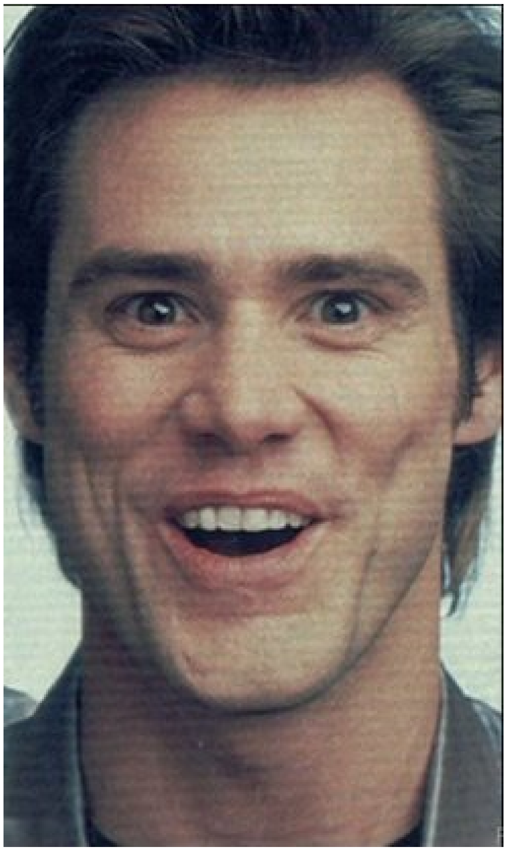
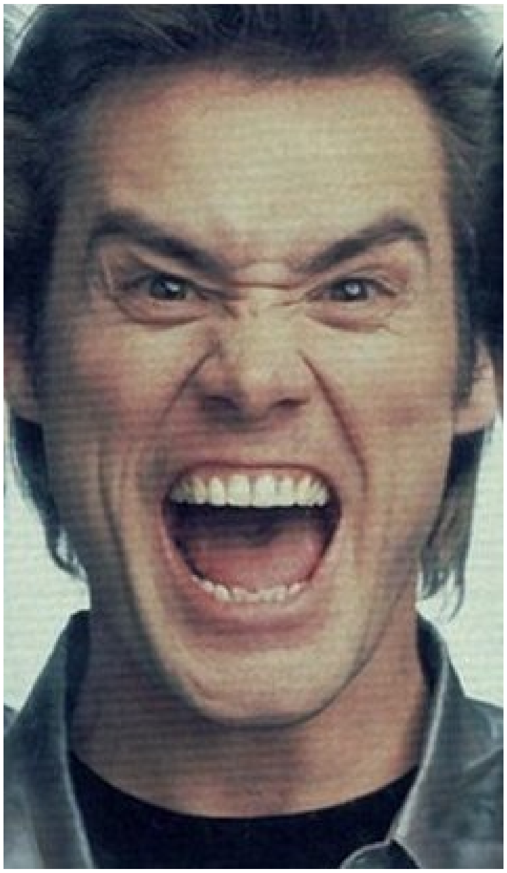
Computer Vision *Name:Abhay Doke UID:29552668*

Mini Project 02

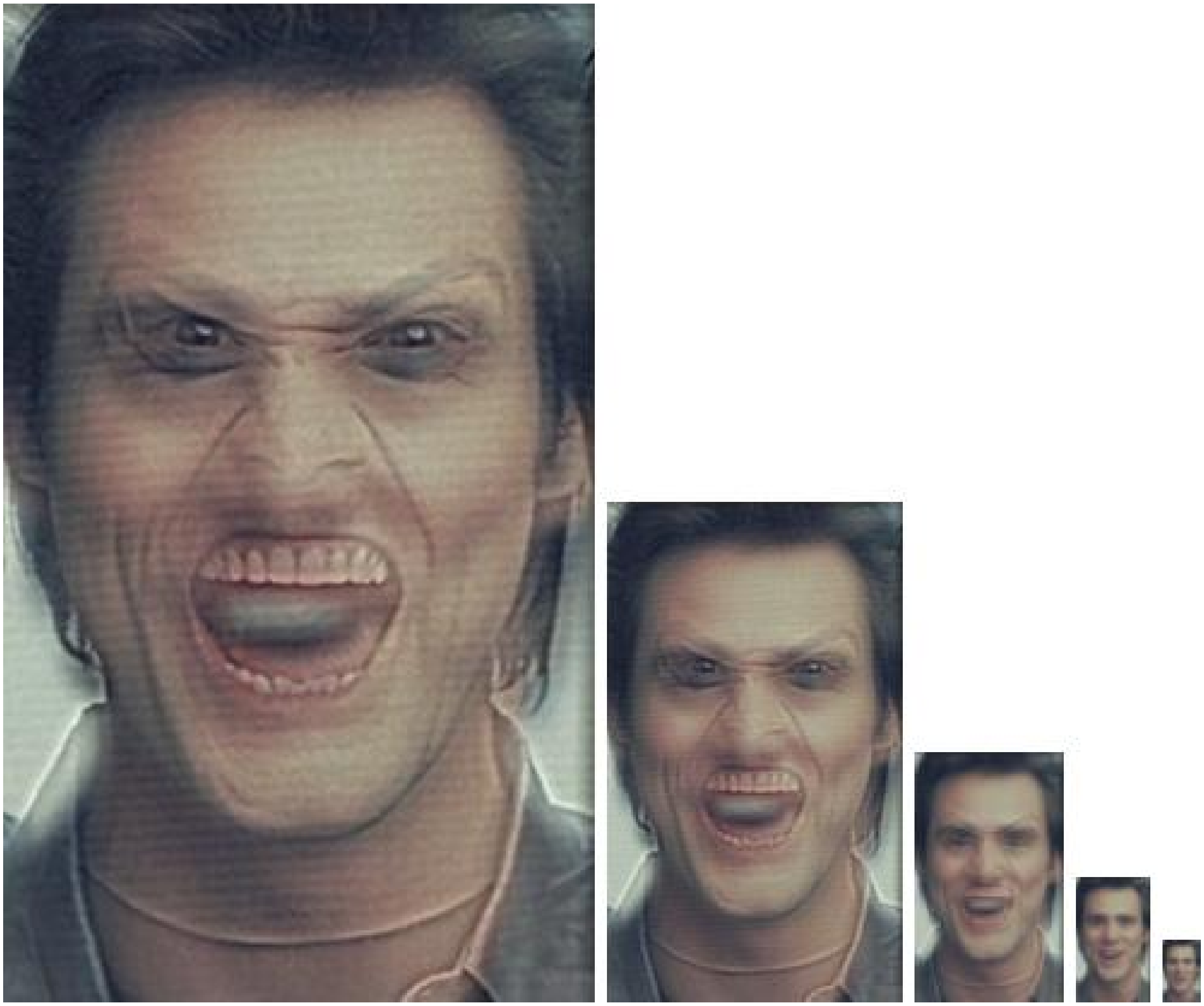
1.Hybrid Images

Images Used for hybrid images:





Output Hybrid image:



For the Blurring, I used Sigma = 7 and for the Sharpening, Sigma = 5.

Code for making Hybrid Images:

```
function hybridImage = hybridImage( im1, im2, sig1, sig2 )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

im1 = double(im1);
im2 = double(im2);
filter1 = fspecial('Gaussian', sig1*2+1, sig1);
filter2 = fspecial('Gaussian', sig2*2+1, sig2);

blurred = imfilter(im1, filter1, 'replicate');
sharpened =  (im2 - imfilter(im2, filter2, 'replicate'));
hybridImage = blurred + sharpened;
%imshow(vis_hybrid_image(mat2gray(hybridImage(j2,j3,7,5)))) Best result
end
```

2. Photometric stereo

Code for **prepareData.m**

```
function output = prepareData(imArray, ambientImage)
% PREPAREDATA prepares the images for photometric stereo
%   OUTPUT = PREPAREDATA(IMARRAY, AMBIENTIMAGE)
%
%   Input:
%       IMARRAY - [h w n] image array
%       AMBIENTIMAGE - [h w] image
%
%   Output:
%       OUTPUT - [h w n] image, suitably processed
%
% Author: Subhransu Maji
%

% Implement this %
% Step 1. Subtract the ambientImage from each image in imArray

output = zeros(size(imArray));
[h,w,n] = size(imArray)
for i = 1:n
    img = imArray(:,:,i);
    img = img - ambientImage;
    % Step 2. Make sure no pixel is less than zero
    img(img<0) = 0;
    % Step 3. Rescale the values in imarray to be between 0 and 1
    img = img/255;

    output(:,:,i) = img;
end
```

Code for **photometricStereo.m**

```
function [albedoImage, surfaceNormals] = photometricStereo(imArray, lightDirs)
% PHOTOMETRICSTEREO compute intrinsic image decomposition from images
%   [ALBEDOIMAGE, SURFACENORMALS] = PHOTOMETRICSTEREO(IMARRAY, LIGHTDIRS)
%   comptutes the ALBEDOIMAGE and SURFACENORMALS from an array of images
%   with their lighting directions. The surface is assumed to be perfectly
%   lambertian so that the measured intensity is proportional to the albedo
%   times the dot product between the surface normal and lighting
%   direction. The lights are assumed to be of unit intensity.
%
%   Input:
%       IMARRAY - [h w n] array of images, i.e., n images of size [h w]
%       LIGHTDIRS - [n 3] array of unit normals for the light directions
%
```

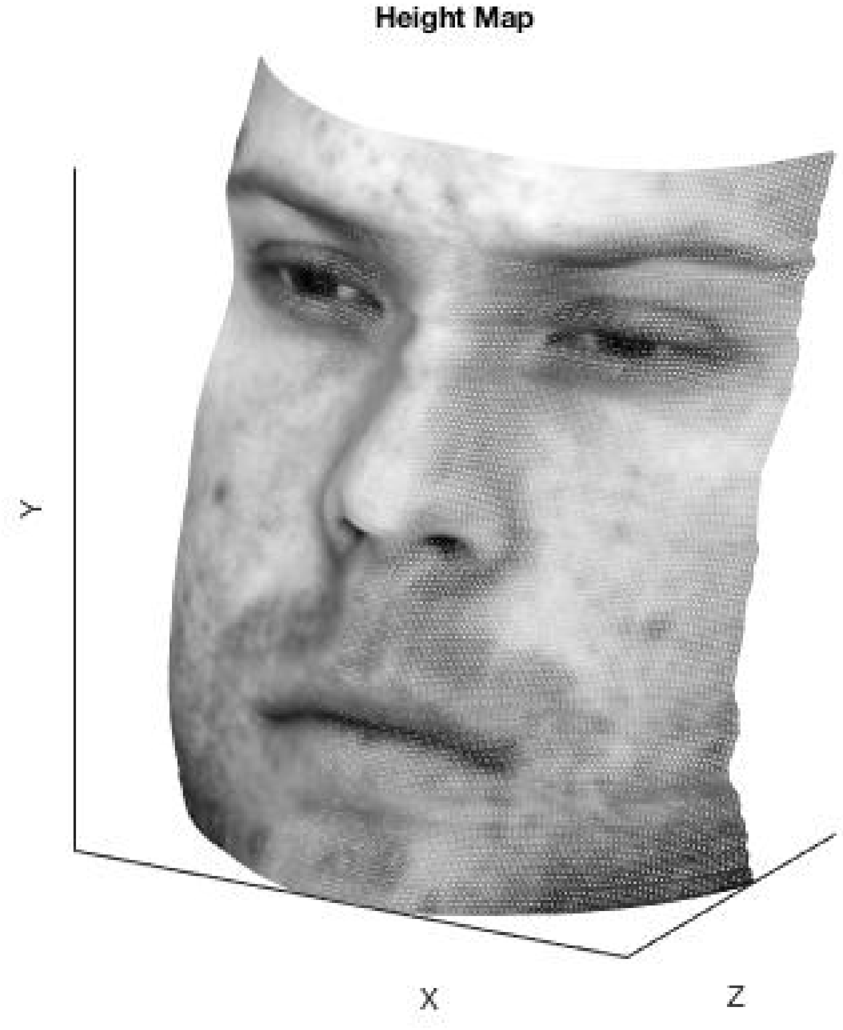
```
%   Output:
%        ALBEDOIMAGE - [h w] image specifying albedos
%        SURFACENORMALS - [h w 3] array of unit normals for each pixel
%
% Author: Subhransu Maji
%
% Acknowledgement: Based on a similar homework by Lana Lazebnik

[h, w, n] = size(imArray);
albedoImage = ones(h,w);

for i = 1:h
    for j = 1:w
        pixelVals = reshape(imArray(i,j,:),[n,1]);

        %disp(size(pixelVals));
        %disp(’lightDirs’);
        % disp(size(lightDirs));

        gox = lightDirs\pixelVals;

        %disp(’gox’);
        %disp(size(gox));

        magnitude = norm(gox);
        albedoImage(i,j) = magnitude;
        gox = gox./magnitude;
        surfaceNormals(i,j,1) = gox(1);
        surfaceNormals(i,j,2) = gox(2);
        surfaceNormals(i,j,3) = gox(3);
    end
end
```

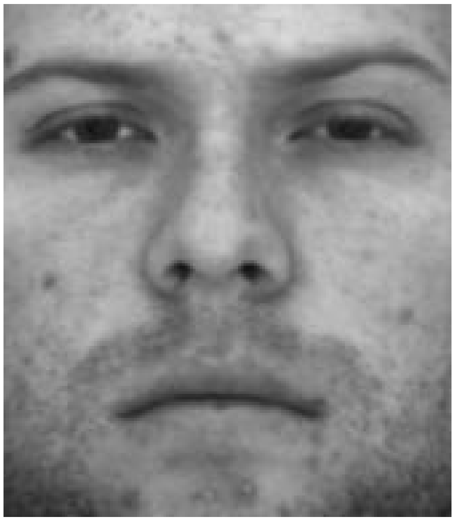
Code for **getSurface.m**

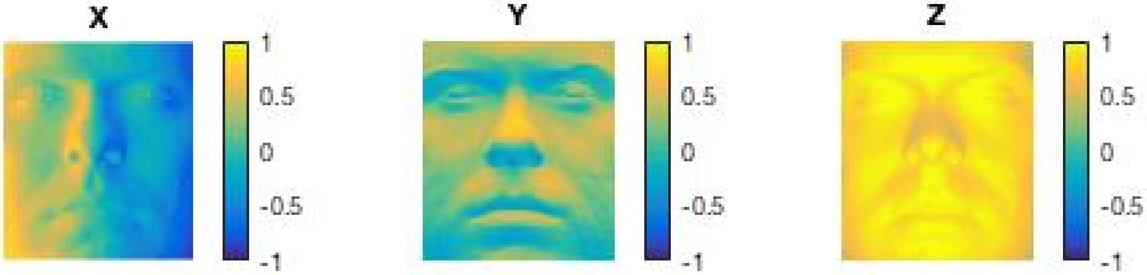
```
function  heightMap = getSurface(surfaceNormals, method)
% GETSURFACE computes the surface depth from normals
%   HEIGHTMAP = GETSURFACE(SURFACENORMALS, IMAGESIZE, METHOD) computes
%   HEIGHTMAP from the SURFACENORMALS using various METHODs.
%
% Input:
%   SURFACENORMALS: height x width x 3 array of unit surface normals
%   METHOD: the intergration method to be used
%
% Output:
%   HEIGHTMAP: height map of object
[h,w,n] = size(surfaceNormals);
heightMap = zeros([h w]);
gxy = zeros([h w]);
gxx = zeros([h w]);

gxx = surfaceNormals(:,:,2)./surfaceNormals(:,:,3);
gxy = surfaceNormals(:,:,1)./surfaceNormals(:,:,3);
```

```
switch method
    case 'column'
        %% implement this %%%
        for i = 1:h
            for j = 1:w
                sum = 0;
                for x = 1:i
                    sum = sum + gxx(x,1);
                end
                for y = 2:j
                    sum = sum+gxy(i,y);
                end

                heightMap(i,j) = sum;
            end
         end

    case 'row'
        %%% implement this %%%
        for i = 1:h
            for j = 1:w
                sum = 0;
                for y = 1:j
                    sum = sum + gxy(1,y);
                end
                for x = 2:i
                    sum = sum + gxx(x,j);
                end
                heightMap(i,j) = sum;
            end
         end

    case 'average'
        %%% implement this %%%
        for i = 1:h
            for j = 1:w
                sum1 = 0;
                for y = 1:j
                    sum1 = sum1 + gxy(1,y);
                end
                for x = 2:i
                    sum1 = sum1 + gxx(x,j);
                end


                sum2 = 0;
                for x = 1:i
                    sum2 = sum2 + gxx(x,1);
                end
                for y = 2:j
                    sum2 = sum2 + gxy(i,y);
                end

                heightMap(i,j) = (sum1+sum2)/2.0;
            end
        end


    case 'random'
        %%% implement this %%%
        for i = 1:h
            for j = 1:w
                x=1;
                y=1;
                sum = 0;
                while(x<i & y<j)
                    sum = sum+gxx(x,y);

                    y=y+1;
                    if y<j
                        sum=sum+gxy(x,y);

                    else
                        break;
                    end
                    x=x+1;
                    if x<i
                        sum=sum+gxx(x,y);

                    else
                    end
                end
                if y==j & x<i
                    for a=x:i
                        sum = sum+gxx(a,y);

                    end
                elseif x==i & y<j
                    for a=y:j
                        sum=sum+gxy(x,a);

                    end
                else
                    sum = sum+gxx(x,y);

                end

                heightMap(i,j)=sum;
            end
        end
end
```

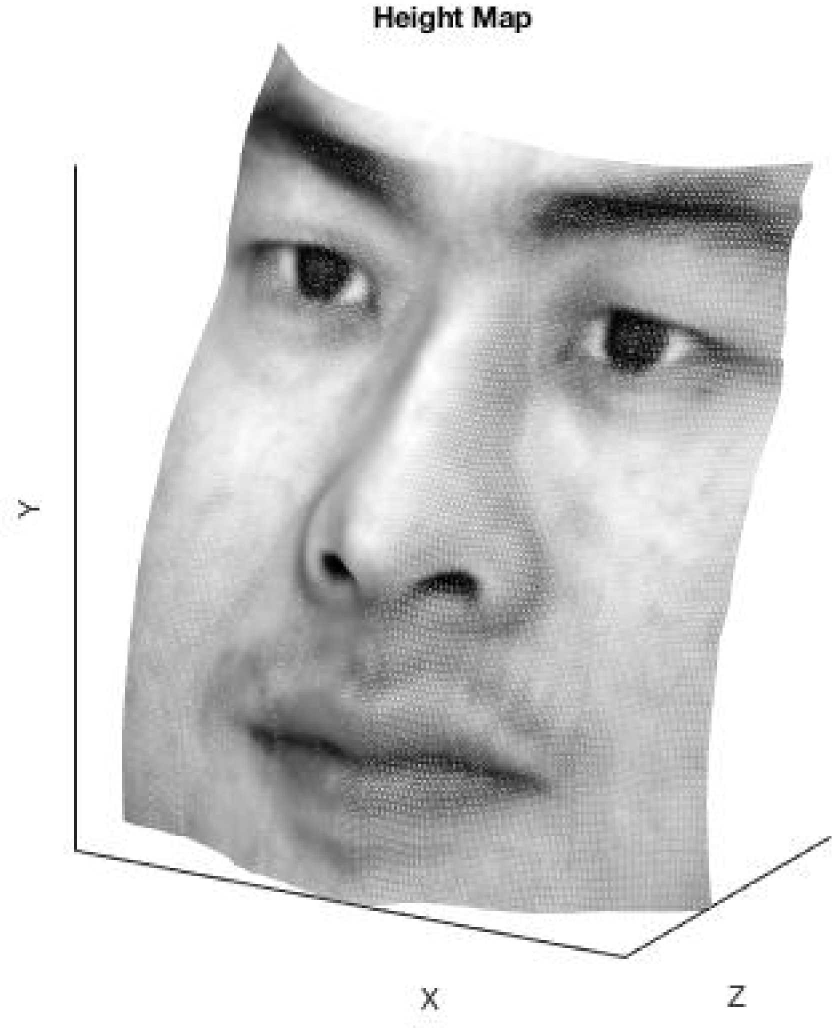
Results for YaleB01:

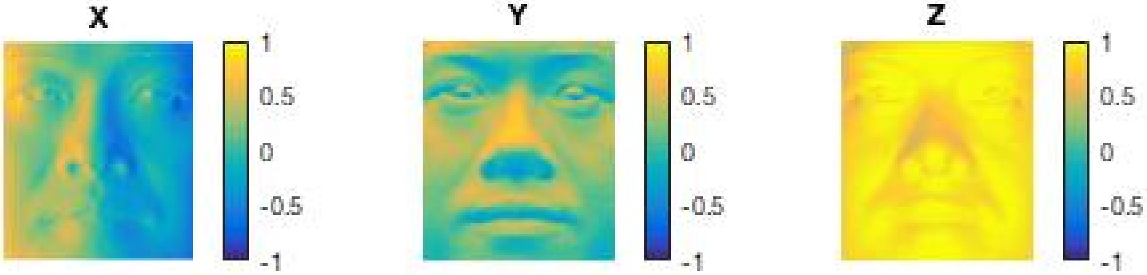




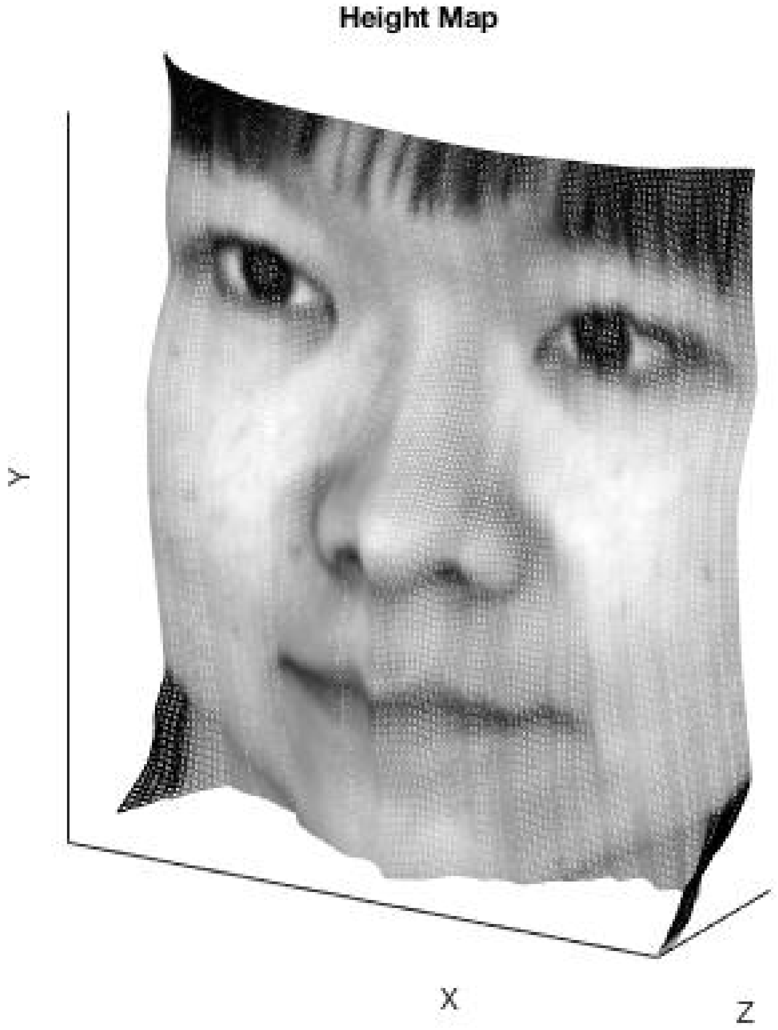

X
1
0.5
0
-0.5
-1
Y
1
0.5
0
-0.5
-1
Z
1
0.5
0
-0.5
-1

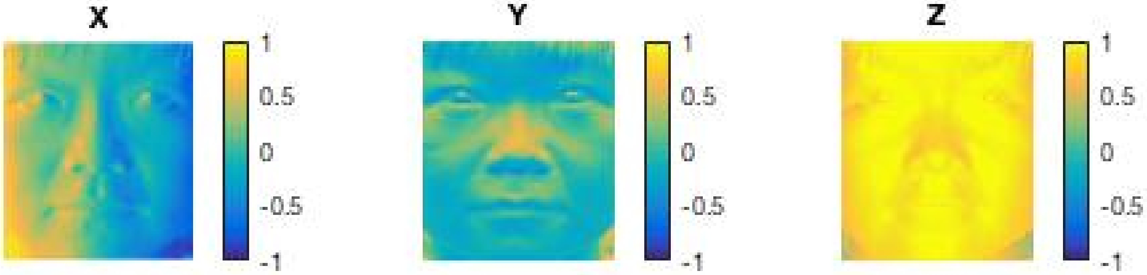
Results for YaleB02:



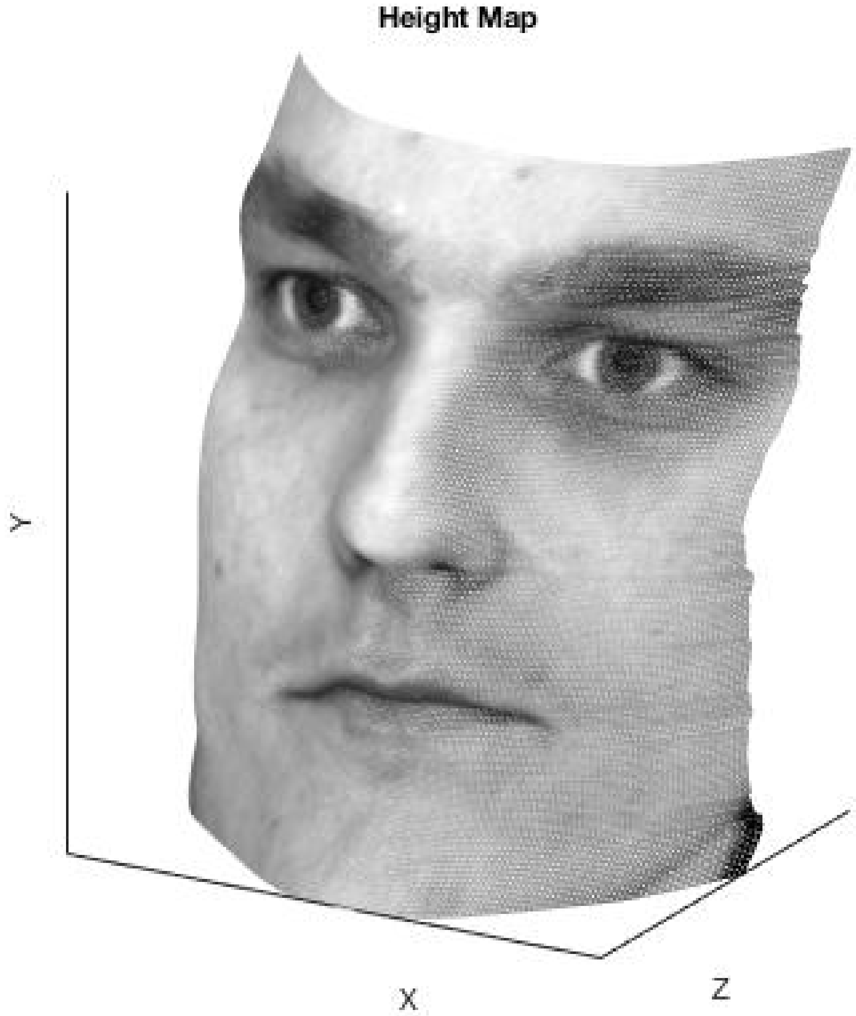

X
1
0.5
0
-0.5
-1
Y
1
0.5
0
-0.5
-1
Z
1
0.5
0
-0.5
-1

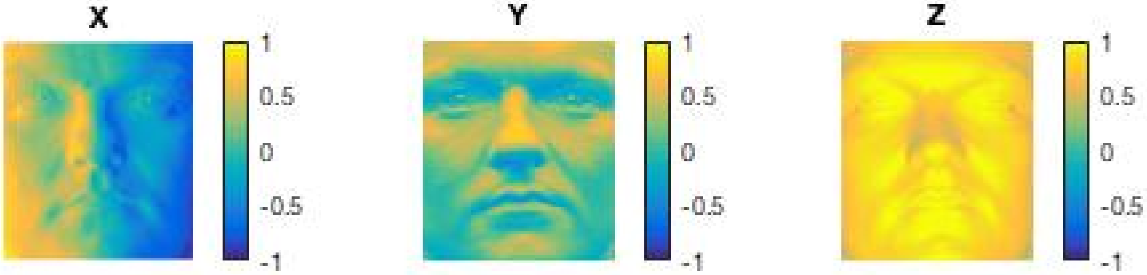
Results for YaleB05:



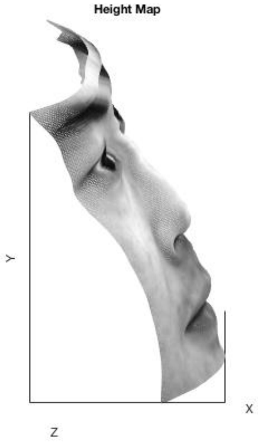

X
Y
Z
1
0.5
0
-0.5
-1

Results for YaleB07:



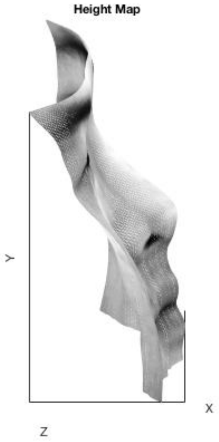

X
1
0.5
0
-0.5
-1
Y
1
0.5
0
-0.5
-1
Z
1
0.5
0
-0.5
-1

Difference between the integration methods for the yaleB02 subject.



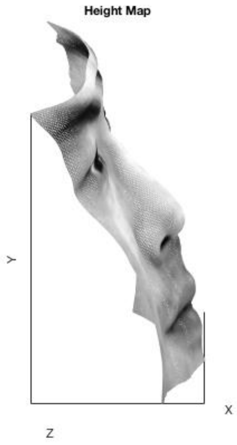
copy.jpg copy.jpg

Figure 1: Integrating Columns and then Rows



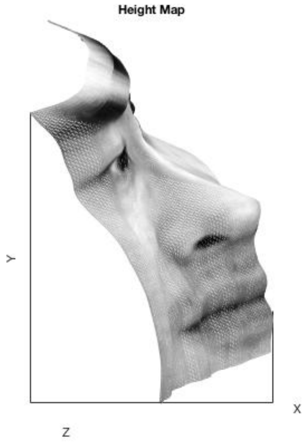
copy.jpg copy.jpg

Figure 2: Integrating Rows and then Columns



copy.jpg copy.jpg

Figure 3: Average method



copy.jpg copy.jpg

Figure 4: Random Method

For the yaleB02 the average method works the best and the random method performs the worst. Integrating with the column method makes the surface narrowed while integrating with the row method makes it more widened. These effects can be resolved by using the average method.

Yale face data has shadow areas which violates assumptions of the shape-from shading method. Points which are in shadow affects the computation of the albedo image which in turns affects the computation of g(x, y). This leads to false albedo points and also the surface depth gets affected and it becomes uneven.