Assignment 1: Shape from Shading

Zhenye Na(zna2)

1. Implementation solution

(1) **Preprocess data:** Due to linearity of light, I subtract ambient image from each image in `imarray` and make sure no pixel is less than zero, then I rescale values in imarray to be between 0 and 1 by dividing by 255 (or the highest value in `imarray`).

(2) **Estimate the albedo and normals:** I set up the linear system to solve the least square results to get g by stacking the vectors for every pixel into a 3 x npix matrix.

(3) **Compute the surface height map by integration:** Integration of discrete values simply is summing their discrete values. Here we have four different integration methods.

- **Column Integration:** Integrating first the rows, then the columns. Here we are giving more weight to **y** gradient because we mainly use q to integrate over the image.
- **Row Integration:** Integrating first along the columns, then the rows. Here we are giving more weight to **x** gradient because we mainly use p to integrate over the image.
- **Average Integration:** Average of the first two options.
- **Random Integration:** Average of multiple random paths. This method increased the quality of output height_map. I think this is because this random path method treats **y** gradient or **x** gradient equally, so it is not give more 'weight' on one particular gradient.

Below is the algorithm for Photometric Stereo.

Obtain many images in a fixed view under different illuminants
Determine the matrix $\mathcal{V}$ from source and camera information

Inferring albedo and normal:
For each point in the image array that is not shadowed
  Stack image values into a vector $\boldsymbol{i}$
  Solve $\mathcal{V}\boldsymbol{g} = \boldsymbol{i}$ to obtain $\boldsymbol{g}$ for this point
  Albedo at this point is $|\boldsymbol{g}|$
  Normal at this point is $\frac{\boldsymbol{g}}{|\boldsymbol{g}|}$
  p at this point is $\frac{N_1}{N_3}$
  q at this point is $\frac{N_2}{N_3}$
end
Check: is $(\frac{\partial p}{\partial y} - \frac{\partial q}{\partial x})^2$ small everywhere?

Integration:
Top left corner of height map is zero
For each pixel in the left column of height map
  height value = previous height value + corresponding q value
end
For each row
  For each element of the row except for leftmost
    height value = previous height value + corresponding p value
  end
end

Algorithm 2.2: Photometric Stereo.

For 4 different input images, average running time of 4 methods is presented as follows:

Method	Running Time
'Column'	0.0025(s)
'Row'	0.0031(s)
'Average'	0.0051(s)
'Random'	13.2373(s)

Table 1. Average running time of four methods

Even Though the running time is in an increasing trend, but the running time for first three method is very closed. Furthermore, the quality of output height_map image is kind of also in an increasing trend. Random integration method gives a more ideal result.

2. Output of different methods

Here are the different output images for each method with taking ‘yaleB02’ for example.

(1) Column method

yaleB02_albedo yaleB02_normals_color yaleB02_normals _x yaleB02_normals _y yaleB02_normals _y

Figure 1. YaleB02: Albedo and Surface Normals via Column method

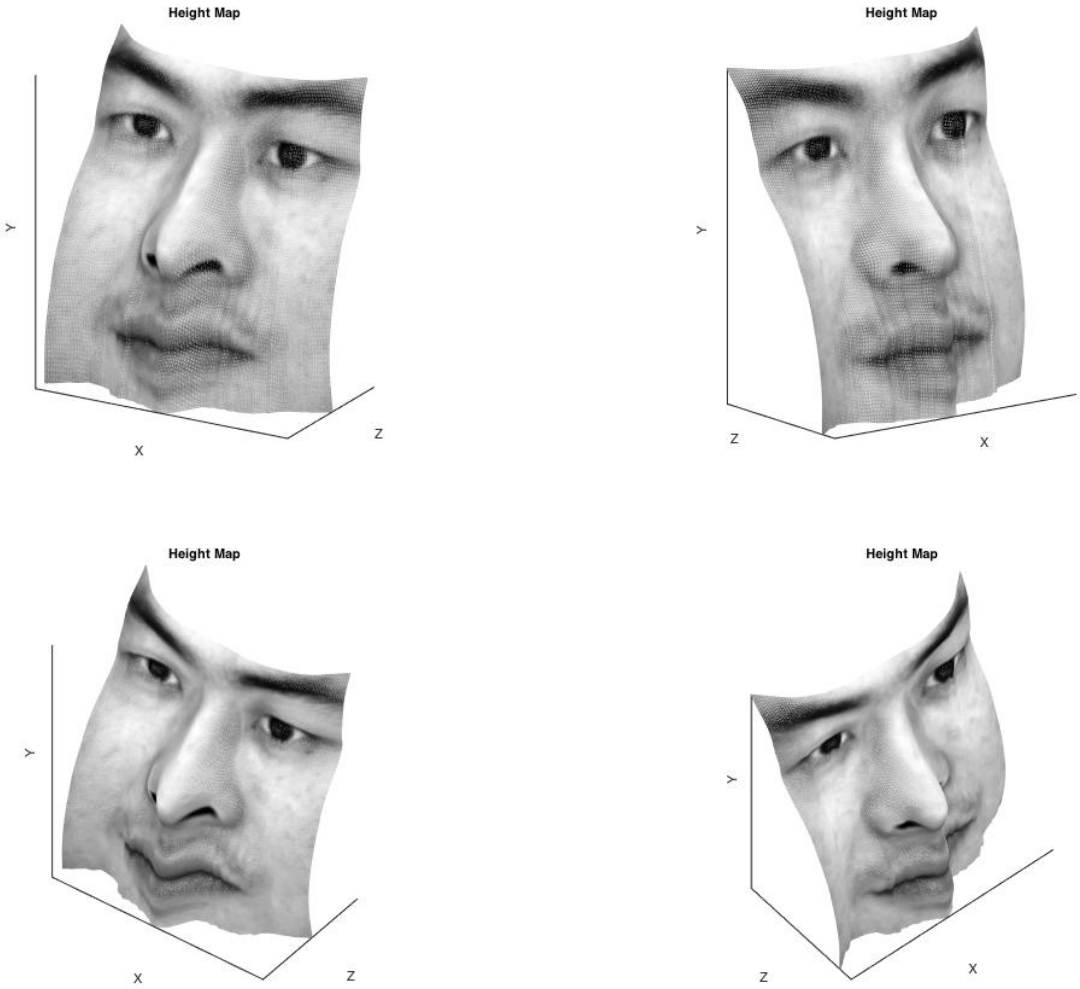


Figure 2. YaleB02: Recovered Surface via Column method

(2) Row method

yaleB02_albedo | yaleB02_normals_color | yaleB02_normals _x | yaleB02_normals _y | yaleB02_normals _z

Figure 3. YaleB02: Albedo and Surface Normals via Row method

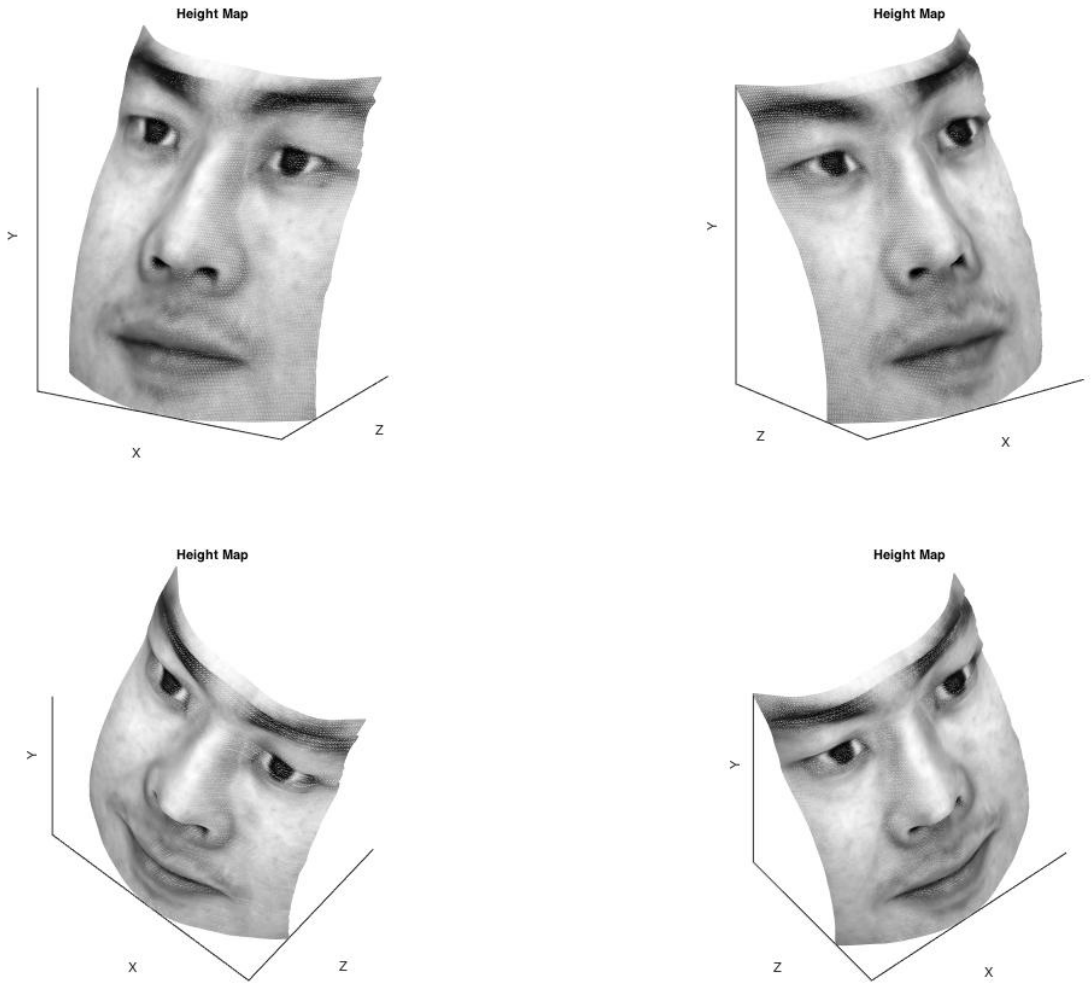


Figure 4. YaleB02: Recovered Surface via Row method

(3) Average method

yaleB02_albedo | yaleB02_normals_color | yaleB02_normals _x | yaleB02_normals _y | yaleB02_normals _z

Figure 5. YaleB02: Albedo and Surface Normals via Average method

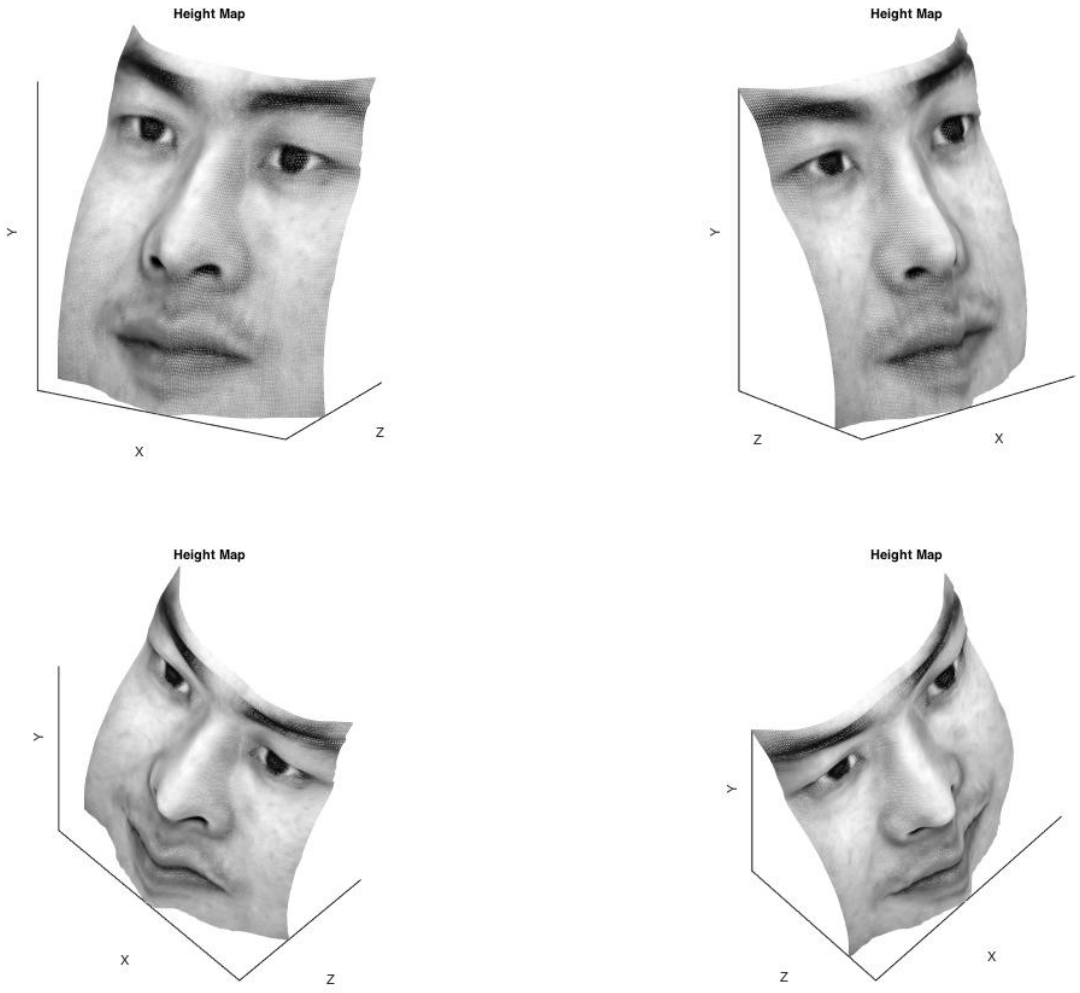


Figure 6. YaleB02: Recovered Surface via Average method

(4) Random method

yaleB02_albedo yaleB02_normals_color yaleB02_normals _x yaleB02_normals _y yaleB02_normals _z

Figure 7. YaleB02: Albedo and Surface Normals via Random method

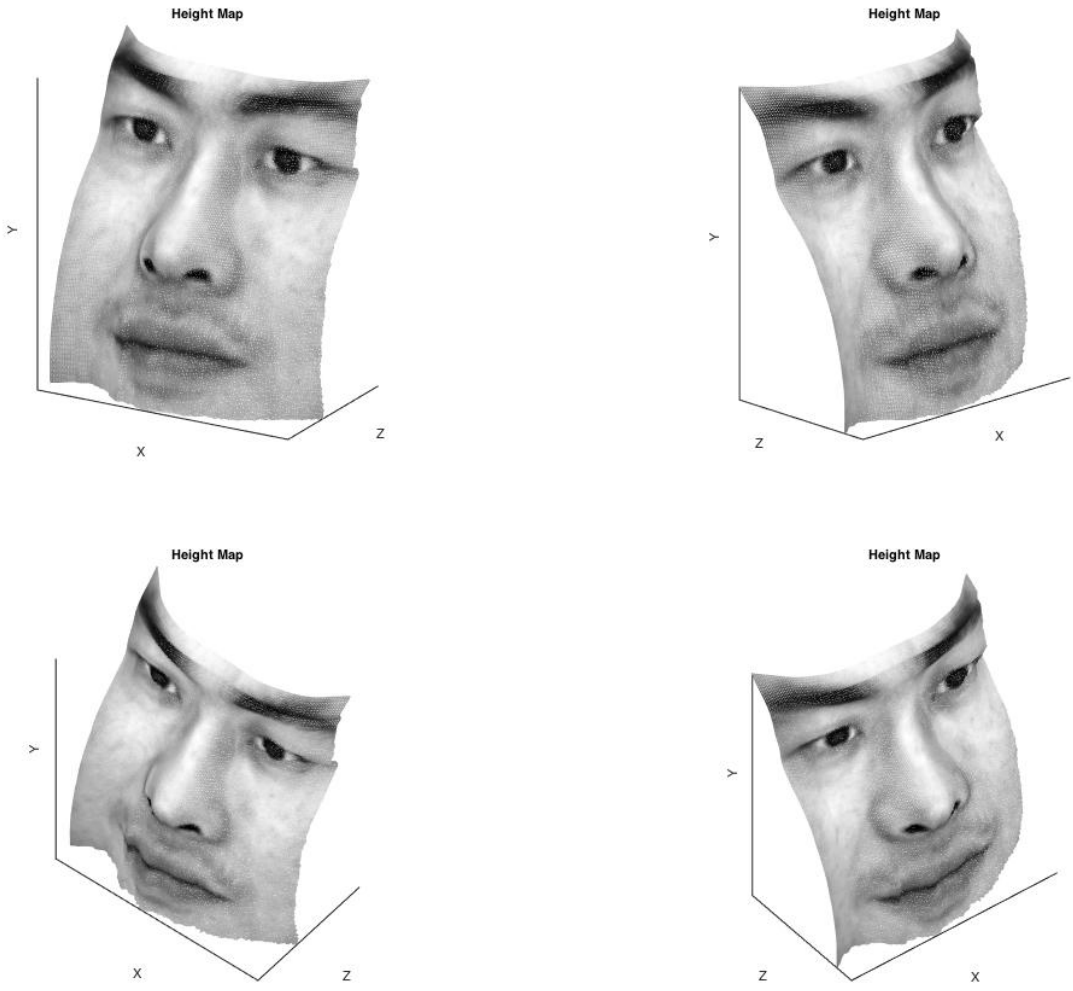


Figure 8. YaleB02: Recovered Surface via Random method

(5) Discussion of different approaches

For all the four methods, to some extent, reflect something of the true surface of the portrait and reveal the difference of these four methods.

For Column method, we first integrate the first row, then we integrate along columns. So in this case, we kind of focus more on the 'y_derivative' which is the q value. Furthermore, from the recovered height map, we can observe that the shape of 'mouth' seems to have the same shape of 'nose'. That because we integrate from up to down and we keep the 'height' of nose and add to the 'mouth' part. So we get a nose-like mouth in the recovered height map of first method. Besides, I perform four experiment of Column method and take the average, this method is the most time-saving one.

For Row method, we integrate the first column first, then integrate along rows. So we mainly update values using p value, which is the 'x_derivative'. So within the recovered height map, the height of nose is kind of be flattened because the nearby pixels do not have such 'heights'. The Average method is just take the average of the first two approaches.

For the **Random method**, it gives the **best** quality of the output. The height map image I attached in the previous section, I only use 25 random paths to integrate and then take the average based on the number of different paths. Because the path you integrate is random chosen and you take the average of multiple values. The weightage of p and q is kind of equally treated so the recovered image is the best. Furthermore, if you generate more paths to integrate, the quality will also be enhanced, but computational complexity is based on the number of random paths you choose and also the size of the images. If either of those is pretty large, it will be very time-consuming and also the quality of image is not increased linearly.

3. Output display

(1) YaleB01

yaleB01_albedo | yaleB01_normals_color | yaleB01_normals _x | yaleB01_normals _y | yaleB01_normals _z

Figure 9. YaleB01: Albedo and Surface Normals via Random method

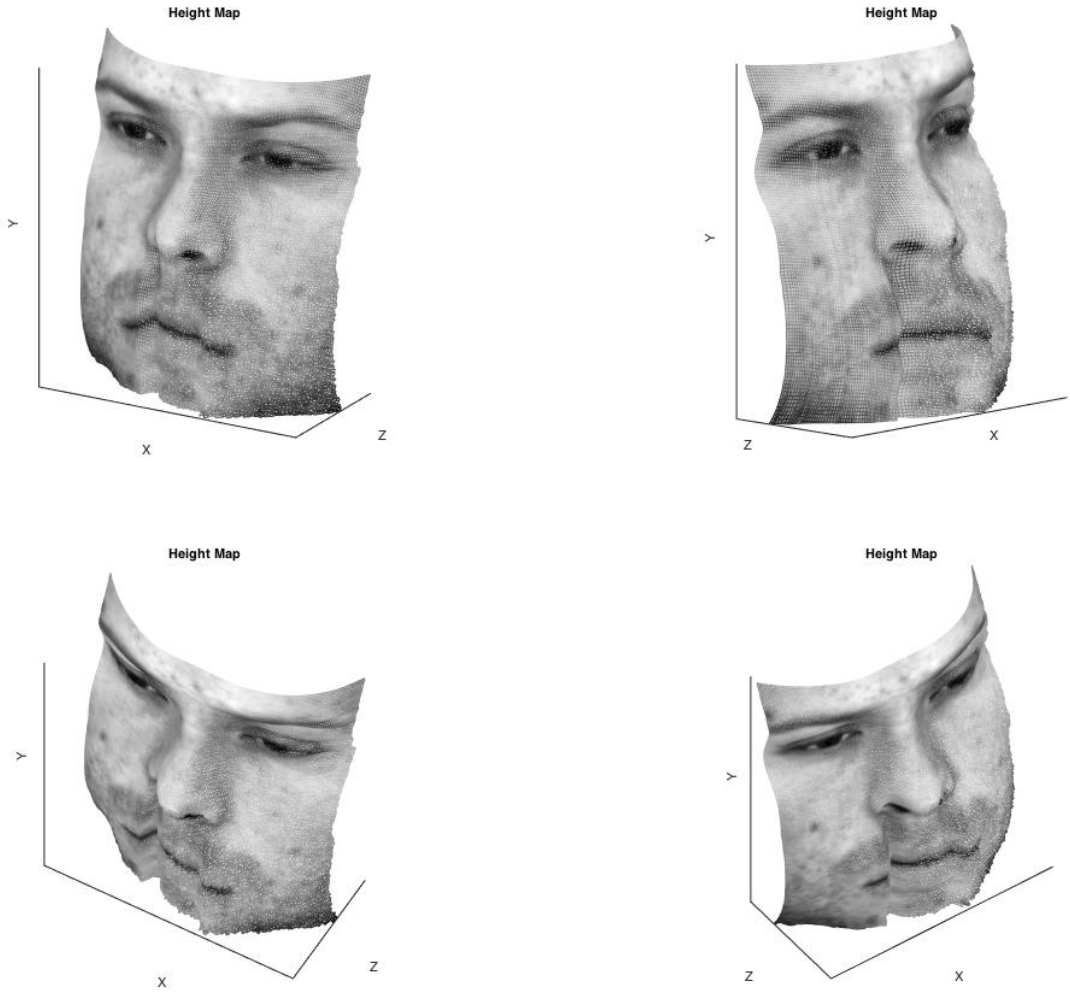


Figure 10. YaleB01: Recovered Surface via Random method

(2) YaleB02

yaleB02_albedo | yaleB02_normals_color | yaleB02_normals _x | yaleB02_normals _y | yaleB02_normals _z

Figure 11. YaleB02: Albedo and Surface Normals via Random method

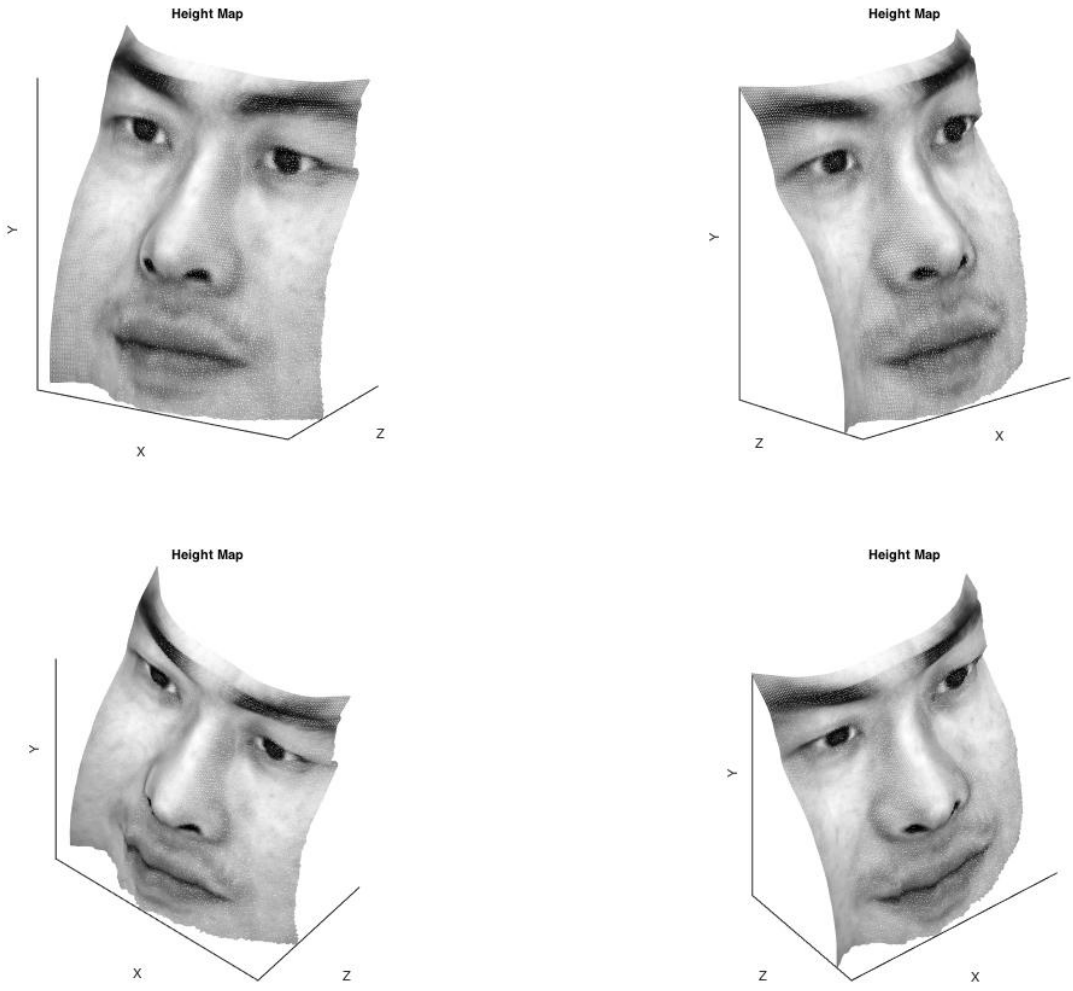


Figure 12. YaleB02: Recovered Surface via Random method

(3) YaleB05

yaleB05_albedo yaleB05_normals_color yaleB05_normals _x yaleB05_normals _y yaleB05_normals _z

Figure 13. YaleB05: Albedo and Surface Normals via Random method

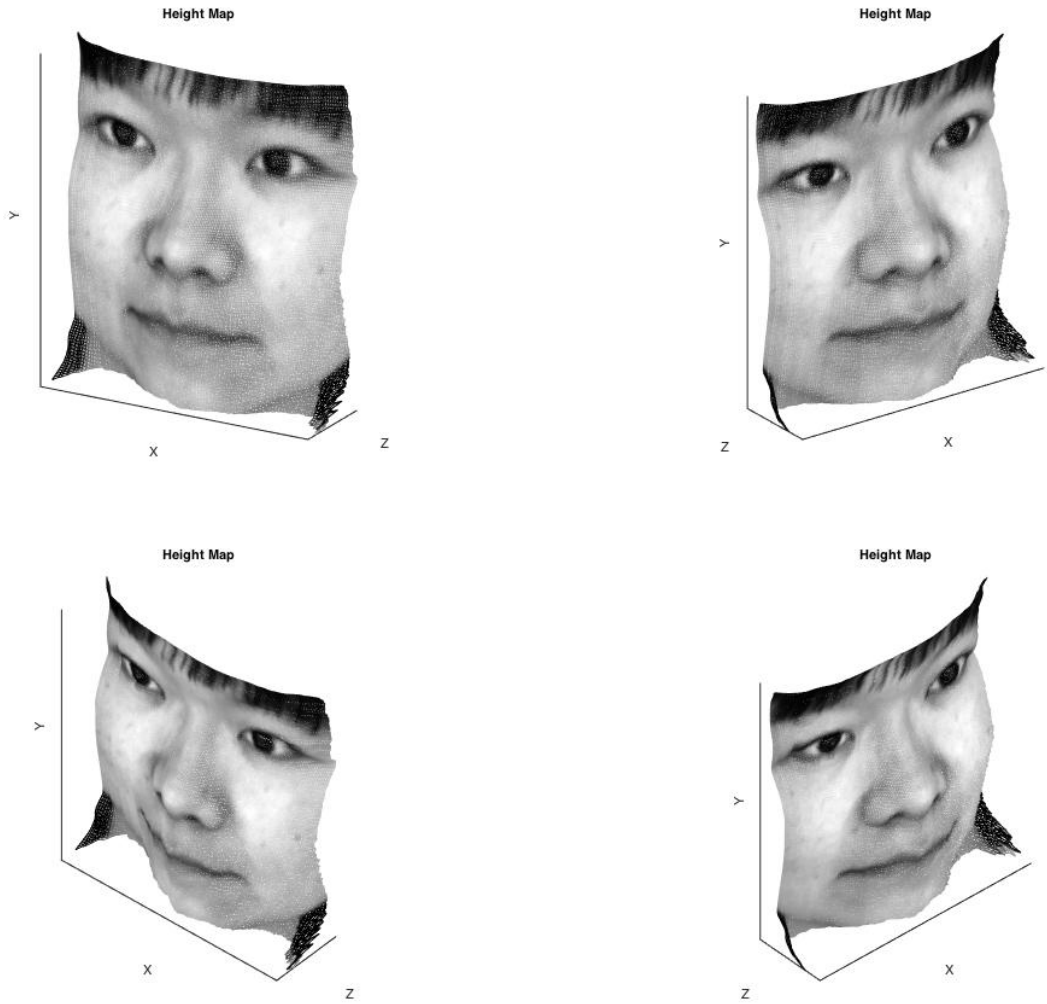
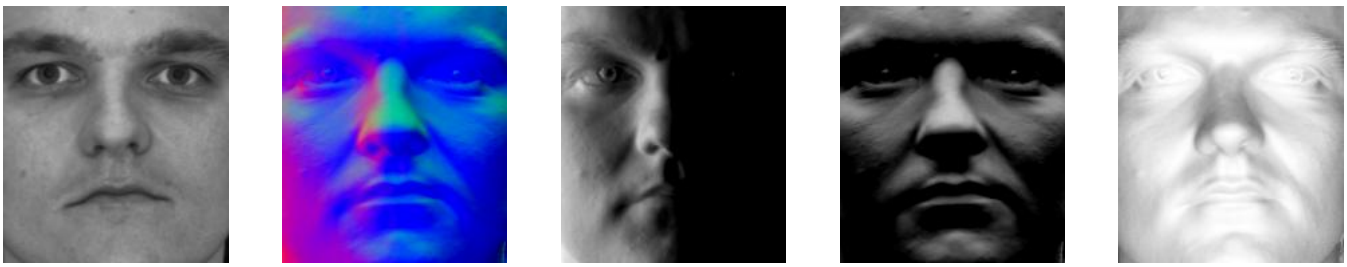


Figure 14. YaleB05: Recovered Surface via Random method

(4) YaleB07



yaleB07_albedo yaleB07_normals_color yaleB07_normals _x yaleB07_normals _y yaleB07_normals _z

Figure 15. YaleB07: Albedo and Surface Normals via Random method

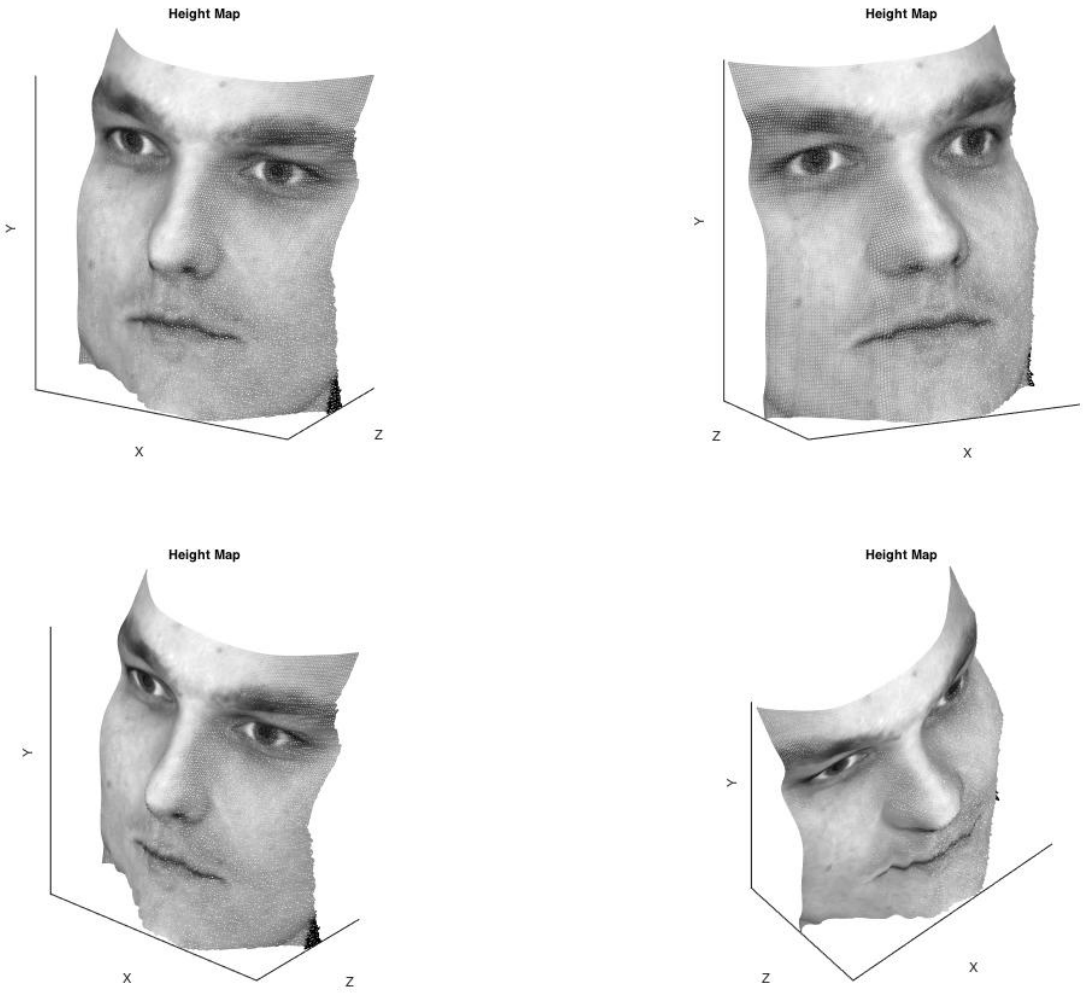


Figure 16. YaleB07: Recovered Surface via Random method

(5) Hot map

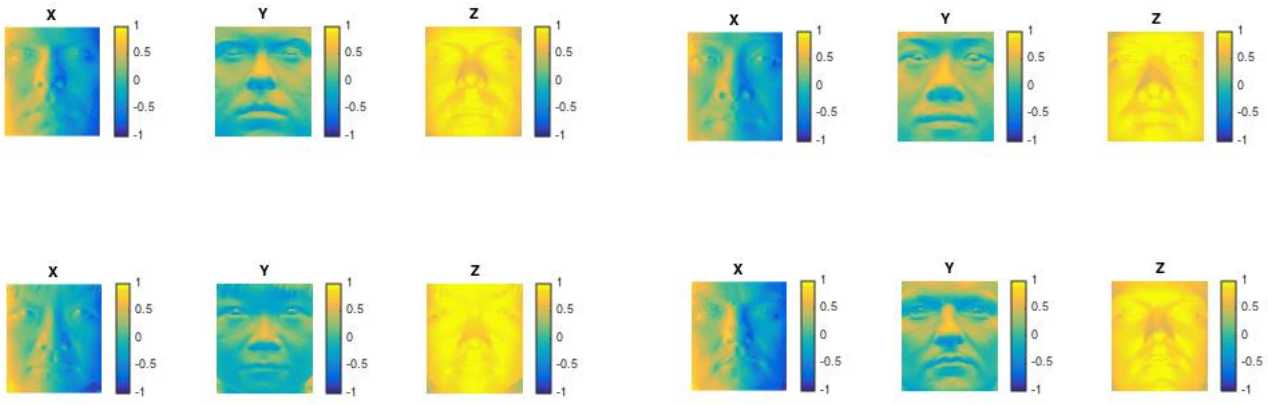


Figure 17. Hot map for 4 images via Random method

4. Discussion

(1) Yale Face data violates the assumptions of the shape-from-shading method.

Here are the assumptions that Photometric Stereo makes:

- A Lambertian object
- A local shading model (each point on a surface receives light only from sources visible at that point)
- A set of known light source directions
- A set of pictures of an object, obtained in exactly the same camera/object configuration but using different sources
- Orthographic projection

Face texture does not belong to Lambertian object, which does not have uniform surface reflectance. Then, there exists specular reflection in Yale face data. Furthermore, when capturing people's face, they cannot make sure the images all look the same, there will be some tiny differences between those faces, like the shape of lip, eyebrow and so forth. Finally, it may violate

the hypothesis that it is an orthographic projection.

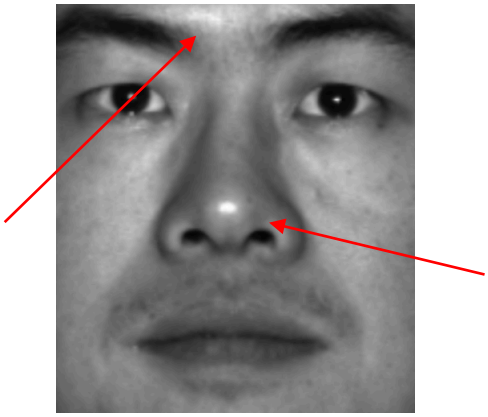


Figure 18. Specular reflection in Yale face data

(2) Limitations of implementation.

- The violations of Yale face data with assumptions shape-from-shading method.
- The `randi` function in the Random integration method. In **MathWorks** Official Website, this function generates 'Uniformly distributed pseudorandom integers'. To some extent, the random paths are not that 'random' at all. Besides, the random path I selected can be represented by the picture below.

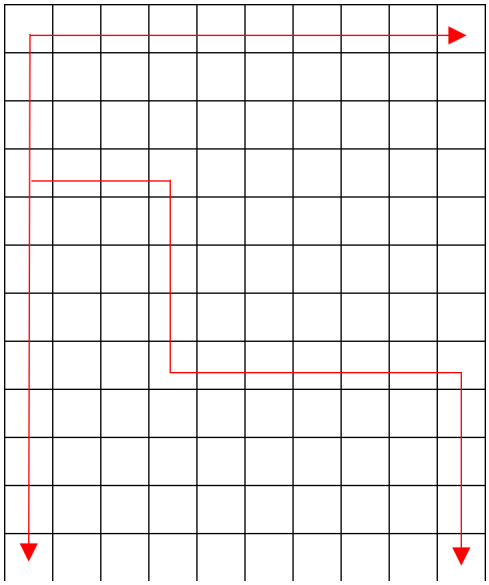


Figure 19. Random paths of Random method

(3) No. of random paths generated.

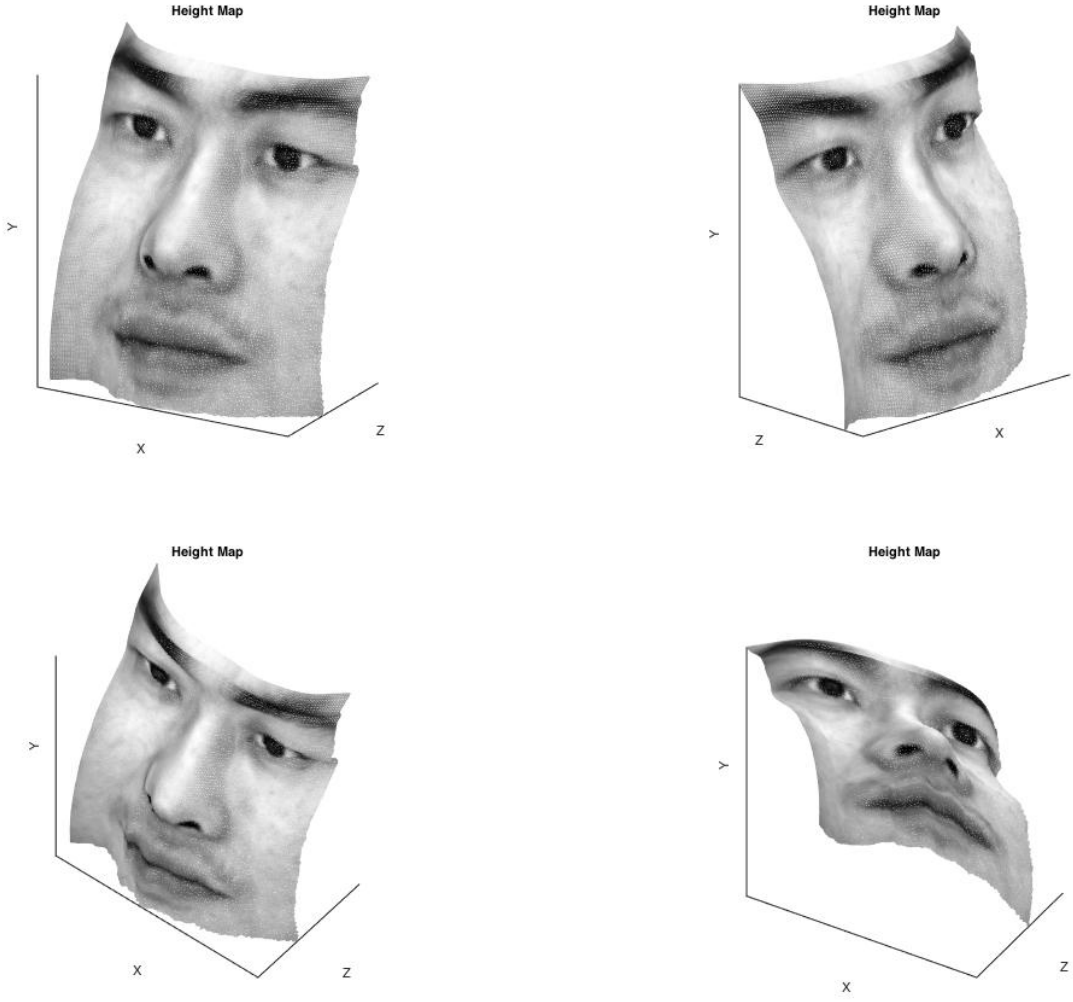


Figure 20. recovered image of YaleB02 from 25 random paths
Elapsed time is 13.553735 seconds.

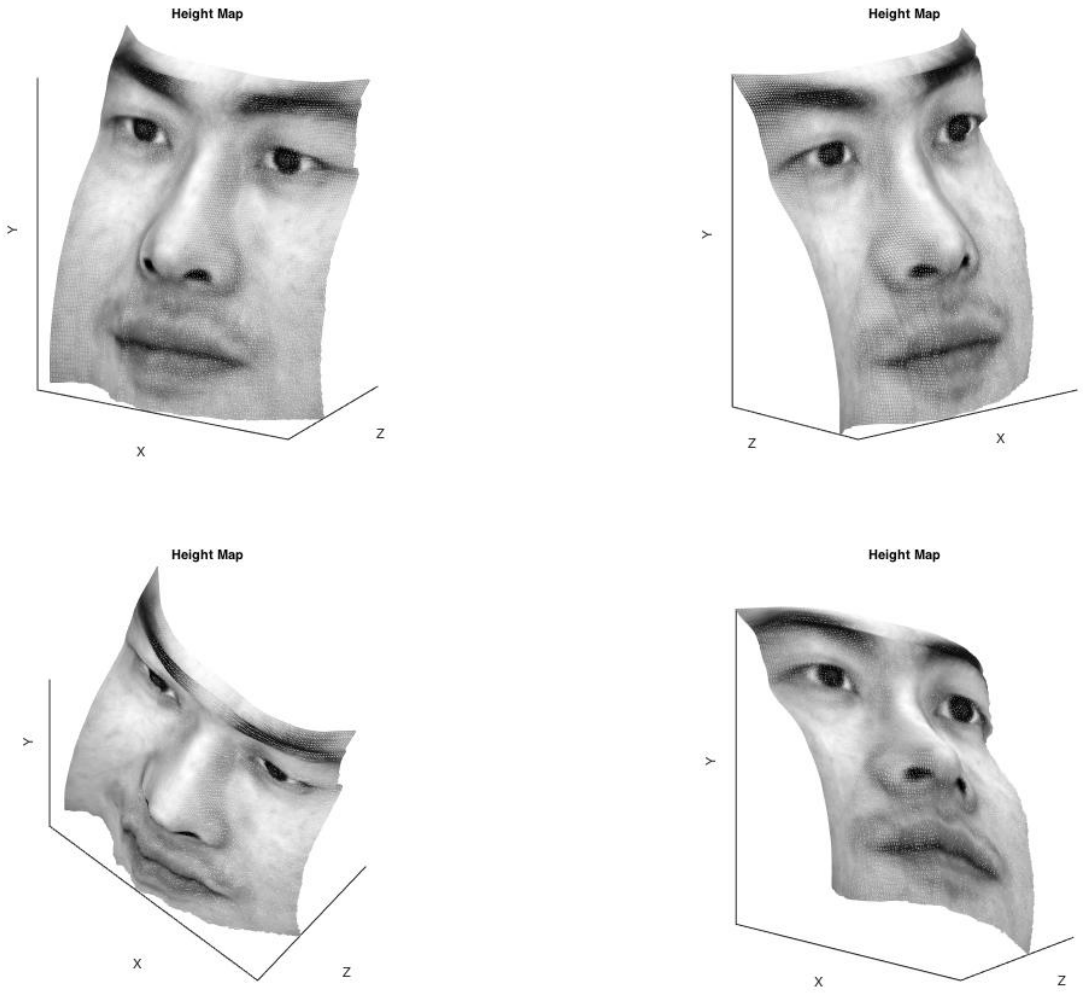


Figure 21. recovered image of YaleB02 from 50 random paths
Elapsed time is 26.403736 seconds.

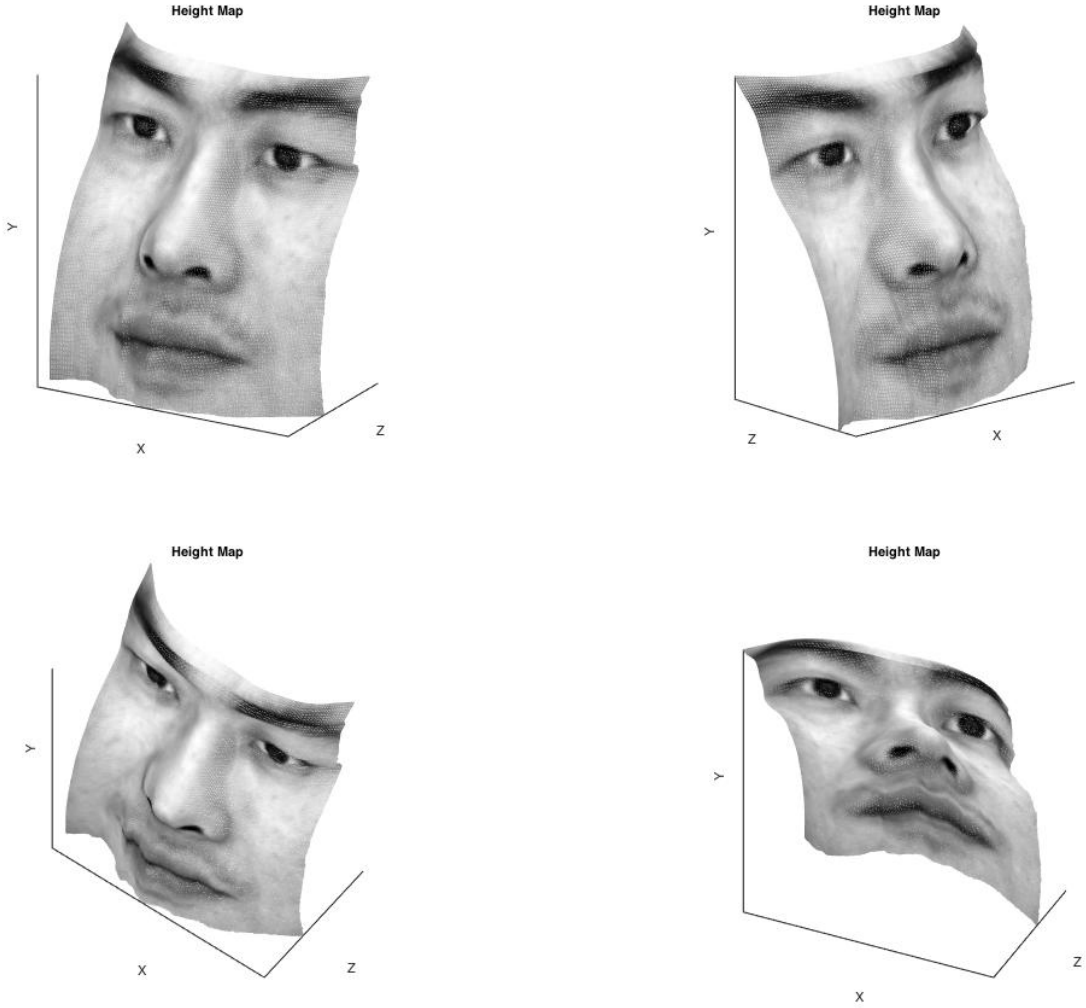


Figure 22. recovered image of YaleB02 from 100 random paths
Elapsed time is 52.648432 seconds.

From the results showing above, the differences between those recovered images are not so apparent. But with the increasing number of random path, the time running the program increases correspondingly.