Name : Zheng Liu

Part-1 : Estimate the albedo and surface normals

1) Insert the albedo image of your test image here:

Figure 1.1 yaleB01_albedo

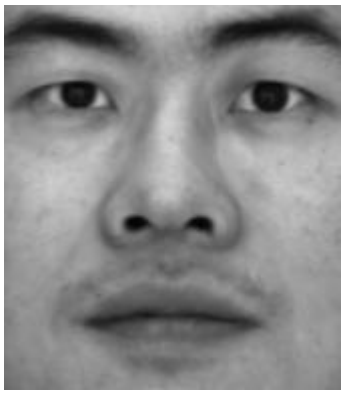


Figure 1.2 yaleB02_albedo

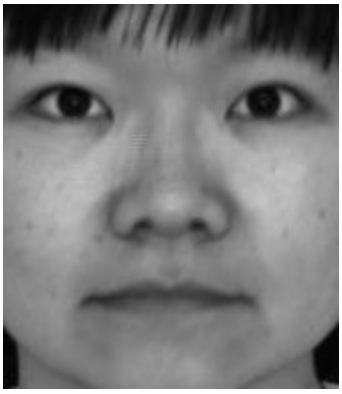


Figure 1.3 yaleB05_albedo

Figure 1.4 yaleB07_albedo

2) What implementation choices did you make? How did it affect the quality and speed of your solution?

In *preprocess* function, I first convert *abimage* and *imarray* to ndarray and follow the instruction to subtract and clip and rescale.

In *photometric_stereo* function, the *imarray* is (h, w, Nimages), 3 dimensions, and I convert it to (Nimages, h x w), 2 dimensions. Then I can directly use the *np.linalg.lstsq()* function once, with *light_dirs (Nimages, 3)* and copied_*imarray* (Nimages, h x w) as inputs, to get the solution for *g*. But this solution is in wrong shape, so I use np.reshape to change it to (h, w, 3). In the end, *albedo_image* is just the magnitude of *g*, and *surface_normal* is the unit vector.

Because I changed the shape of *imarray*, I just need to call the solver function once, and it can return me a stacked solution, and I just need to change the shape of the solution as desired. So this step can be much faster than solving 192 x 168 linear system for each pixel.

In *get_surface* function, the most important implementation is “random” method. I used 10000 iterations to do the average, and in each iteration, I start from a random starting point in the image. This point can separate the image to 4 quadrants, up left area, up right area, bottom left area and bottom right area. Then for each of these areas, I can get the covered fx(x, y) and fy(x, y). Then I randomly choose a integration method, row first or column first. Then do the integration for each area, use np.cumsum. To get the correct result, I also used np.flip() function to make sure that the direction of the np.cumsum() is correct, which means that for negative x and negative y direction, the integration value should be subtracted, but for positive x and positive y direction, the integration value should be added. After finishing all of the integration, I divide the cumulated height map by 10000 to get the averaged value.
Even though the iteration number is large, thank to the numpy functions, I can do the cumsum and flip quickly, so the total running time of my “random” method is really fast and efficient.

3) What are some artifacts and/or limitations of your implementation, and what are possible reasons for them?

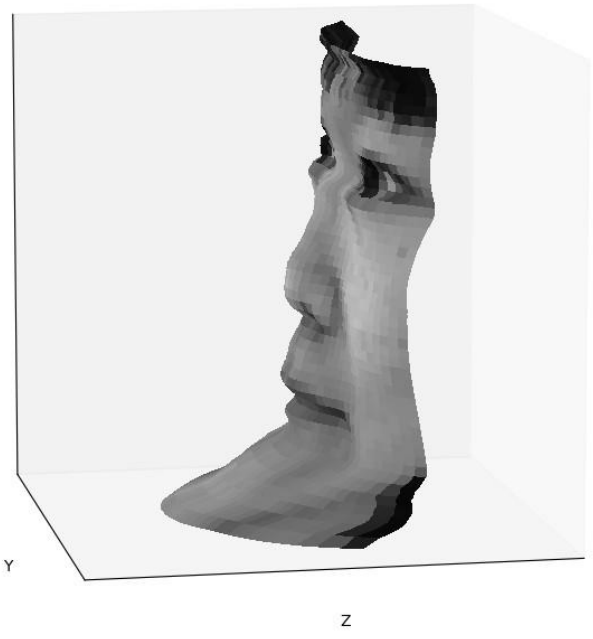


Figure 2.1 Abnormal in chin

This is the artifact of my “column” method, I think the reason is that the black hair is surrounding the chin on corners of the photo, so the integration on over that part of the column will get a huge abnormal value.

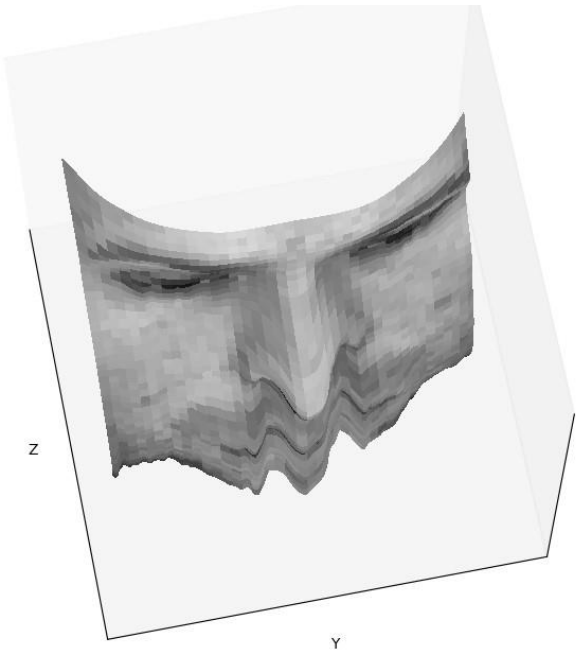


Figure 2.2 wave-shaped mouth

This is the artifact of my “row” method. Two sides along the nose is totally distorted, which indicated that there were shading parts along the nose, so the integrated value along the nose is abnormal.

4) Display the surface normal estimation images below:

Figure 3.1 yaleB01_normals_x

Figure 3.2 yaleB01_normals_y

Figure 3.3 yaleB01_normals_z

Figure 3.4 yaleB02_normals_x

Figure 3.5 yaleB02_normals_y

Figure 3.6 yaleB02_normals_z

Figure 3.7 yaleB05_normals_x

Figure 3.1 yaleB05_normals_y

Figure 3.9 yaleB05_normals_z

Figure 3.10 yaleB07_normals_x

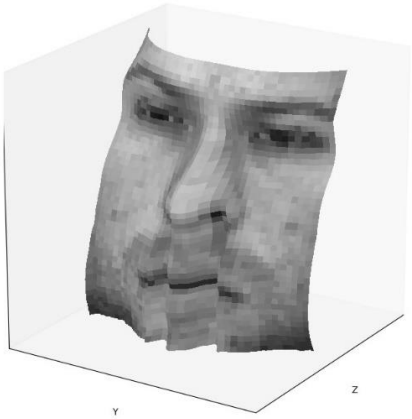
Figure 3.11 yaleB07_normals_y

Figure 3.12 yaleB07_normals_z

Part-2 : Compute Height Map

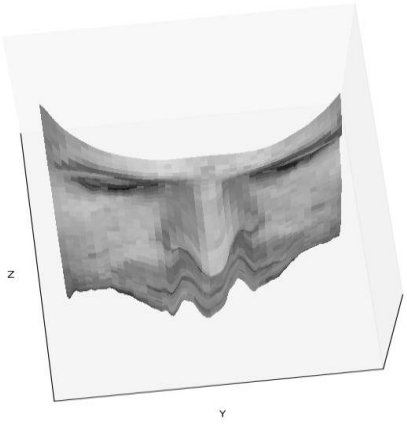
5) For every subject, display the surface height map by integration and display. Select one subject, list height map images computed using different integration method and from different views; for other subjects, only from different views, using the method that you think performs best. When inserting results images into your report, you should resize/compress them appropriately to keep the file size manageable -- but make sure that the correctness and quality of your output can be clearly and easily judged.

I select yaleB01:
Row:

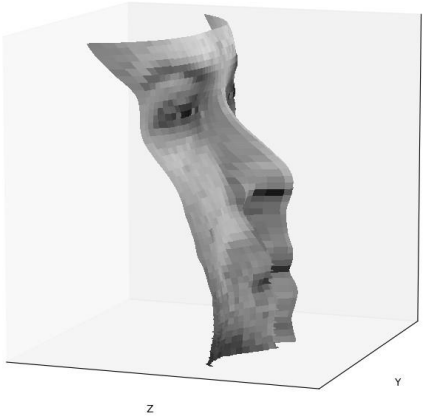


Row method up right

Row method front

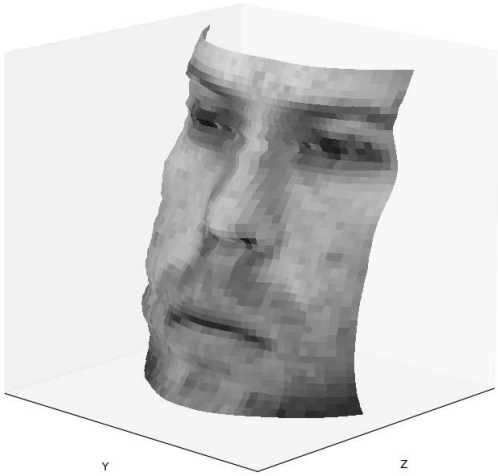


Row method up



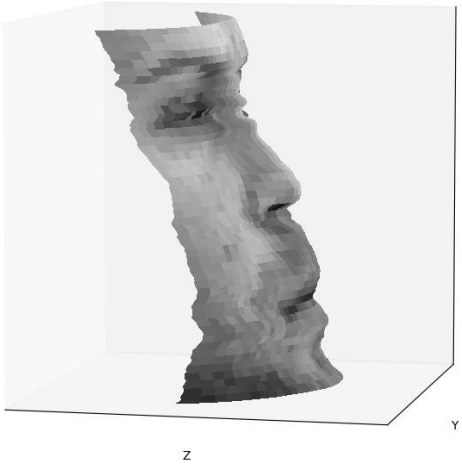
Row method left

Column:

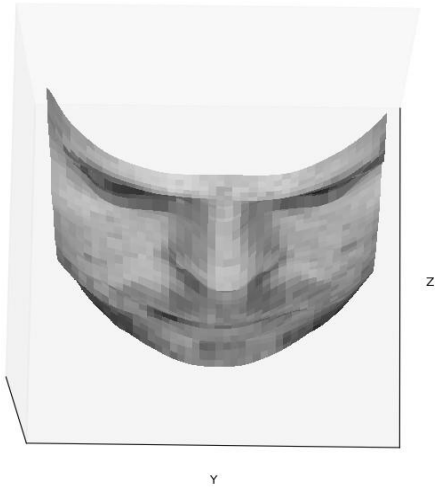


Column method right

Column method front

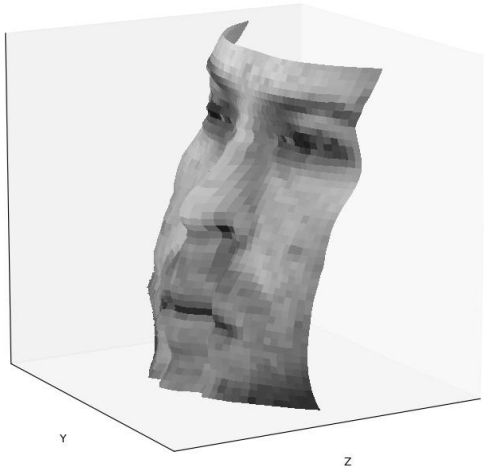


Column method left

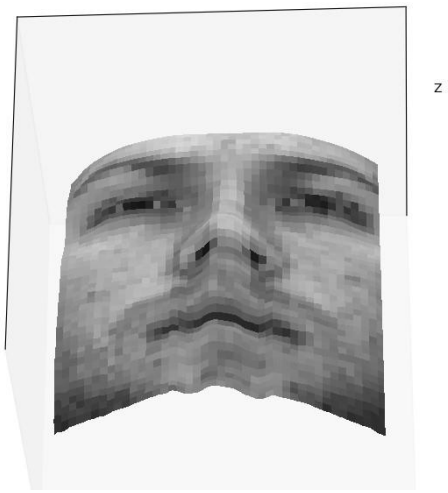


Column method up

Average:

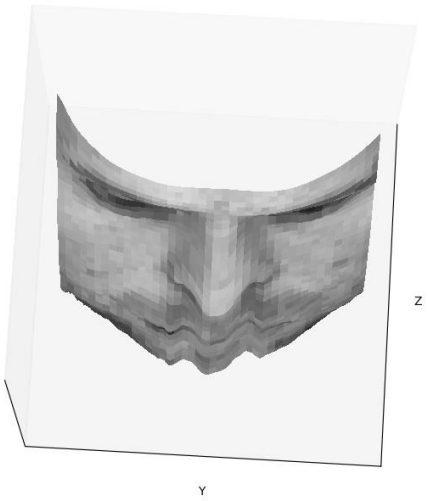


Average method right



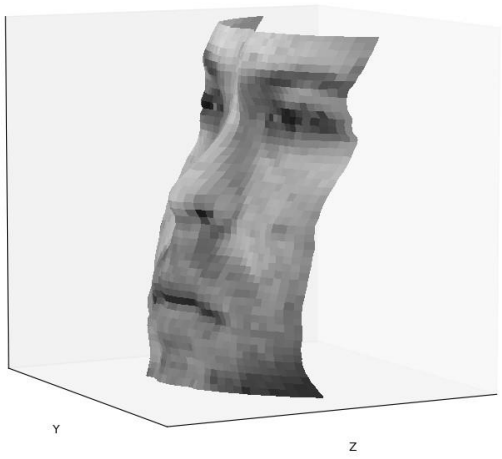
Average method bottom

Average method left

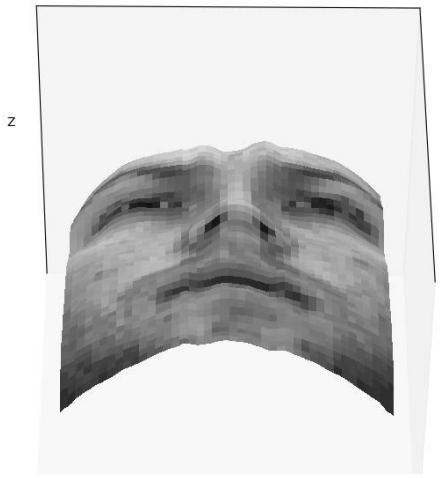


Average method up

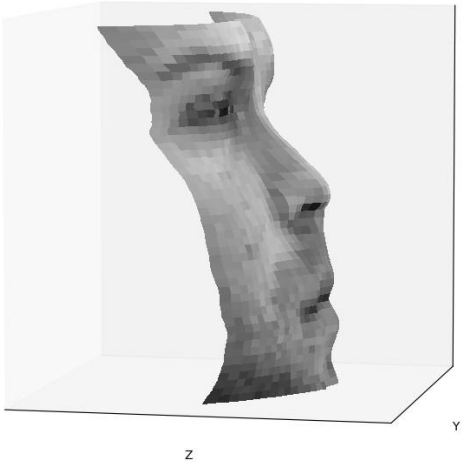
Random:



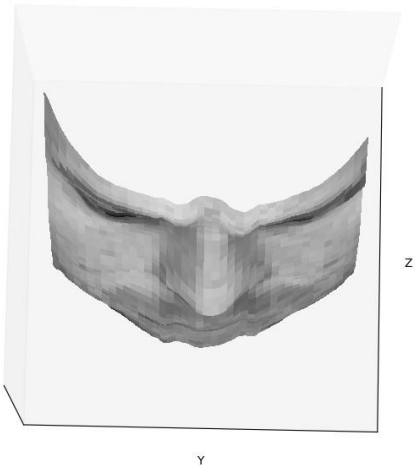
Random method right



Random method bottom

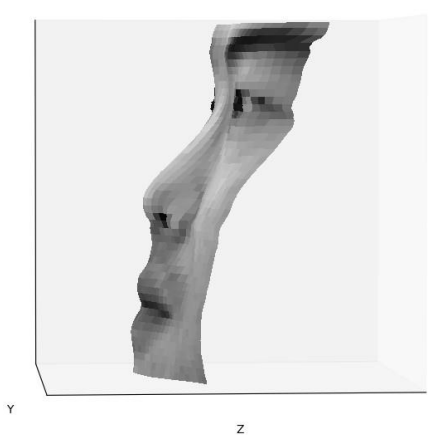


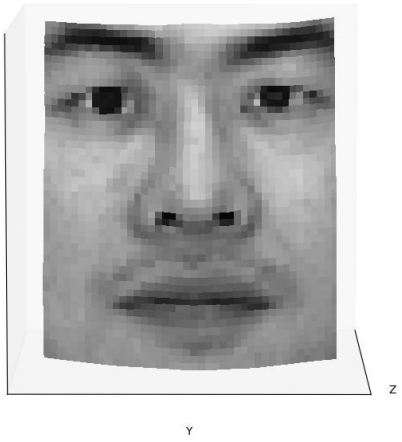
Random method left

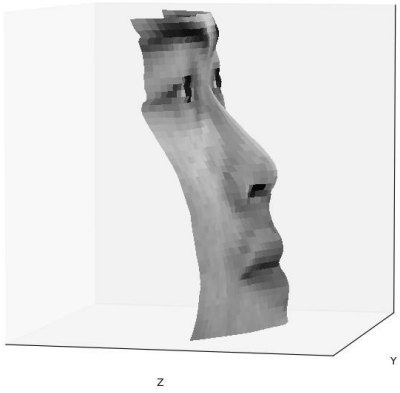


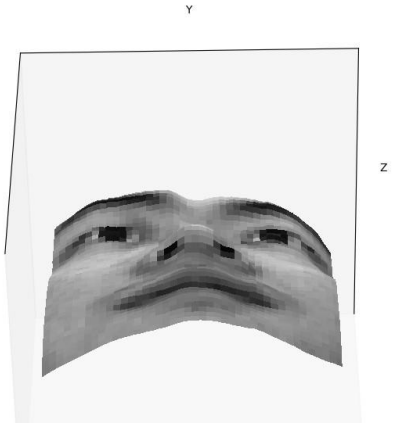
Random method up

yaleB02 Random:

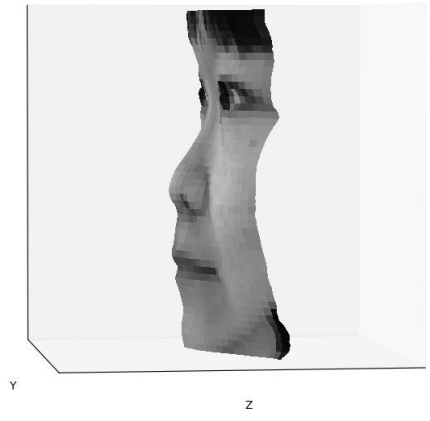


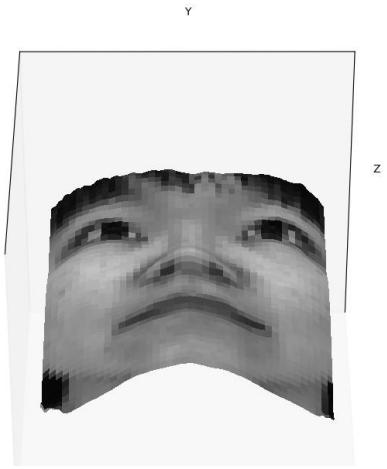


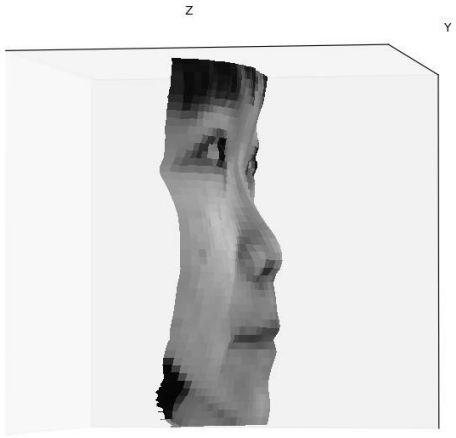


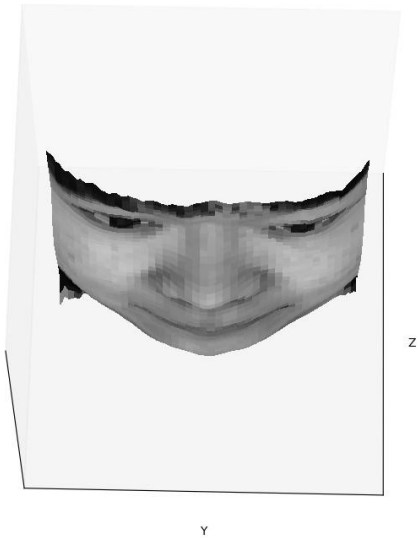


yaleB05 Random:

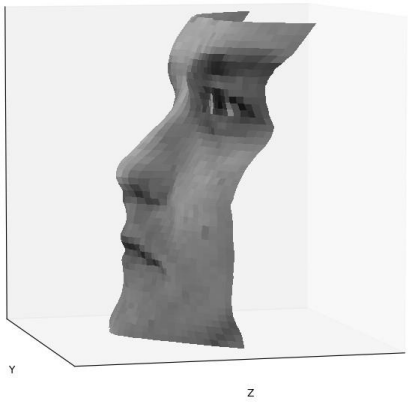


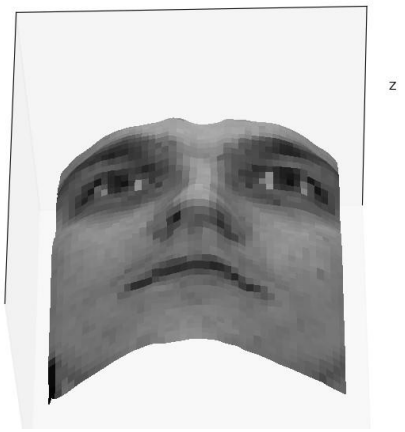


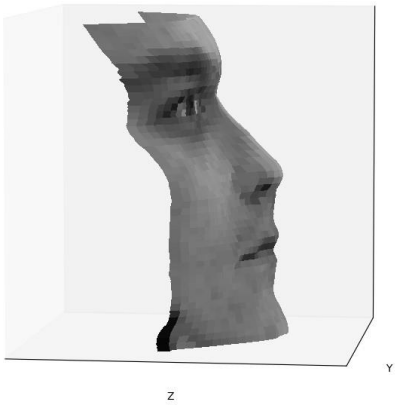


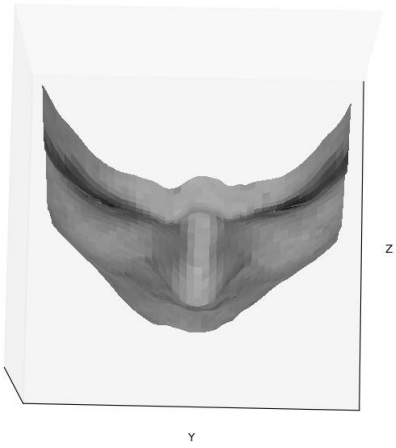


yaleB07 Random:









6) Which integration method produces the best result and why?
   Based on the reconstructed height maps for each subject, I think the “random” method always produces the best result. I think the reason is that the random method is averaging over many paths with different integration values, so this can be more robust to the noise, so even if there is some noise point on some paths, the effect of noise can be smoothed and reduced by the averaging.

7) Compare the average execution time (only on your selected subject, “average” here means you should repeat the execution for several times to reduce random error) with each integration method, and analyze the cause of what you’ve observed:

Integration method	Execution time
random	2.89213s
average	0.00059s
row	0.00036s
column	0.00038s

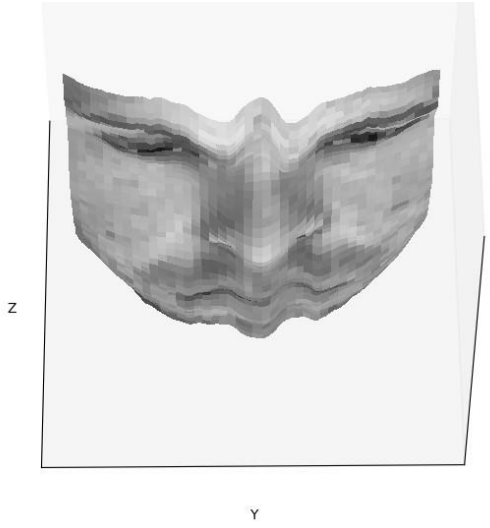
I found that the running time of row and column methods are similar and average method is a little bit longer. And the random method is much longer than other methods. The reason is that the row and column method only do the integration once. Average method does the integration twice. But the random method does the integration 10000 times, in my code. This behavior is expected. In each iteration of the random method, the calculation is similar to row and column method, but it has to run many times to get the final result.

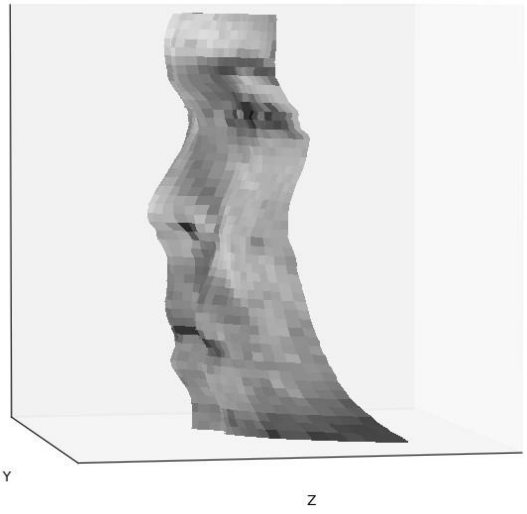
Part-3 : Violation of the assumptions

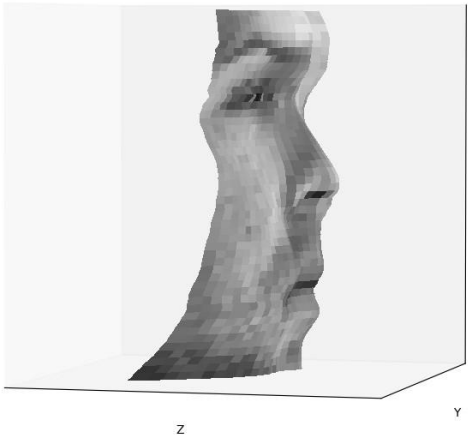
8) Discuss how the Yale Face data violate the assumptions of the shape-from-shading method covered in the slides.
   There are shadows on subject’s face. Some parts of the face are dark with no accurate data. And there are inter-reflections on subject’s face. And the skin is not smooth and uniform over all parts. In this picture, there is a shade beside the nose. And the skin is not smooth when there is moustache.

9) Choose one subject and attempt to select a subset of all viewpoints that better match the assumptions of the method. Show your results for that subset.

Albedo image







10) Discuss whether you were able to get any improvement over a reconstruction computed from all the viewpoints.
I do not think that the recovered image is better with this chosen subset. Even though the albedo image is better, the 3d view images is not so good. The reason is that the chosen subset contains less information than the original image set. Even though this subset may contain less noise and error, the useful information is also lost.

Part-3 : Bonus

Post any extra credit details/images/references used here.
I did nothing for extra credits.