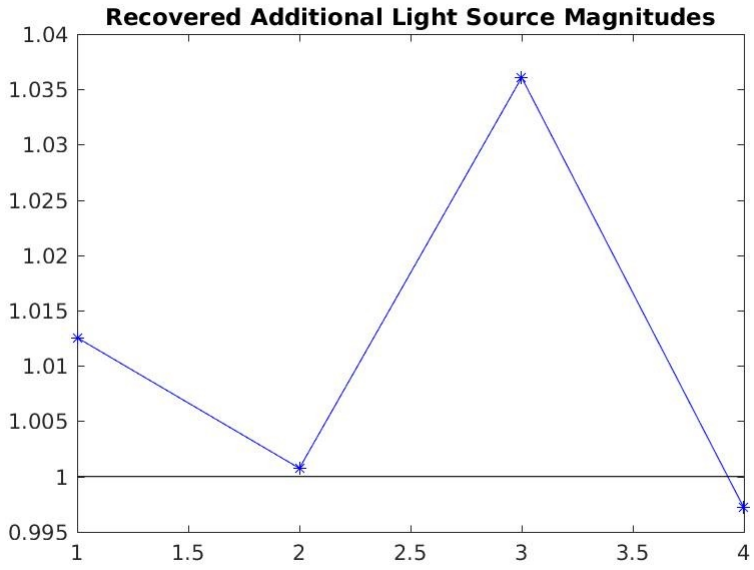


Figure 25: Recovered Additional Buddha Object Light Source Magnitudes

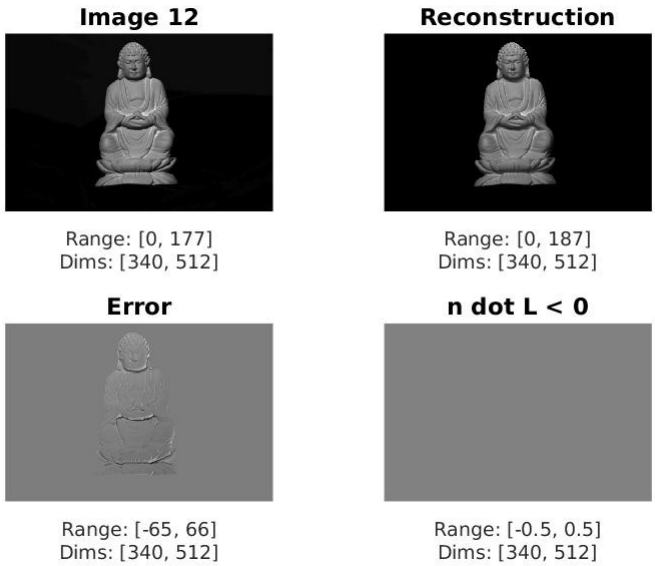


Figure 26: Detailed Reconstruction Error for the 12th image of the Buddha Object

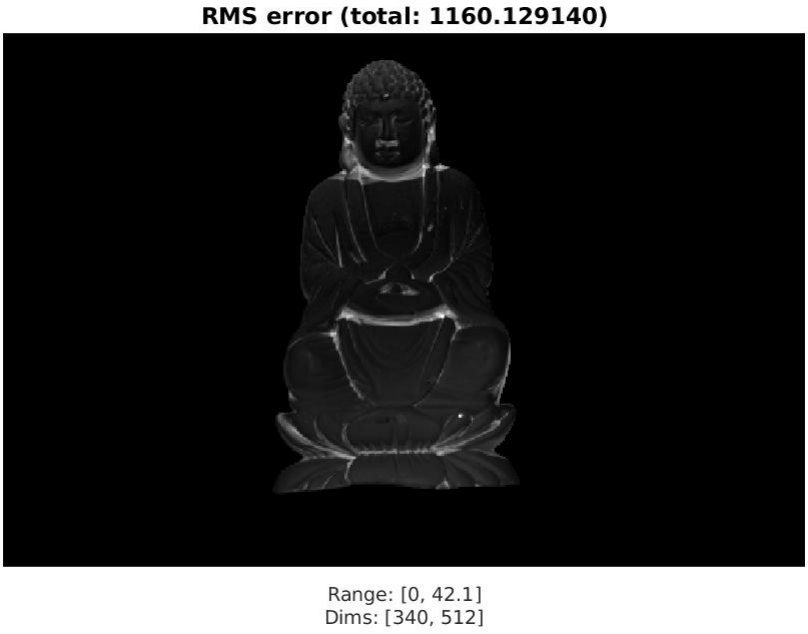


Figure 27: Total RMS Error for the Buddha Object with additional light sources