EECS 442 HW2

1 and 2. Estimate the albedo and surface normals:

The method to get the albedo is just as the slides of lec05. The 18th silde provides a method to calculate g(x,y). To cauculate the g for the whole surface, I use reshape() and permute() function to create an 2*(hw) reshaped imArray, named as ReshapedimArray, and then use backshash operator and the combination of permute and reshape to create the g() of the whole surface. Since g is known, I just use the matlab's dot prime function to calculate the magnitude of each (x,y) point and save it in a h*w matrix named rou. To cauculate the surface normal, I just use matlab's dot division. The code is very simple. No for loop is used since it is more efficient to use matrix calculation rather than nest for loops.

```
[h,w,n]=size(imArray);
ReshapedimArray=reshape(permute(imArray,[3 1 2]),[n, h*w]);
g=permute(reshape((lightDirs\ReshapedimArray),[3,h,w]), [2 3 1]);
rou=(g(:,:,1).^2+g(:,:,2).^2+g(:,:,3).^2).^0.5;
surfaceNormals=g./repmat(rou,[1 1 3]);
albedoImage=rou;
```

3. Compute the surface height map:

3.1: Integrating first the rows, then the columns

The integration of the row is via cumsum() function. Integrate only the first row of Fx, in horizontal direction, named as temp1. And then use repmat function to repeat the row and make it as h*w matrix.

Then integrate the whole matrix of Fy, in vertical direction, named as temp2. The sum of temp1 and temp2 is the heightMap.

3.2: Integrating first along the columns, then the rows

The integration of the row is via cumsum() function. Integrate only the first column of Fy in vertical direction, named as temp3. And then use repmat function to repeat the column and make it as h*w matrix.

Then integrate the whole matrix of Fx, in horizontal direction, named as temp4. The sum of temp3 and temp4 is the heightMap.

3.3: average

It is just like the sum of 3.1 and 3.2, and then divide the sum by 2.

3.4: Average of multiple random paths

First, we choose 100 different paths for each point and calculate the average as the height of that point. That is achieved by for loop for 100 times and divided the sum for 100.

How to decide the integration path is the core essence of this method. If the location is (x,y), then we have to move/integrate (x-1) steps in horizontal direction and (y-1) steps in vertical direction from (1,1). Thus, we first create a 1*x-1 matrix with all ones and 1*y-1 matrix with all zeros. Then I concatenate the two matrix and use randperm() to randomly permute the concatenated matrix named as C. Thus, one stands for integration on horizontal direction and zero stand for integration on vertical direction. For each step, if the corresponding value of C is 1, add the corresponding Fy of that point; otherwise, add the corresponding Fx of that point.

4: In report: Discuss the differences between the four different integration methods

4.1: discuss which method produces the best results and why

(My answer of Question 6 is also here.)

According to the image next page, for each subject, the random method produced the best results. That is because, the original ambientImage data is not perfect as the assumption (in the lecture 05's slides). Thus, if we just integrate in only one or two direction, and rely on the integration on the edge (like the first column, first row in row, column method), the error will be larger due to the reason below. However, if we use random method, since the average will cause "negative" error and "positive" error to neutralize each other to some extent, and the random method will rely less on the image data on the edge, which is not that reliable, the random method will provide the best results.

The violation of the assumption is below:

Firstly, The object is not exactly a Lambertian object. In real word, the reflection distribution is not exactly as the Lambert's law, since the surface is smoother in some place and coarser in some other place.

In addition, there is some difference in reality and the assumption of the local shading model. For each point on the surface, it is possible that some also receive the light from the reflection light of the other point in the same surface. For example, in some situation, the light reflected from the higher part, like the nose may arrive the face.

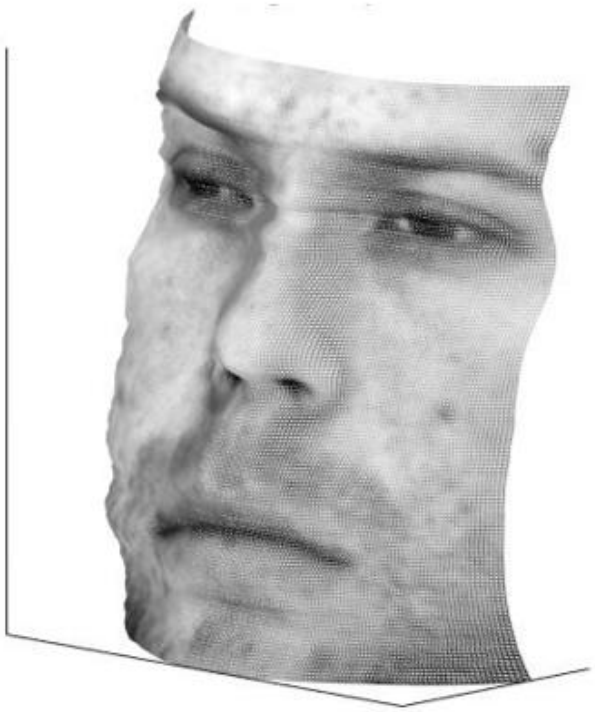
Third, source light direction is not exactly known. If the light source is a point light source, then, the light from a certain light source is not parallel(light the sunlight), but radioactive. Considering that the height of a certain point is known and need to be calculated, the source light direction is not exactly known. If we take the light from one light source is parallel, there will be some error in the final result.

Fourth, the camera moved a bit due to some mechanical reason. That problem is very considerable especially in history pictures, like mentioned in Homework 1.

Finally, we know that for a common photo, it cannot be taken as orthographic projection, and there is also some optics-caused-distortion. Since the distortion is more obvious in the edge and corner compared with in the central part of a photo, the data at the edge is not quite "reliable" and then the edge-relied-method like the column, row and average method(they need the integration of the first row or column) will suffer more from this.

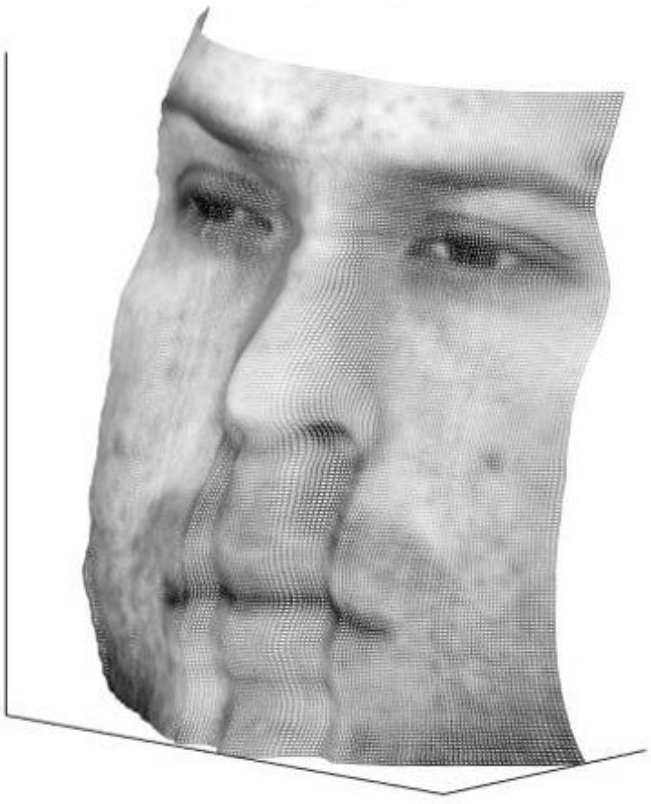
a Subject I choose: yaleB01

Column method:



Row method:

Average Method:



Random method:

4.2: compare the running times of the different approaches

Debug:

Column method: 0.005479s Row method: 0.004242s
Average Method: 0.004524s Random method: 3.780084s

yaleB01

Column method: 0.001402s Row method: 0.000800s

Average Method: 0.004520s Random method: 56.651472s

yaleB02:

Column method: 0.004539s Row method: 0.009374s
Average Method: 0.006696s Random method: 60.639696s

yaleB05:
Column method: 0.005643s Row method: 0.004946s
Average Method: 0.007646s Random method: 60.451948s

yaleB07:

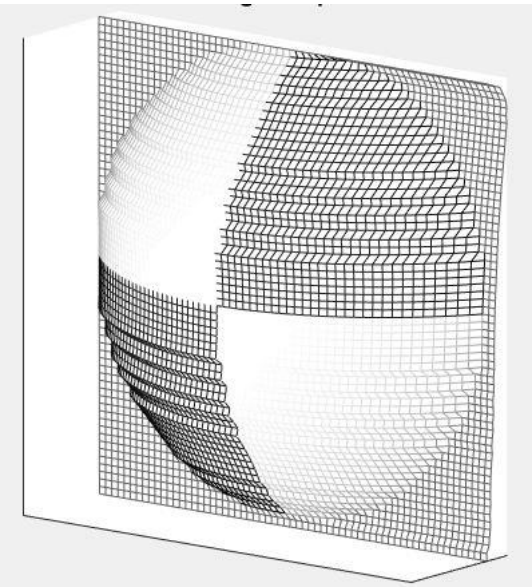
Column method: 0.006909s Row method: 0.006596s
Average Method: 0.005047s Random method: 60.570412s

The random method consume the most time. But the surface quality is the best.

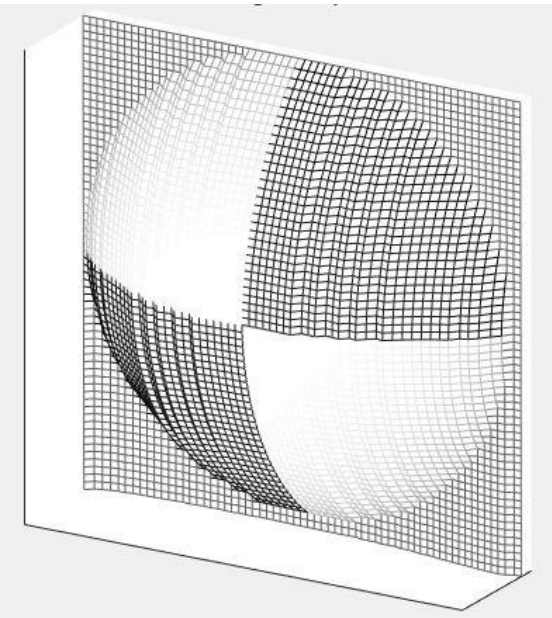
5. Images of albedo maps, surface maps and surface normal:

Subject: debug

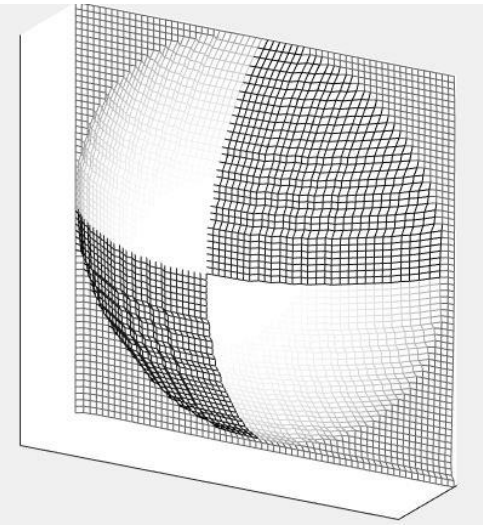
Column method: 0.005479s



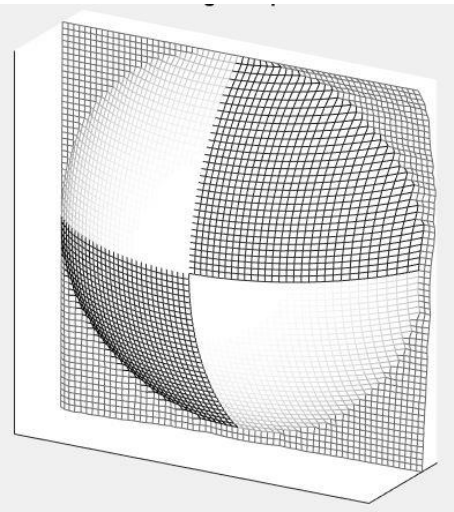
Row method: 0.004242s



Average Method: 0.004524s



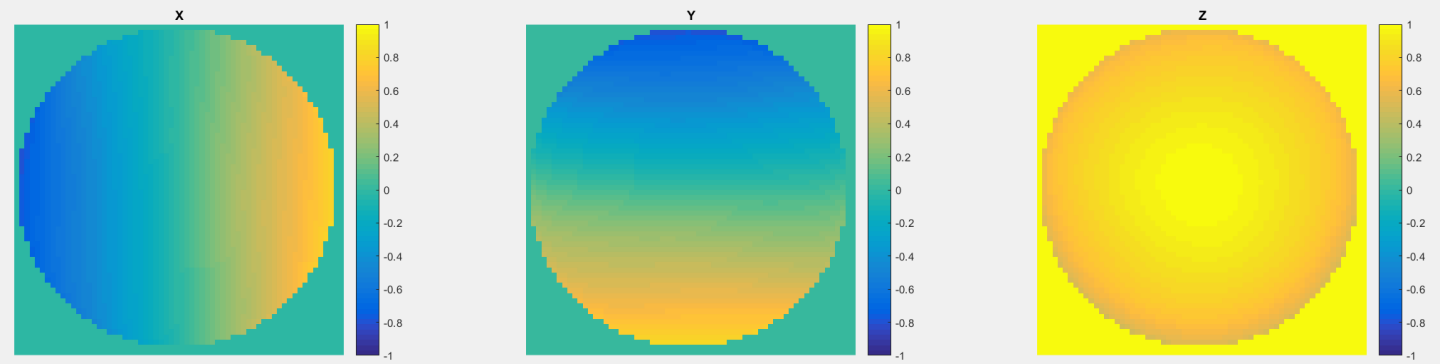
Random method: 3.780084s



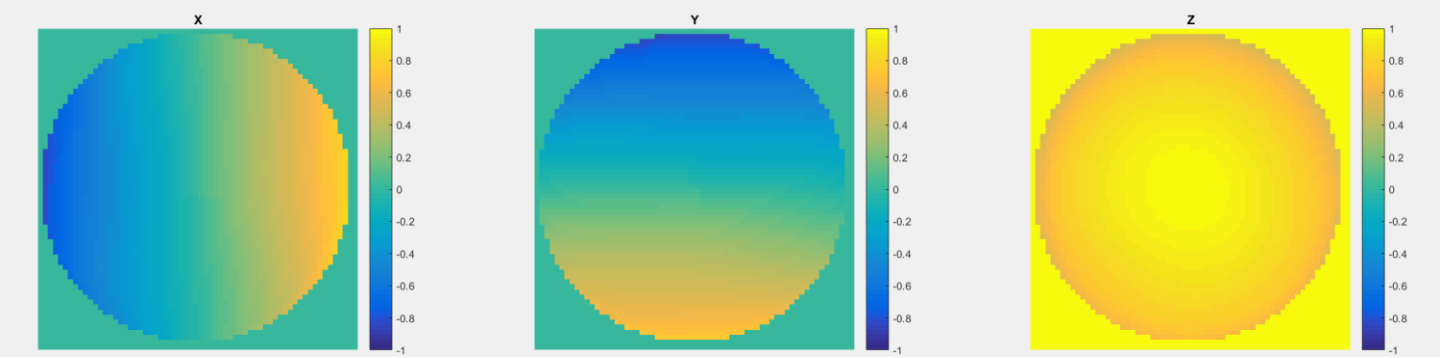
Albedo:

Comparison of the heightmap:

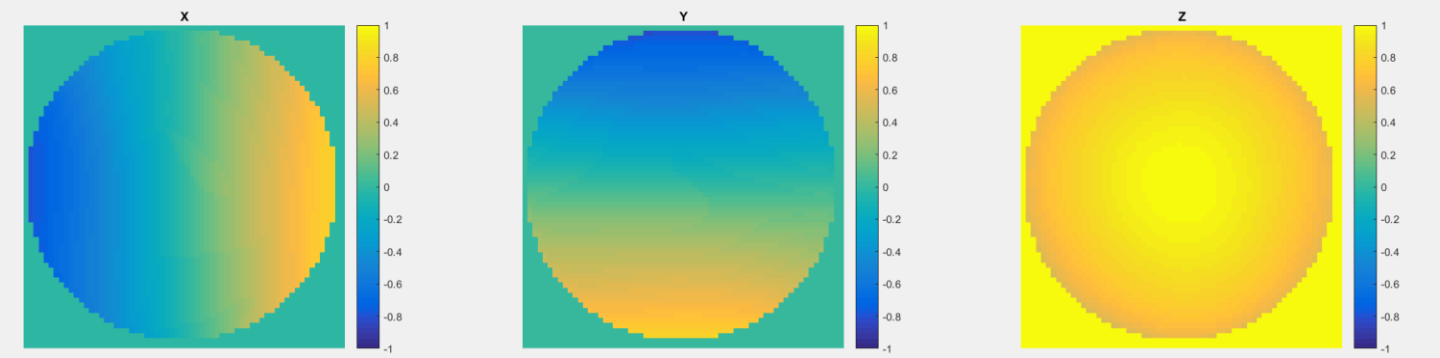
Column method:



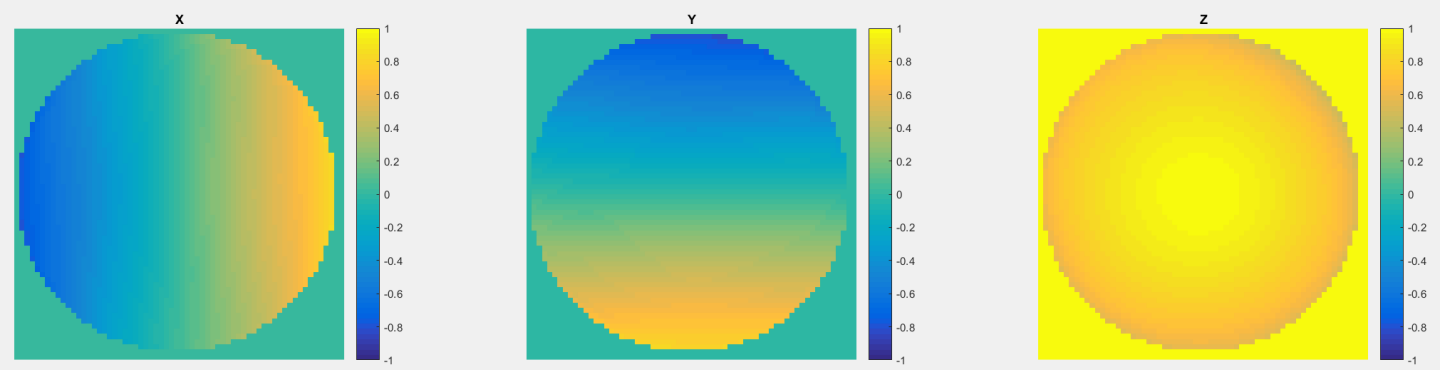
Row method:



Average method:

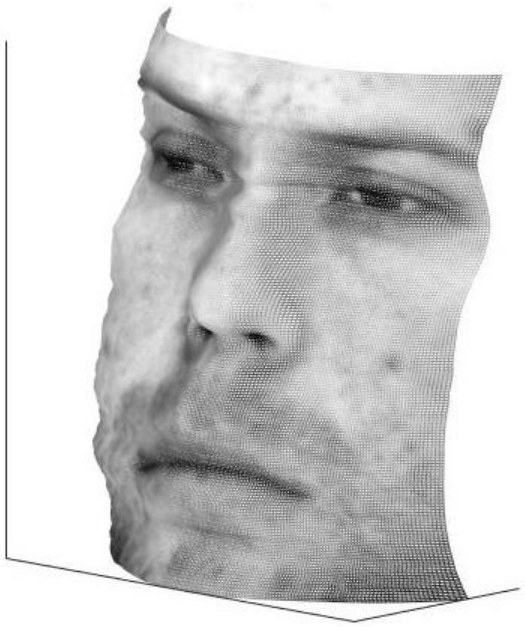


Random method:



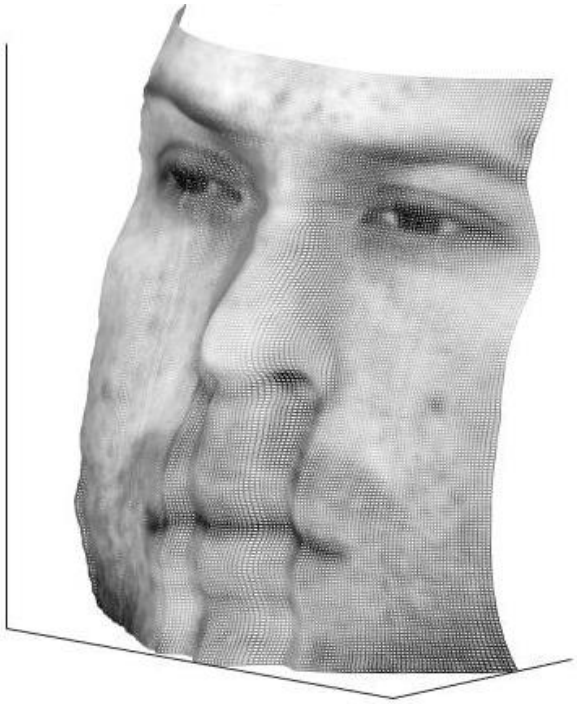
Subject: yaleB01

Column method: 0.001402s



Row method: 0.000800s

Average Method: 0.004520s

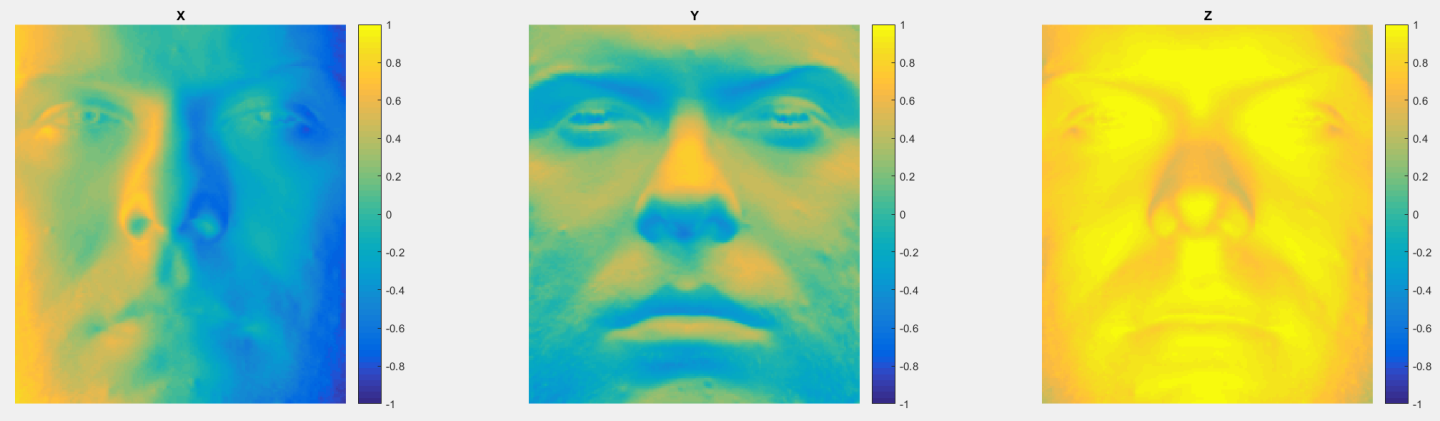


Random method: 56.651472s

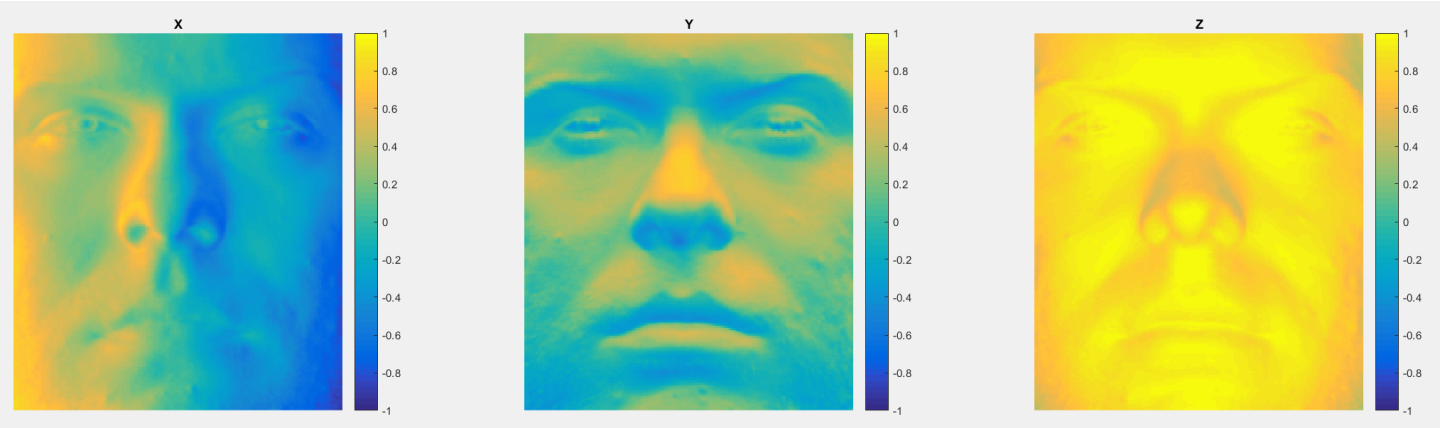
Albedo:

Comparison of the heightmap:

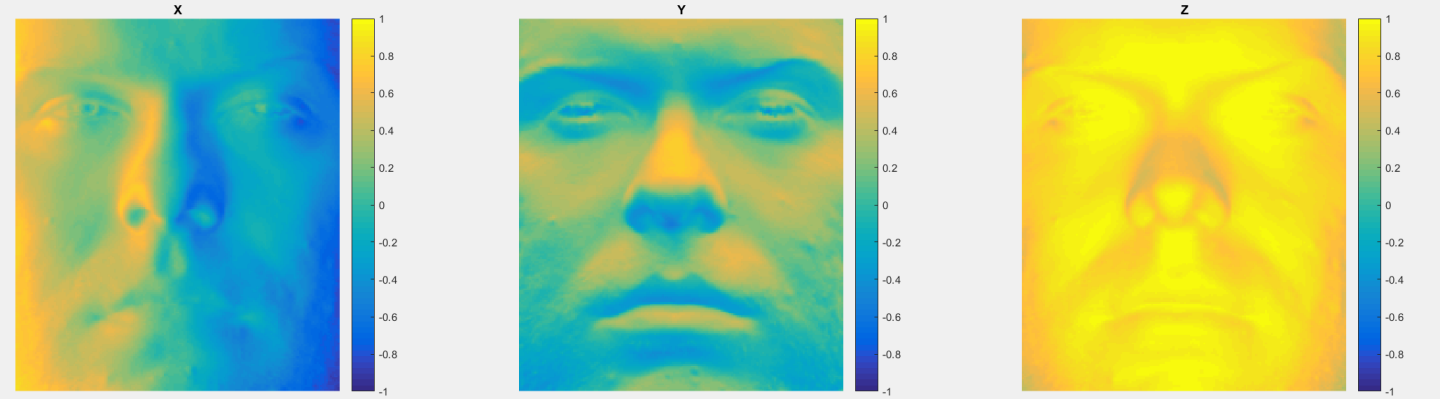
Column method:



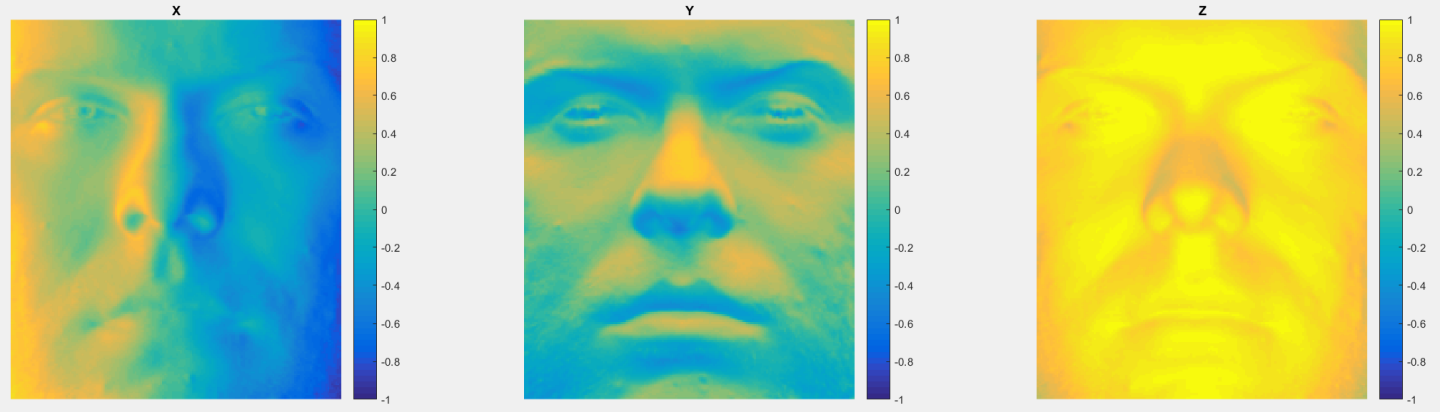
Row method:



Average method:

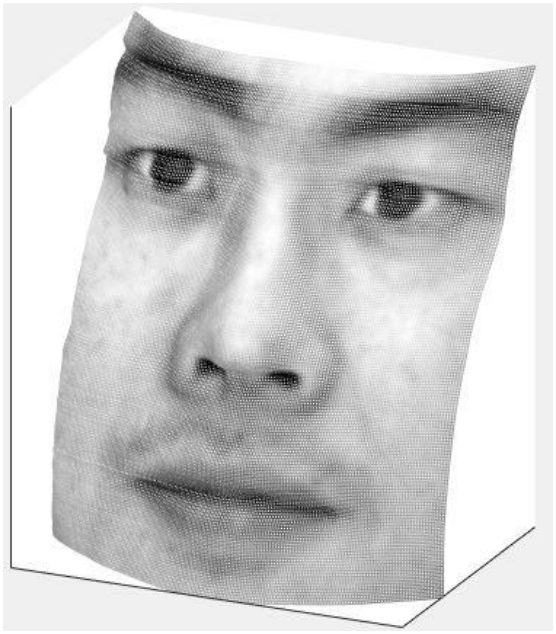


Random method:

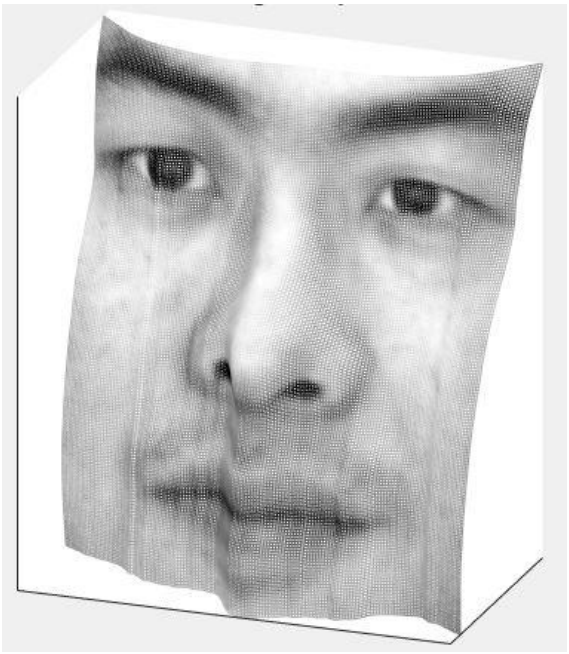


Subject: yaleB02

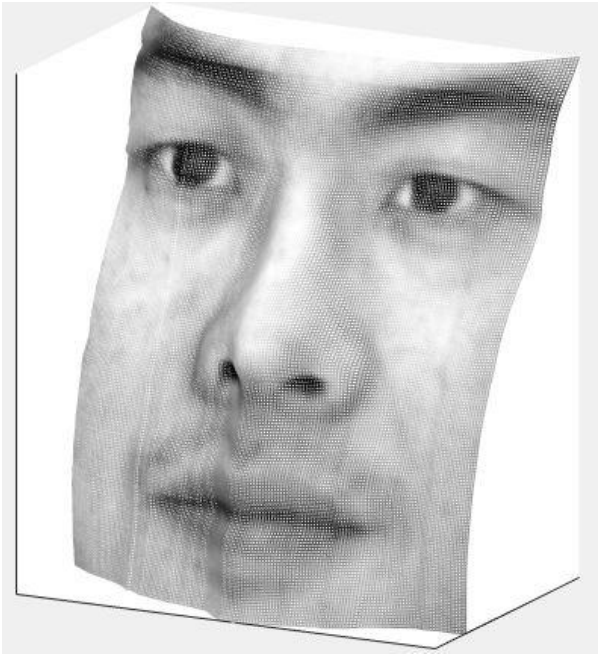
Column method: 0.004539s



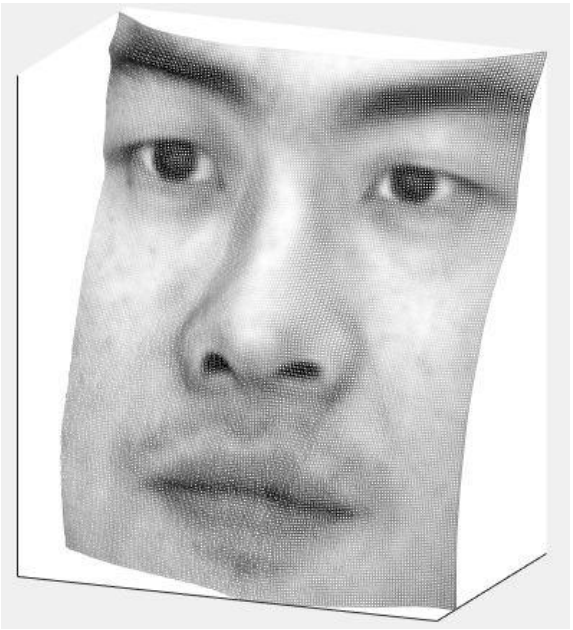
Row method: 0.009374s



Average Method: 0.006696s



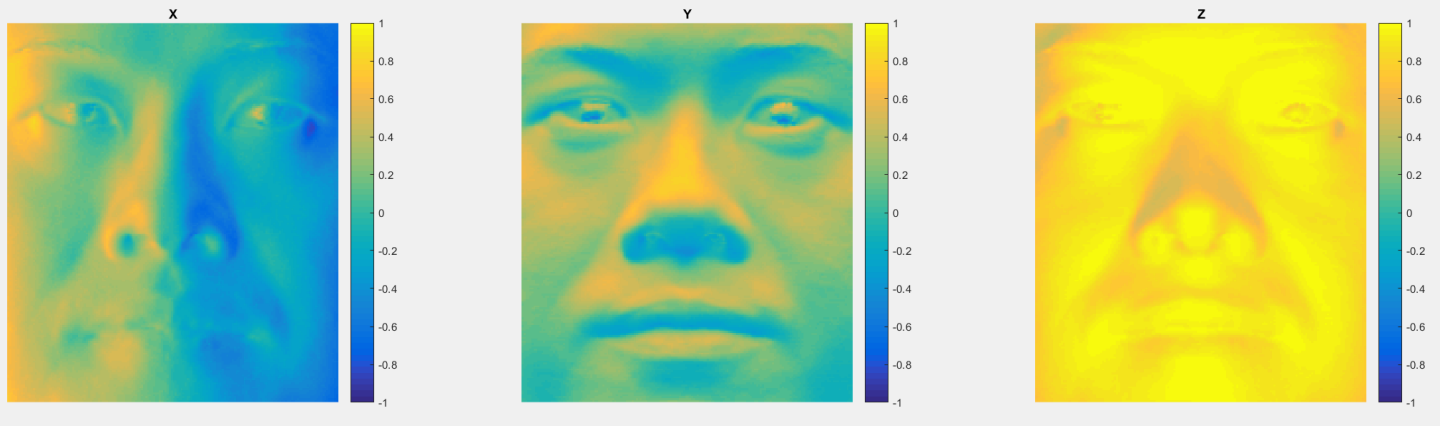
Random method: 60.639696s



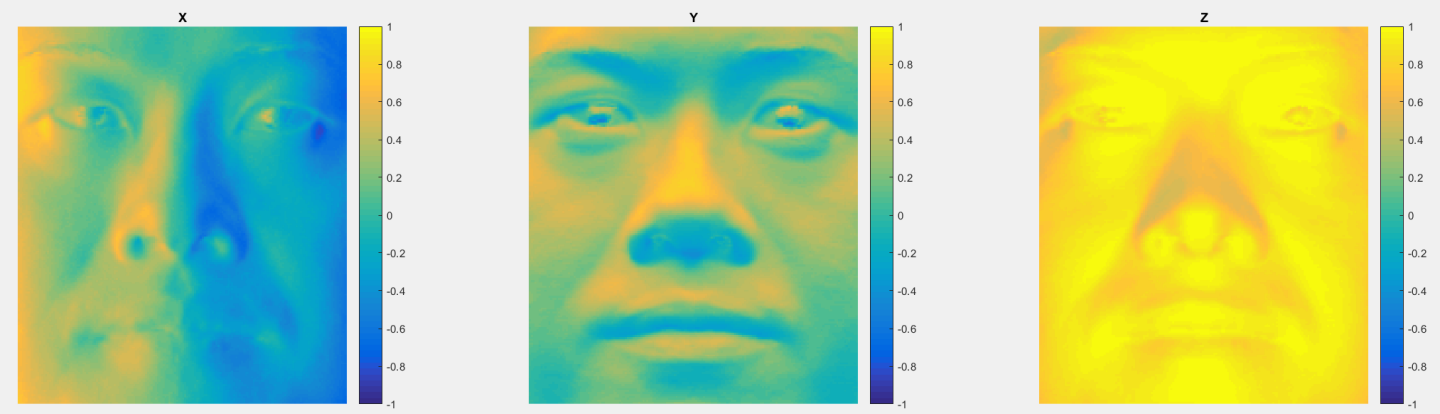
Albedo:

Comparison of the heightmap:

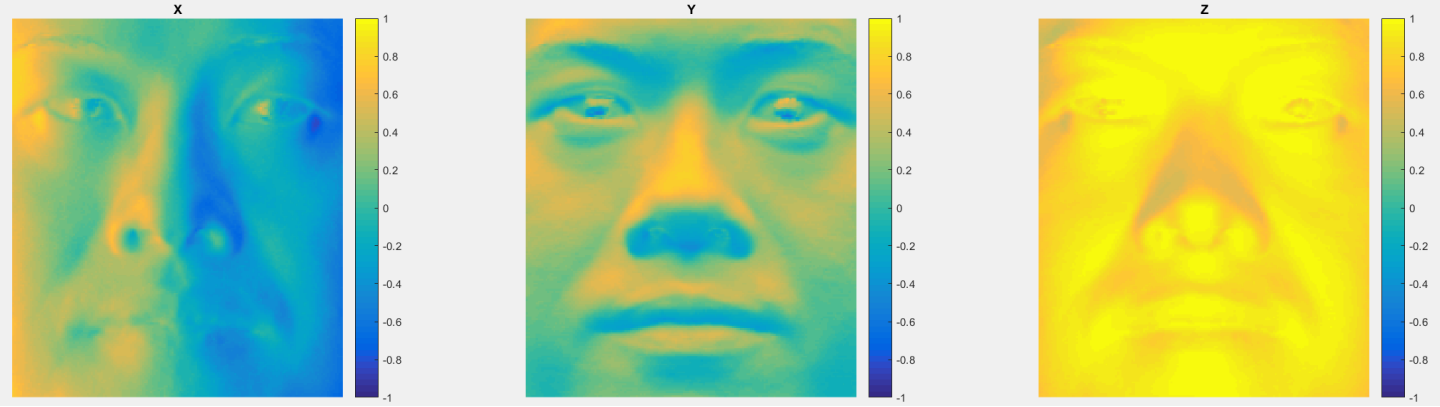
Column method:



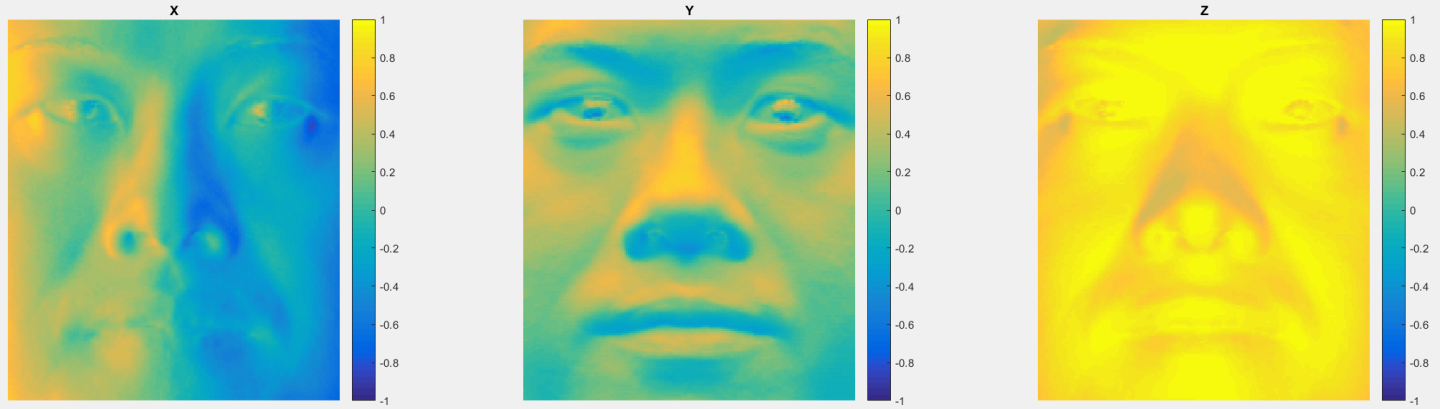
Row method:



Average method:



Random method:

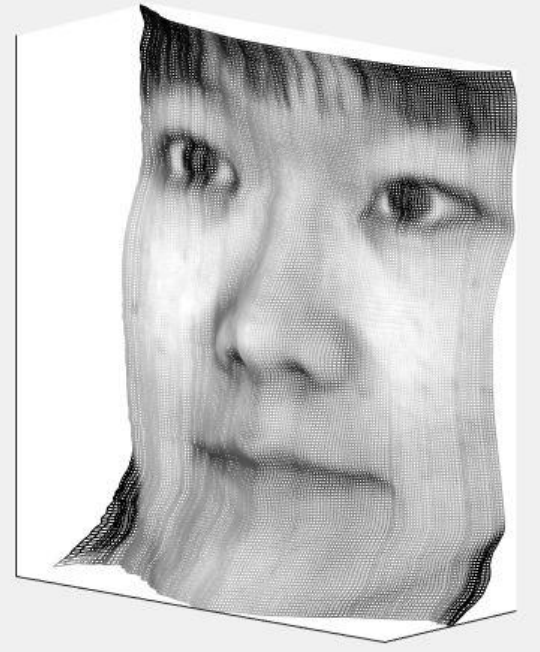


Subject: yaleB05

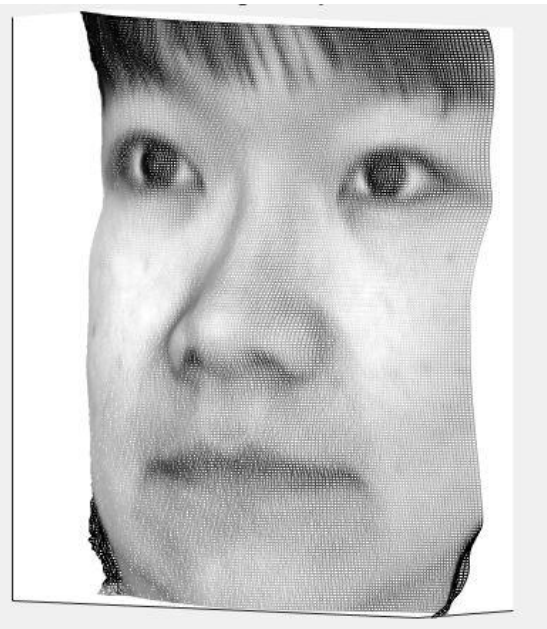
Column method: 0.005643s

Row method: 0.004946s

Average Method: 0.007646s



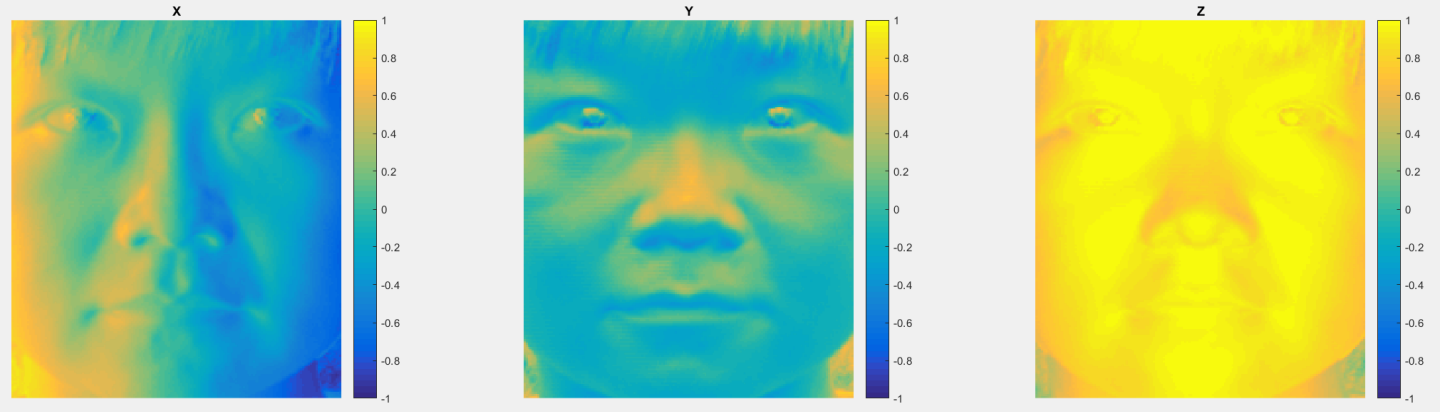
Random method: 60.451948s



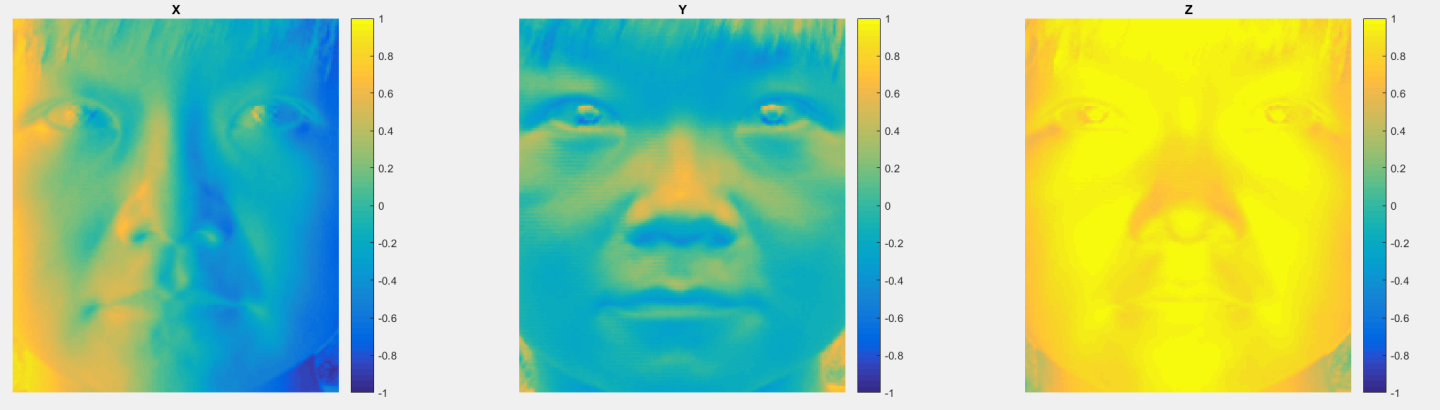
Albedo:

Comparison of the heightmap:

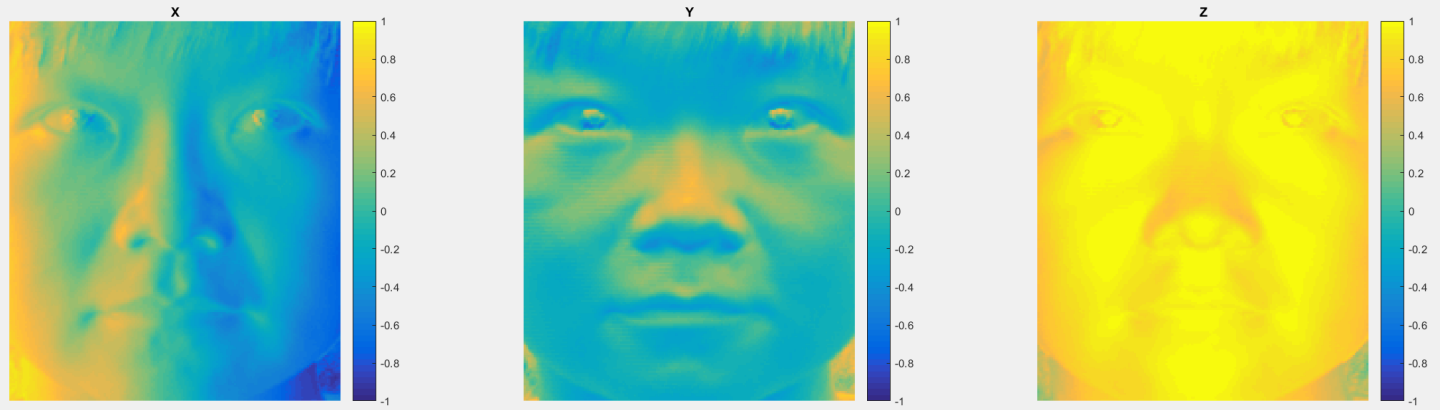
Column method:



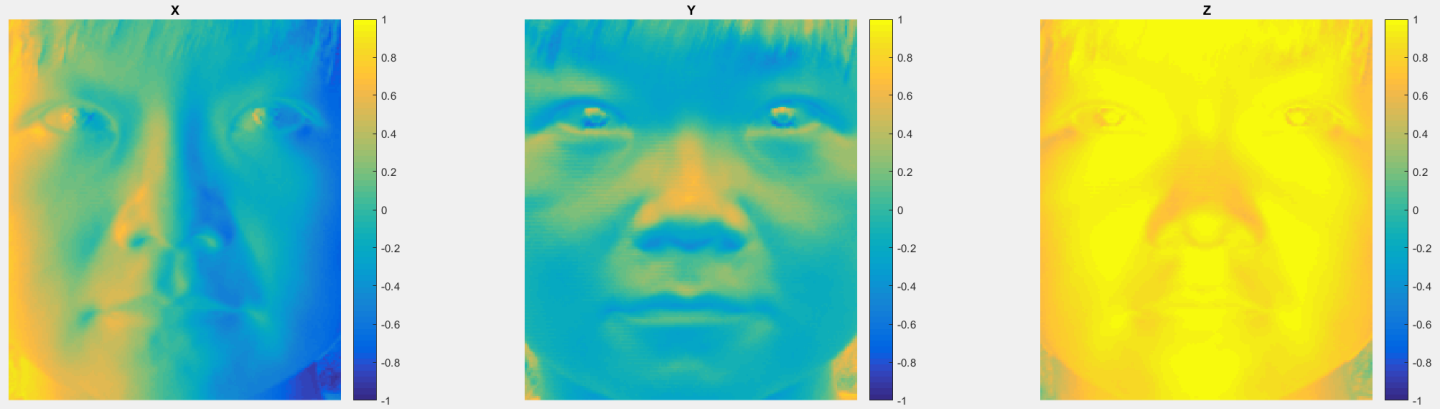
Row method:



Average method:

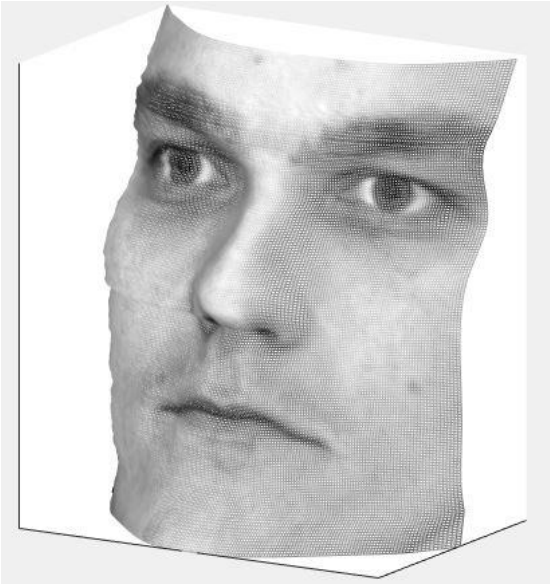


Random method:

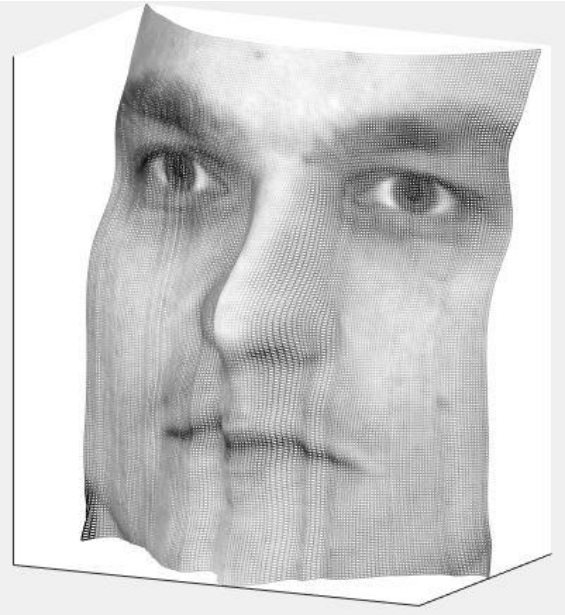


Subject: yaleB07

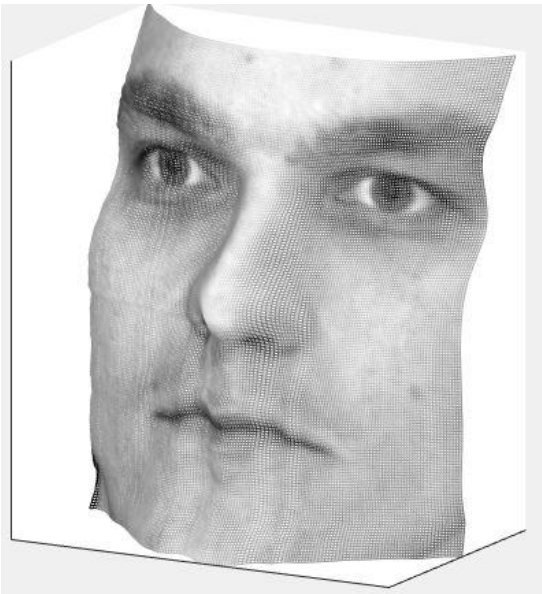
Column method: 0.006909s



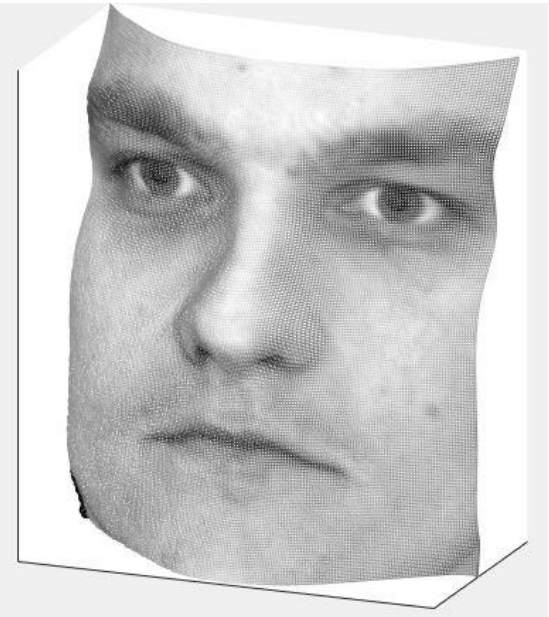
Row method: 0.006596s



Average Method: 0.005047s



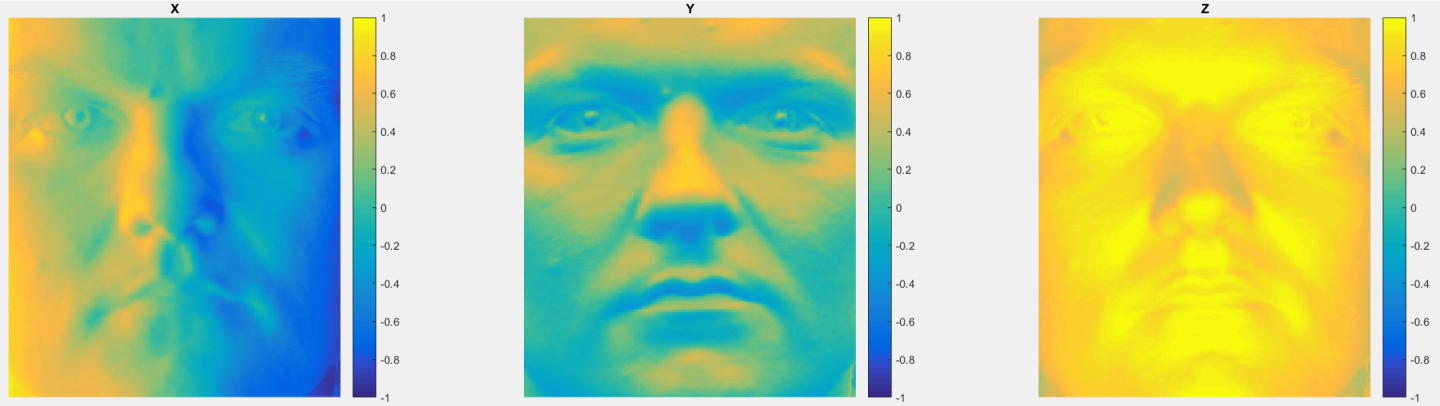
Random method: 60.570412s



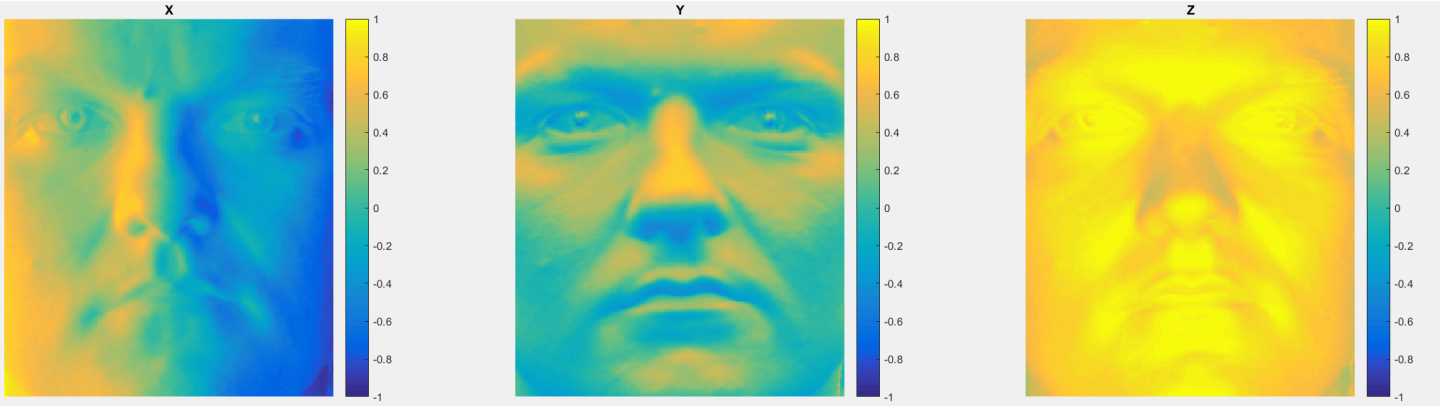
Albedo:

Comparison of the heightmap:

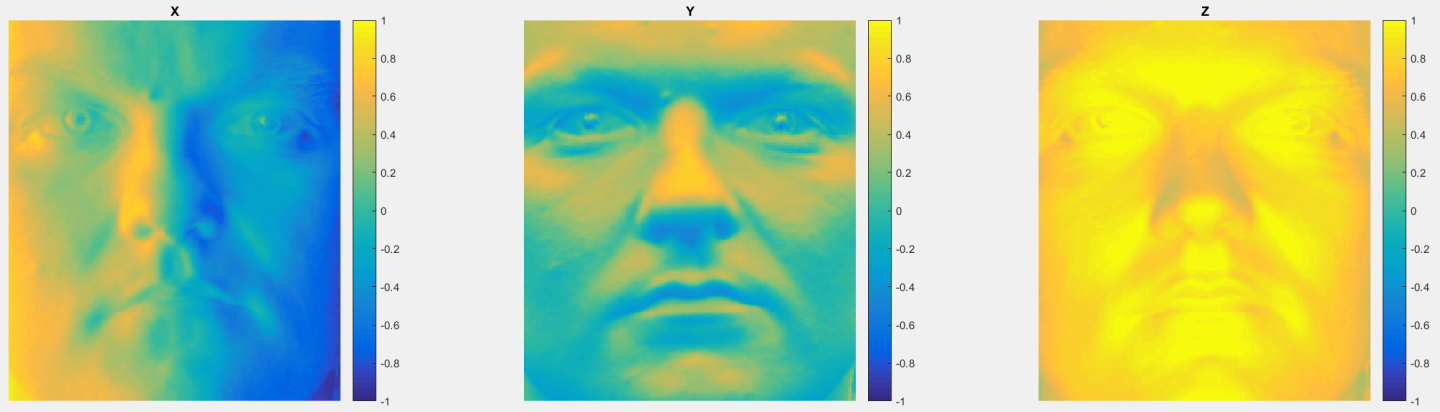
Column method:



Row method:



Average method:



Random method: