1. Computing a canonical view image using photometric stereo

   Below in Figure 1, I provide the image of the Lambertian surface computed from photometric stereo.

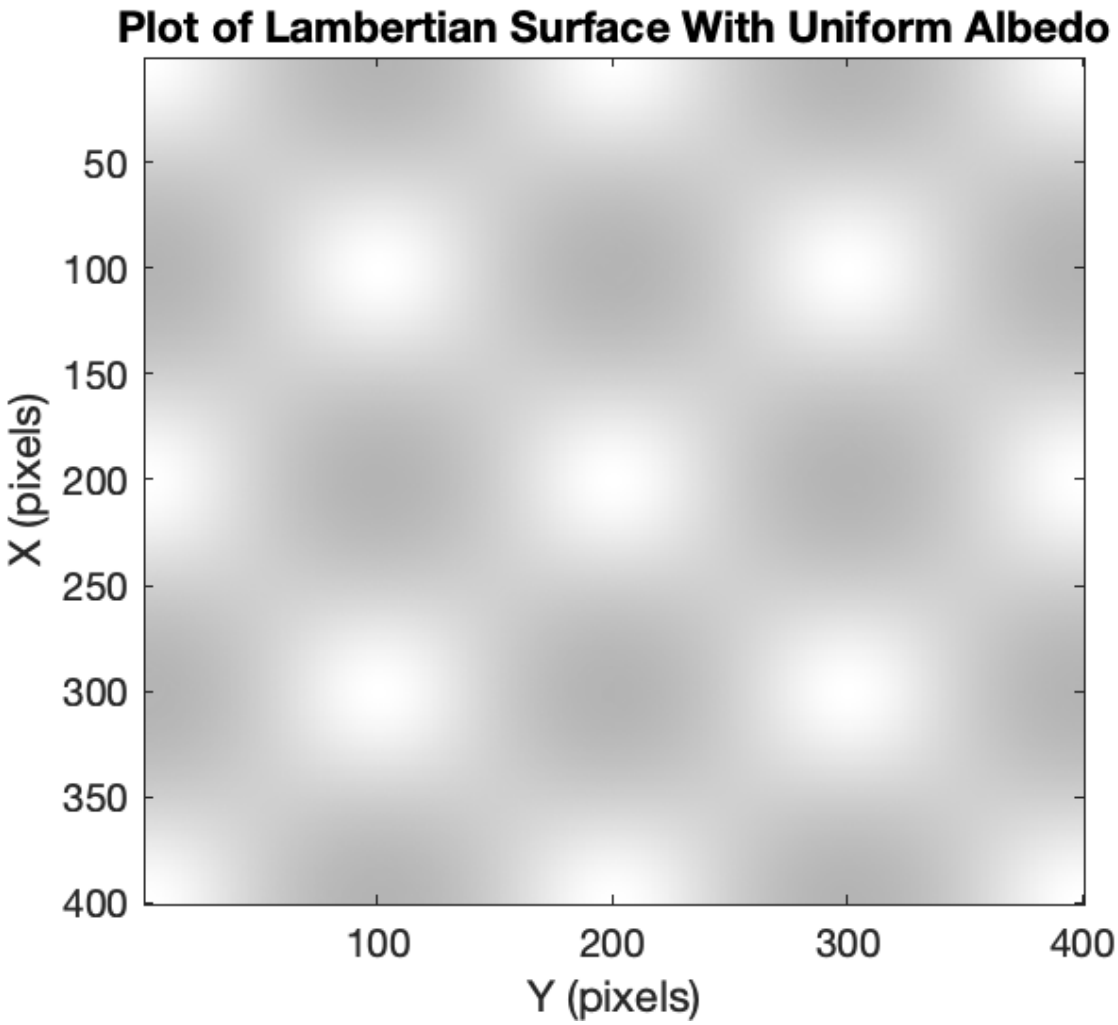


Figure 1: Plot of our Lambertian surface computed from photometric stereo in MATLAB. The X-axis represents the row index of our image matrix which grows downward, and the Y-axix represents the column index of our image matrix which grows rightward. Although the photometric stereo images had different albedo in the four quadrants, our computed Lambertian surface has uniform albedo. This was done by converting the normal vectors obtained from non-homogenous least squares into unit vectors.

Looking at Figure 1 above, we can see a light and dark checkered pattern that is also

consistent with the light and dark pattern seen in the photometric stereo images, just as a visual check.

Since our Lambertian surface image is an orthographic projection, meaning that the point (x,y,z) is simply projected to (x,y,0), only the Z component of our normalized normal vectors affect the Lambertian luminance of their respective points (pixels) on the surface. In MATLAB, this means we only needed to look at the Z-component of our normalized normal vectors in order to compute the brightness of that pixel. We are working in gray scale so we set the three RGB channels to that singular Z-component value. The maximum Z-component value of our normalized normal vectors was 1.0 and the minimum value was around 0.7, and these correspond respectively to the white and gray points on our plot.

White on the surface image means that the surface at that point is perpendicular to the camera direction, assummed to be at (0,0,1) in this assignment. Similarly, darker shades mean that surface is pointing increasingly away from the direction of the camera which corresponds to a smaller Z-component in our normalized normal vector. Notice that there are no black areas on our surface which indicates that there is no surface point that points at a perpendicular angle (or greater) relative to the position of the camera.

2. Computing a depth map of the Lambertian surface

Below in Figure 2, I provide the computed depth map of the Lambertian surface as a 3D plot.

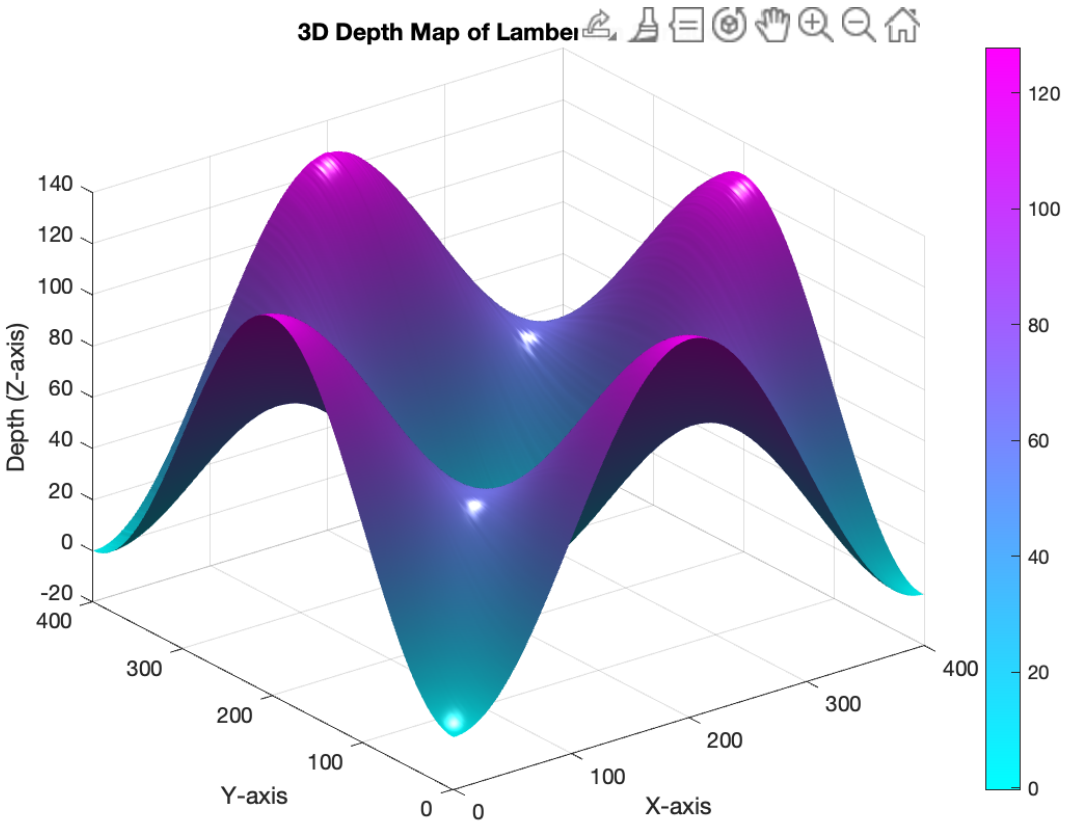


Figure 2: Depth map of the Lambertian surface plotted in MATLAB. The legend on the right denotes the gradient encoding of the depth going from low (aqua blue) to high (neon purple). The surface is sinusoidal in nature. Lighting was added (via MATLAB function light()) directly above the mapped surface to highlight the areas that were perpendicular to the direction of the camera at (0,0,1). This is seen by the reflectance of light on the peaks, troughs, and saddle areas of the mapped surface.

In order produce the depth map as seen in Figure 2 above in MATLAB, you first have to compute the partial derivatives $f_x = \frac{n_x}{n_z}$ and $f_y = \frac{n_y}{n_z}$ from our normal vectors, and these partial derivatives signify the rate at which the height (Z-value) increases with

respect to the X-value and Y-value respectively.

Having computed $f_x, f_y$, we can then initialize a 2D, 400x400 depth map of all zeroes to represent the height at every point of the Lambertian surface. Then, loop through the depth map to add the cumulative sum of $f_x$ from index to index (i,j), and then loop through the depth map once more to add the cumulative sum of $f_y$ to the cumulative sum of $f_x$ that is already stored in depth map from the previous loop. This will result in our final depth map which we can then use along with functions meshgrid() and surf() in MATLAB in order to plot the surface as in Figure 2 above.

3. Relating the 3D depth map to the canonical view image.

   The 3D depth map of the Lambertian surface in Figure 2 is consistent with the canonical view image in Figure 1. The white areas in Figure 1 correspond to the peaks, valleys, and saddle areas of the mapped surface in Figure 2 which makes sense because, physically speaking, these are the only parts of a sinusoidal surface that could be perpendicular to the direction of the assumed camera above the surface, and we defined maximum luminescence at this perpendicularity. The steep regions of Figure 2 correspond to the gray areas in Figure 1 because the normalized normal vectors at these points would have a lower Z-component which is the only determinant of brightness at that point for our photometric stereo.

   One thing in particular are the minima and maxima of our mapped surface in Figure 2. From our canonical image in Figure 1, although we see the areas that are perpendicular to the direction of the camera (white), it is not obvious which of these areas are peaks, or troughs, or saddle regions just from the canonical image alone. Here is my understanding. When we compute the depth map using the normal vectors at every point, we arbitrarily set our starting position at (0,0) to be at sea level (z=0). Then, from our derivation of $f_x, f_y$, it is implicit that a normal vector pointing in the positive

X or positive Y direction means a negative change in the Z direction with respect to that axis at that point, and a normal vector pointing in the negative X or negative Y direction means a positive change in the Z direction with respect to that axis at that point.

4. Calculating surface normal vectors from our depth map.

   Below in Figure 3, I provide the canonical view image recreated from our 3D depth map from Figure 2.

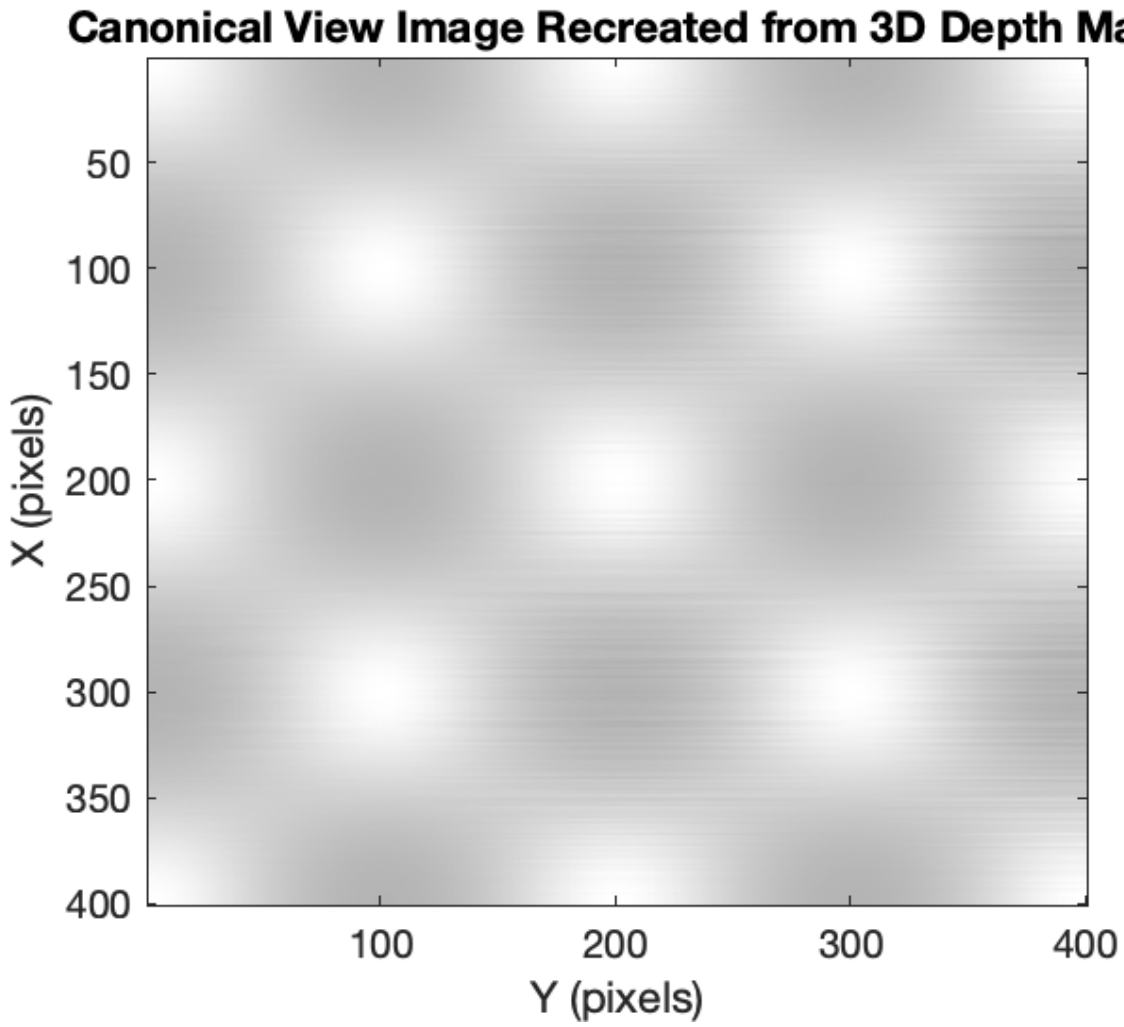


Figure 3: Canonical view image recreated from our 3D depth map. It is the same checkered pattern as Figure 1 which serves as a check of our 3D depth map in Figure 2.

The canonical view image in Figure 3 was recreated from our 3D depth map by getting the normal vectors from our computed $f_x, f_y$ gradients in question 2. To get the normal vectors from our gradients, you simply do $(-f_x, -f_y, 1)$ and then normalize the normal

vectors again so that we have workable Z-values which we plot to get our recreated canonical image as seen in Figure 3 above.

Notice that there is noise in the form of visible lines in Figure 3. I am not sure of the exact cause, but my guess is that this is rounding error from finding the gradient (through division), and then re-normalizing the computed normal vectors (which involves another division).