Mini-project

Neha Yadav

1 Photometric stereo

The input to the algorithm is a set of photographs taken in known lighting directions and the output of the algorithm is the albedo (paint), normal direction, and the height map.

1. **Read images and light sources.** For each subject (sub-directory in the `data` folder), read in the images and light source directions. This is accomplished by the function `loadFaceImages.m` (which, in turn, calls `getpgmraw.m` to read the PGM files). The function `loadFaceImages()` returns the images for the 64 light source directions, an ambient image (i.e., image taken with all the light sources turned off), and the light directions for each image.

2. **Preprocess the data.** We need the images under the illumination of only the point light source. Fortunately, due to linearity of light we can do this by subtracting the ambient image from all the images. Also, set any negative values to zero and rescale the resulting intensities to between 0 and 1 by dividing by 255. This is Implemented in the `prepareData.m` file.

3. **Estimate the albedo and normals.** The function in `photometricStereo.m` takes as input the stack of images and the matrix of the light source directions, and returns an albedo image and normal directions. The normal directions should be encoded as a three dimensional array of size $h \times w \times 3$ where the third dimension corresponds to the x-, y-, and z-components of the normal. To solve for the albedo and the normals, we will need to set up a linear system.

4. **Compute the surface height map by integration.** instead of continuous integration of the partial derivatives over a path, it will simply be summing their discrete values. the implemention of the integration is in the `getSurface.m` file. To get the best results, one should compute integrals over multiple paths and average the results. The following variants of integration are implemented:

   - Integrating first the rows, then the columns. That is, your path first goes along the same row as the pixel along the top, and then goes vertically down to the pixel. It is possible to implement this without nested loops using the `cumsum()` function.
   - Integrating first along the columns, then the rows.
   - Average of the first two options.
   - Average of multiple random paths. For this, it is fine to use nested loops. One should determine the number of paths experimentally.

5. The results are displayed using `displayOutput` and `plotSurfaceNormals` functions.

Hint: One can start by setting the `subjectName='debug'` which creates images from a toy scene with known geometry and albedo. One can debug their code against this before trying the faces.

- **Column Integration**: Integrating first the rows, then the columns. Ripples were observed in vertical direction. Here we are giving more weigth to y gradient.
- **Row Integration**: Integrating first along the columns, then the rows. Ripples were observed while moving horizontally. More Weightage given to x gradient.

- **Average Integration**: Average of the first two options. Ripples were subsized but the did not disapper completely. More prominent ripples still observed.
- **Random Integration**: Average of multiple random paths. Quality of surface increased. Best results observed for this method.. This is beacause we average gradients form various paths and it is not specific to one particular gradient
- **Yale Face data violate the assumptions of the shape-from-shading method** Many real-world objects are textured and violate the requirement of uniform surface reflectance. skin is not strictly Lambertian, and exhibits significant subsurface scattering.While the true surface for the image is unknown, the iso-contour curves look highly plausible, with the exception of regions with depth discontinuities (such as the ridge on the image) which violate our assumption of surface differentiability.

yaleB01Albedo yaleB01nNormalsColor yaleB01NormalsX yaleB01NormalsY yaleB01NormalsZ

Figure 1: YaleB01: Albedo and Surface Normals

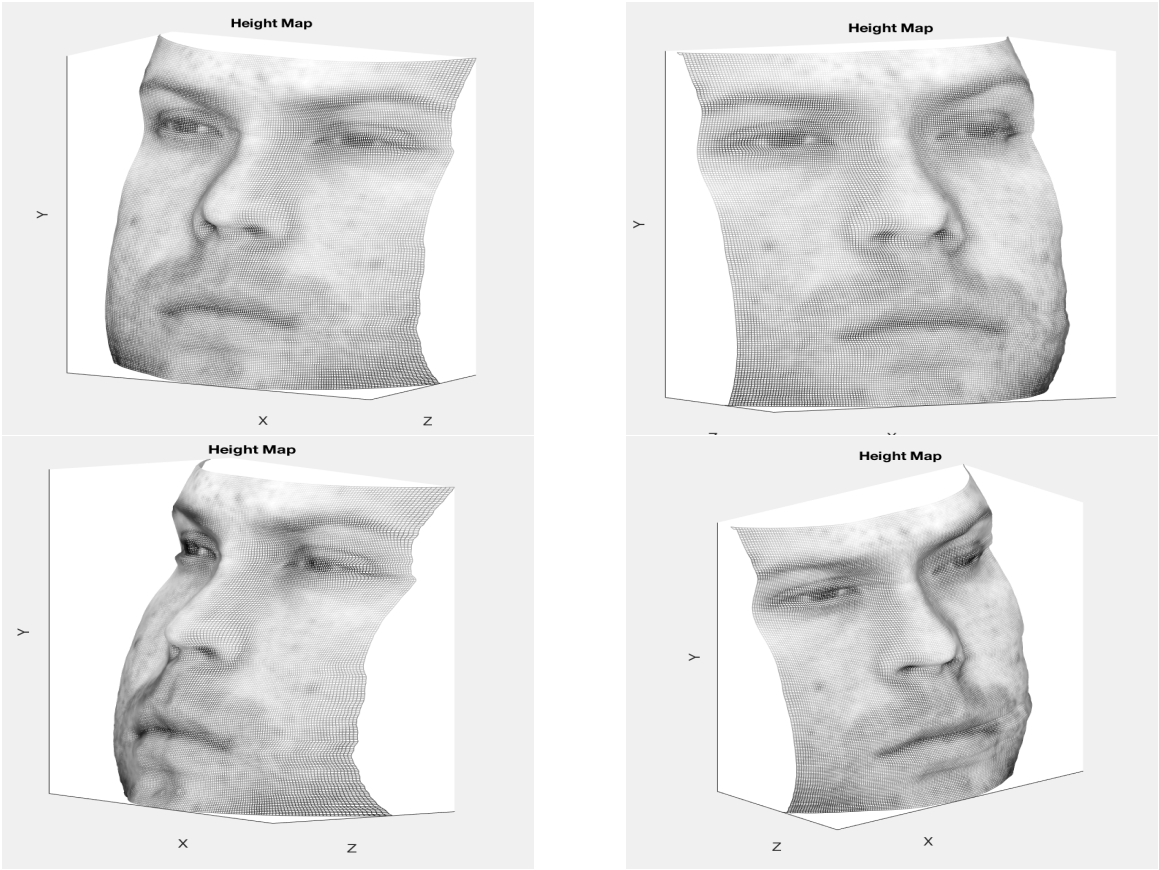


Figure 2: YaleB01: Recovered Surface

yaleB02Albedo yaleB02nNormalsColor yaleB02NormalsX yaleB02NormalsY yaleB02NormalsZ

Figure 3: YaleB02: Albedo and Surface Normals

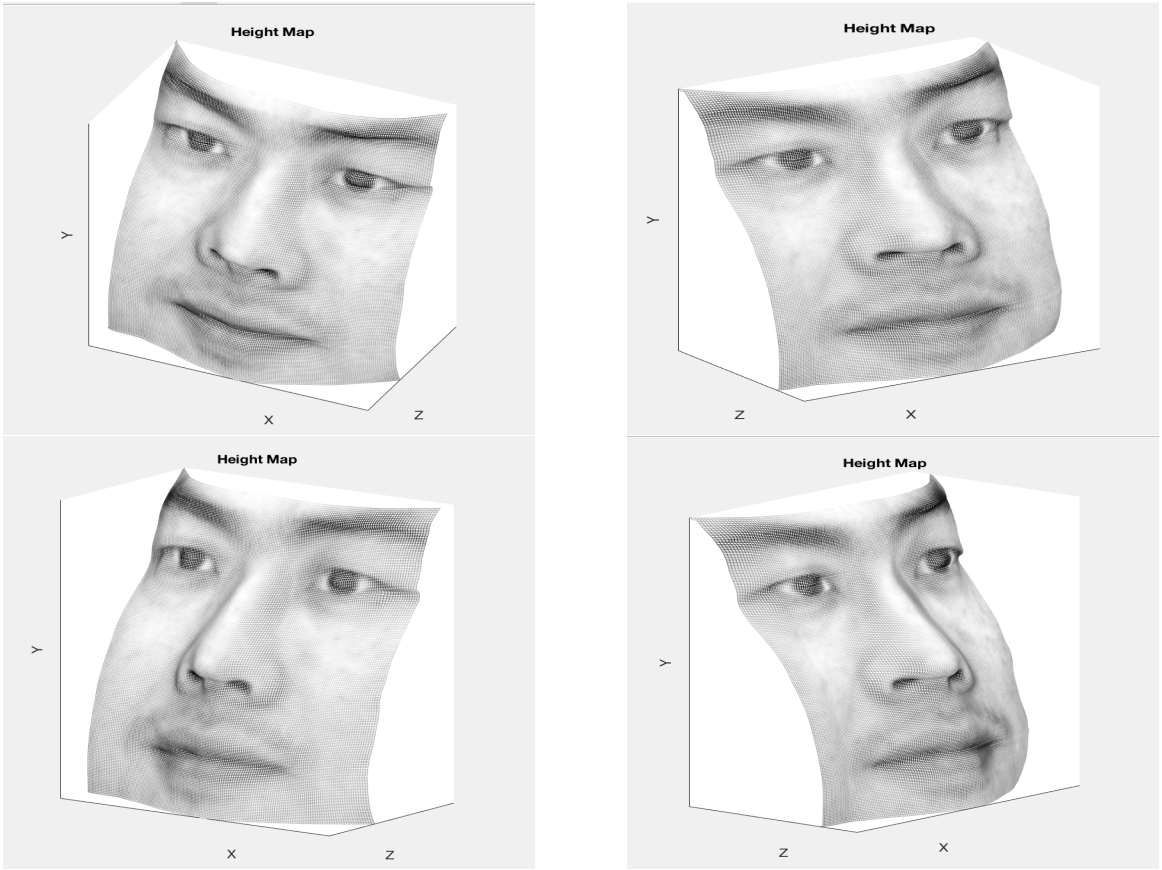


Figure 4: YaleB02: Recovered Surface

yaleB05Albedo yaleB05nNormalsColor yaleB05NormalsX yaleB05NormalsY yaleB05NormalsZ

Figure 5: YaleB05: Albedo and Surface Normals

Figure 6: YaleB05: Recovered Surface


yaleB07Albedo


yaleB07NormalsColor


yaleB07NormalsX


yaleB07NormalsY


yaleB07NormalsZ

Figure 7: YaleB07: Albedo and Surface Normals

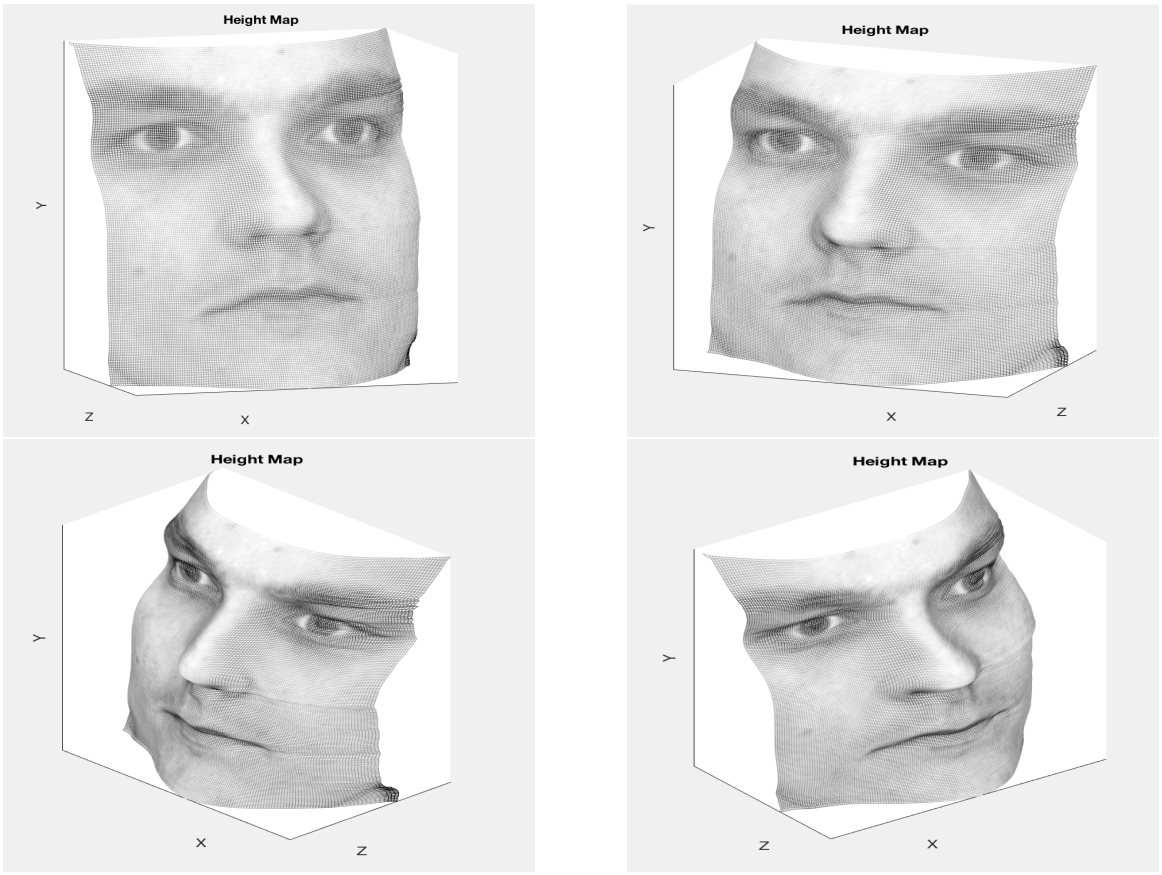


Figure 8: YaleB07: Recovered Surface

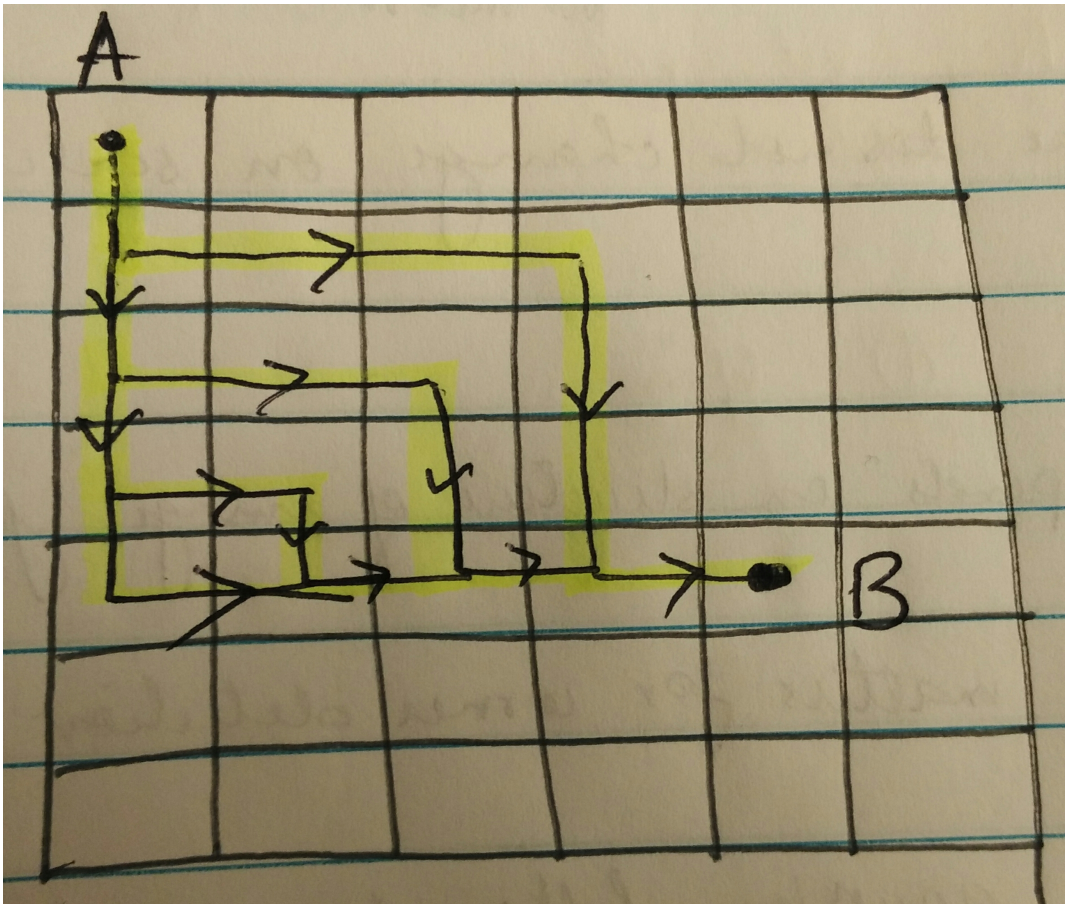
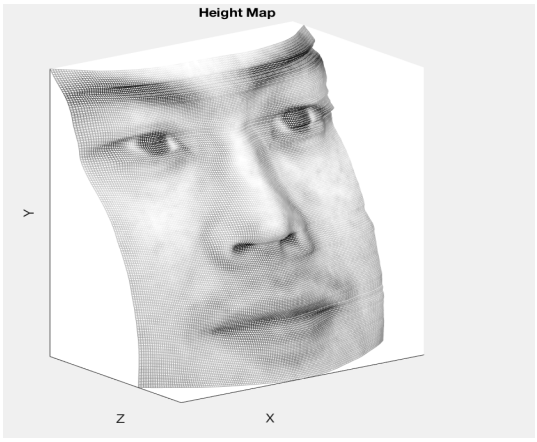
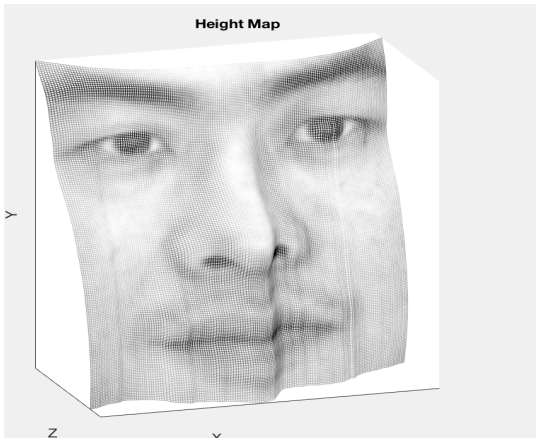


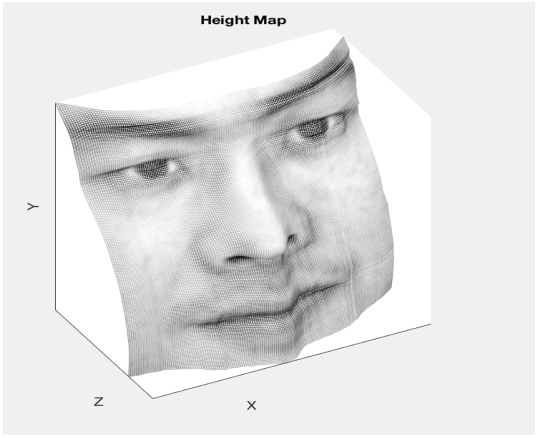
Figure 9: Random path used to reach B from A. I have calculated such random paths for each pixels and averaged them.



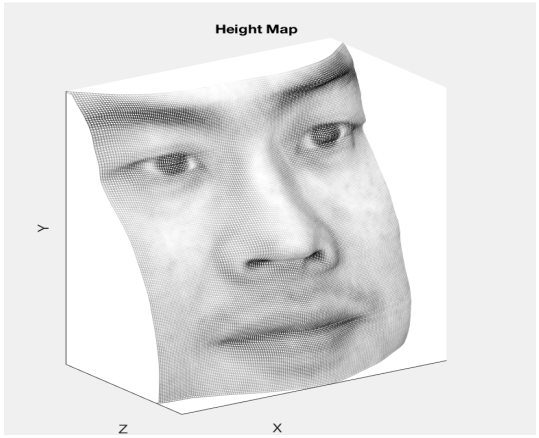
Row Integration



Column Integration



Average Integration



Random Integration

Figure 14: YaleB02: Recovered Surface using different integration methods