PHOTPMETRIC STERO

CHARU HANS

Problem Description

A. 3d reconstruction of object using a series of images taken under varying light direction. The vectors representing lighting direction can be estimated using the chrome ball method, from the lighting direction one can approximate surface normals for the object, and from these generate a surface map. Albedos and RGB-encoded normals were generated for all of the provided test objects.
B. Apply (1) by removing specularity component from given dataset.

Part – A

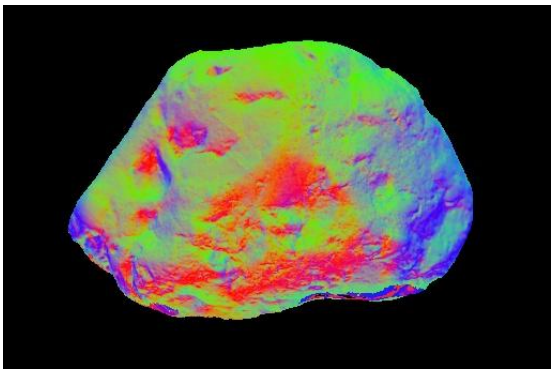
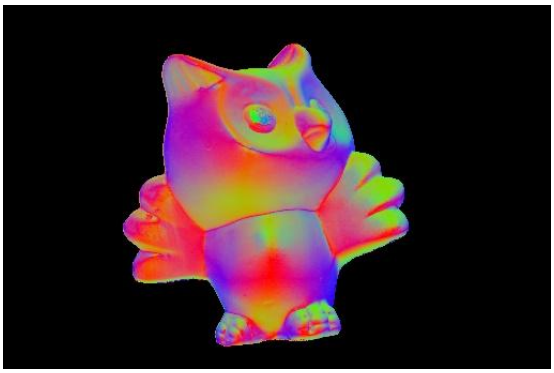
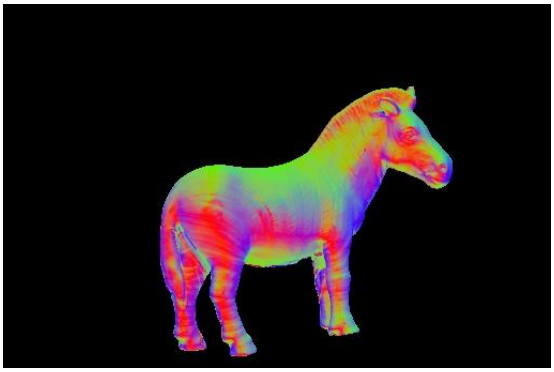
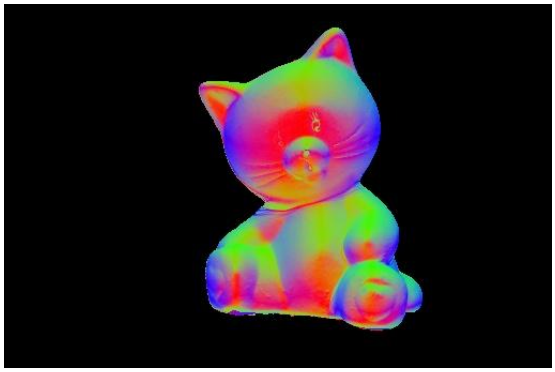
1. Calibration
   a. Compute the centroid of the chrome ball mask as the center of the ball, [Cx,Cy]
   b. Compute the average distances between the boundary pixels to the center as the radius
   c. Compute the centroid of highlight pixels, [Bx,By]
   d. Normal at any given point on its surface:
      i. Nx = Cx – Bx
      ii. Ny = Cy - By
      iii. Ny = sqrt(R^2 – Nx^2 - Ny^2)
   e. Compute the vector direction of the light with the equation L = 2*dot(N,R)*N – R

Result

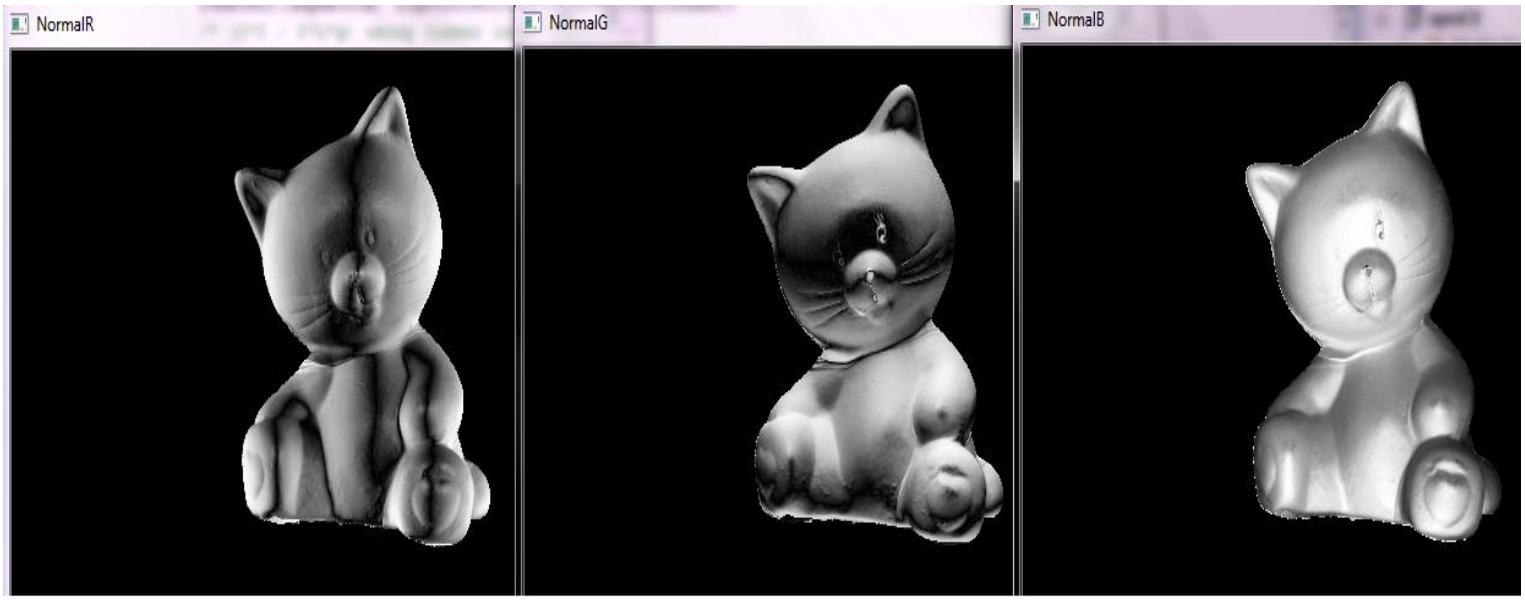
0.504132,	**0.464747,**	**0.727916**
0.251585,	**0.134179,**	**0.958489**
-0.0337589,	**0.177234,**	**0.98359**
-0.0906432,	**0.444975,**	**0.890944**
-0.321589,	**0.506503,**	**0.800022**
-0.105037,	**0.565581,**	**0.817976**
0.285667,	**0.432581,**	**0.855142**
0.10729,	**0.429162,**	**0.896833**

2. Normal Map: The intensity at each pixel should be the inner product of the lighting direction L and the surface normal n, times the albedo kd , **I=kd(L.n)**

   To simplify, we say that kdn = g. To find the surface normal at each pixel, we solve a least squares solution, by minimizing, **Q=sum(Ii - Lig')^2**

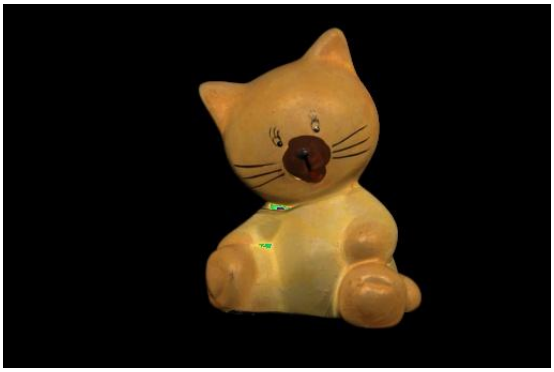
Above images shows, RGB Surface normals and below images shows surface normal in channel R, G and B.



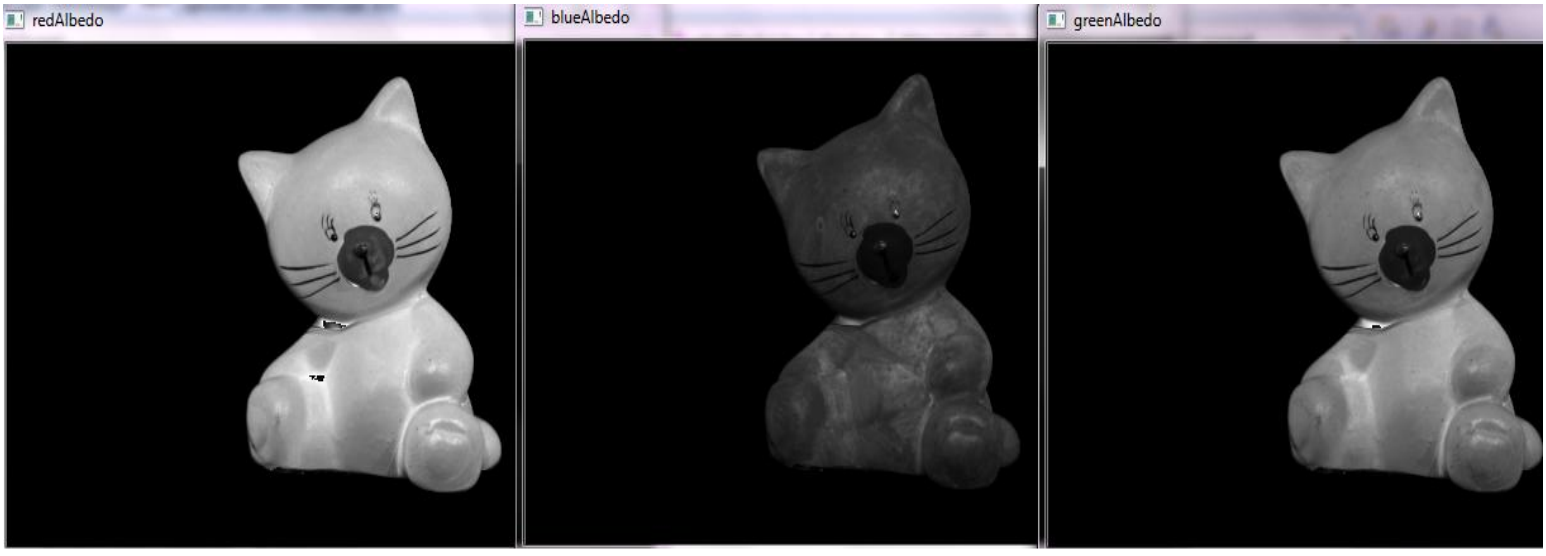
3. Albedo: Once we have the normals at each pixel, the albedo is calc b following equations:

$$k_d = \sum_i I_i \mathbf{L}_i \mathbf{n}^T / \sum_i (\mathbf{L}_i \mathbf{n}^T)^2 \qquad k_d = I \cdot J / J \cdot J$$

So now, if J=Li.n, then the albedo at each pixel is simply the inner product I and J normalized by J.J, sum(Ii.Ji)/sum(Ji.Ji), i = lighting source directions.



Above images shows, Albedo mapls and below images shows Albedo map in channel R, G and B.

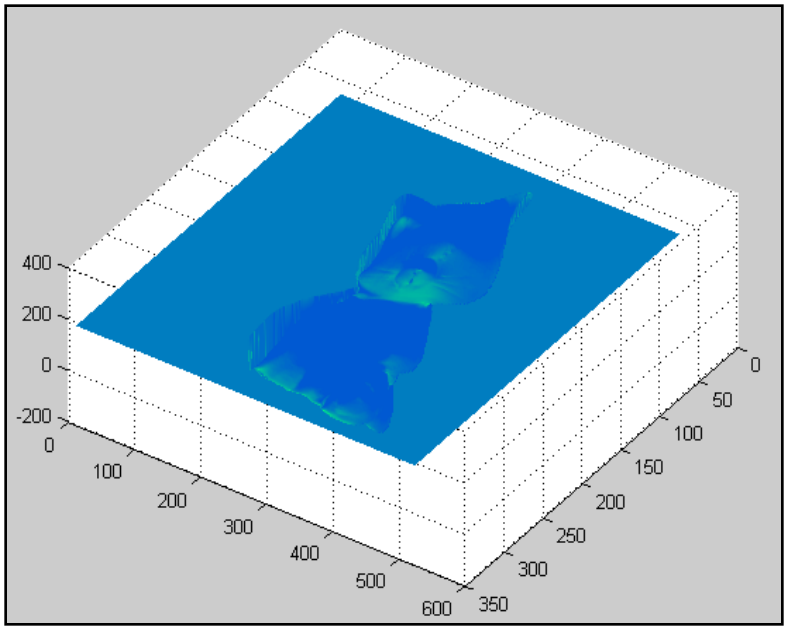


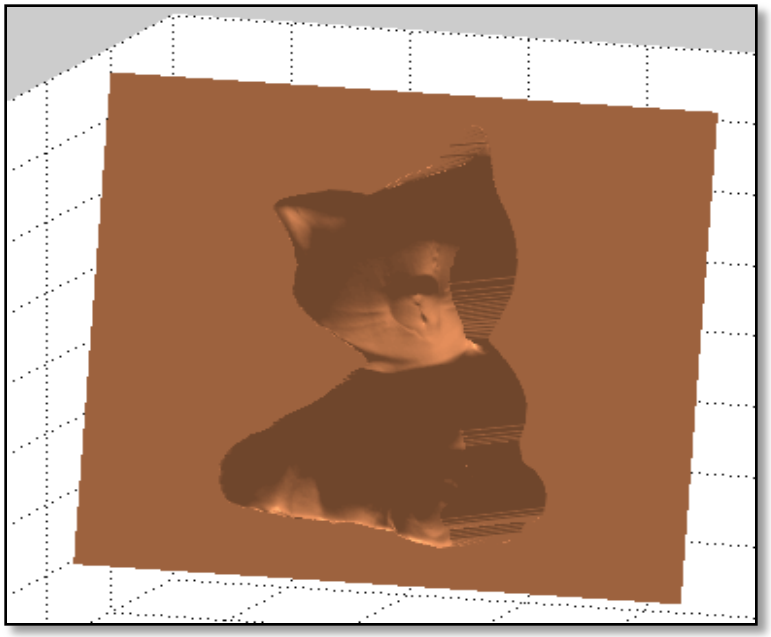
4. Surface Reconstruction

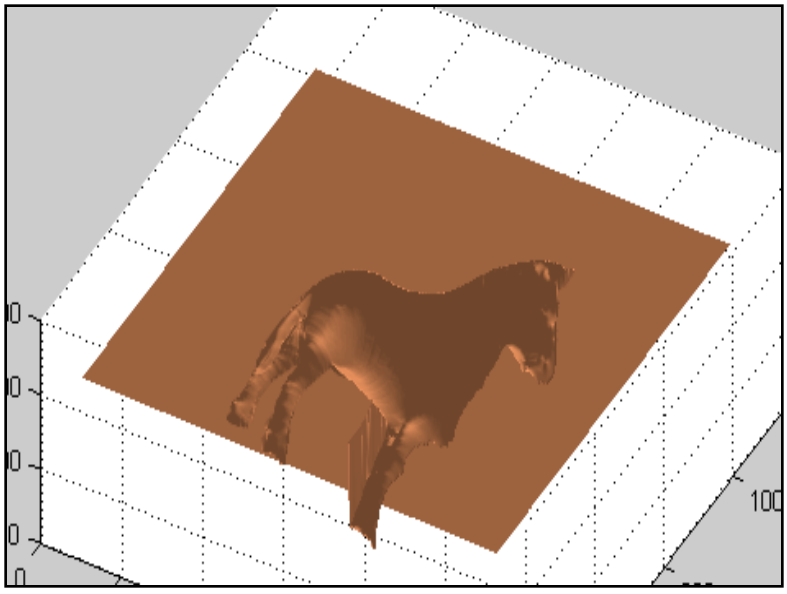
After normal vectors for each point are estimated, we can also estimate the height for different point by solving the equation array

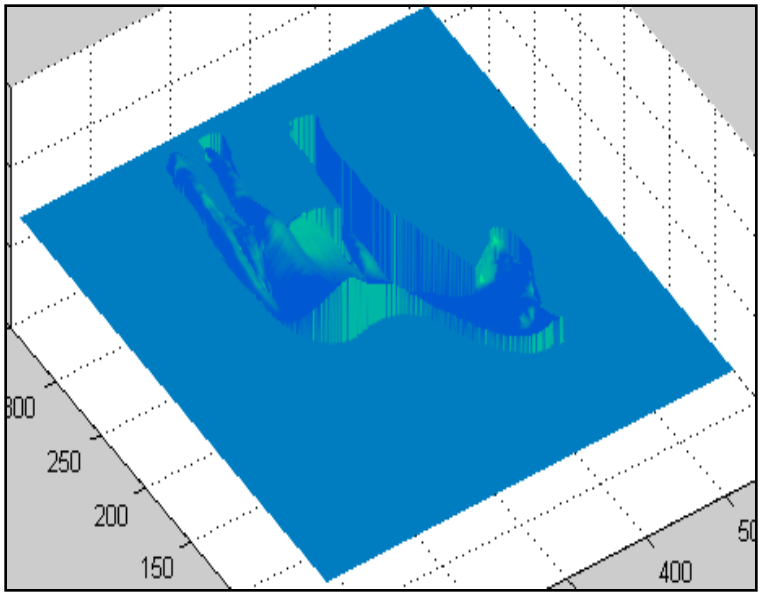
$$n_x + n_z(z(i+1,j) - z(i,j)) = 0 \text{ and } n_y + n_z(z(i,j+1) - z(i,j)) = 0$$

using least square methods. Because of the sparse character of coefficient matrix, we use the sparse matrix in Matlab to save memory in computation. After we estimate the height in the mask area, we can use surfl to plot the 3D shape and can view it from different angle. Below are the results.



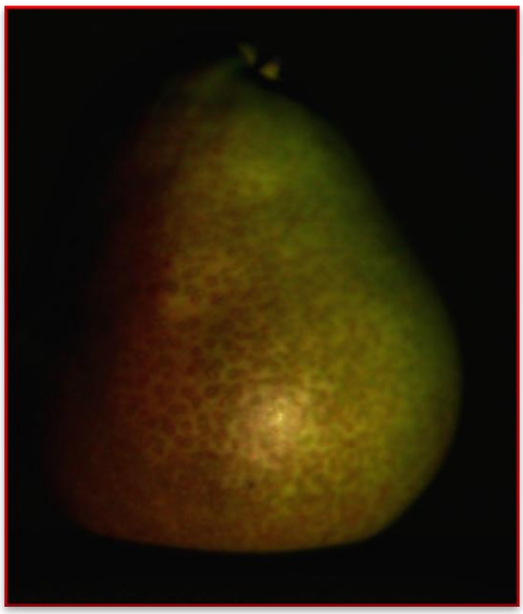
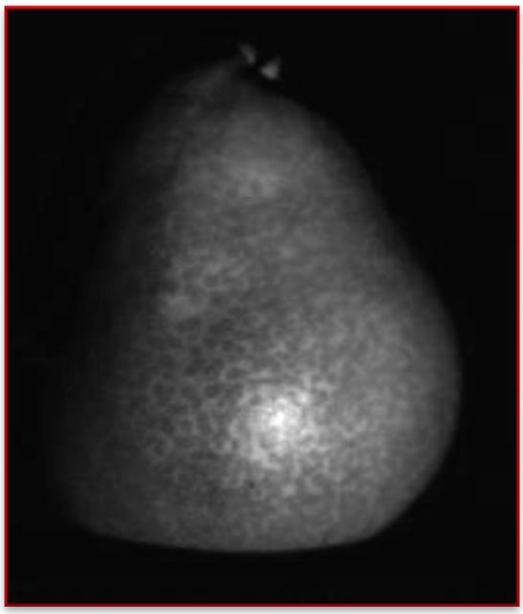
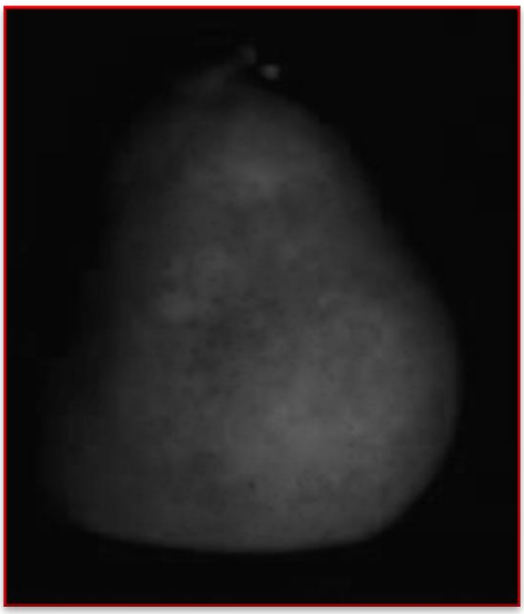


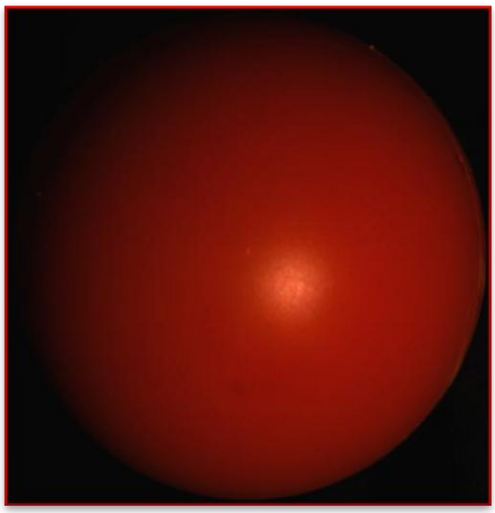
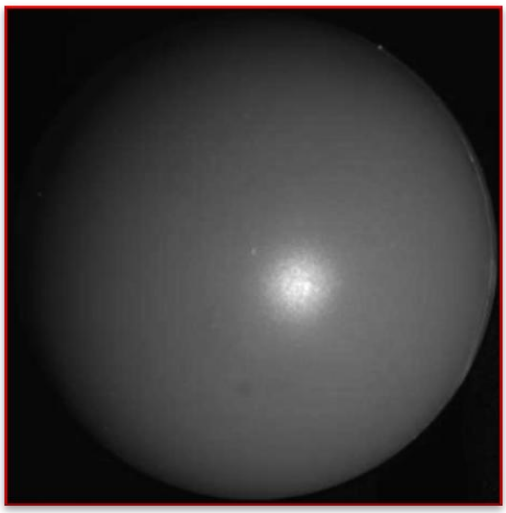




Part – B

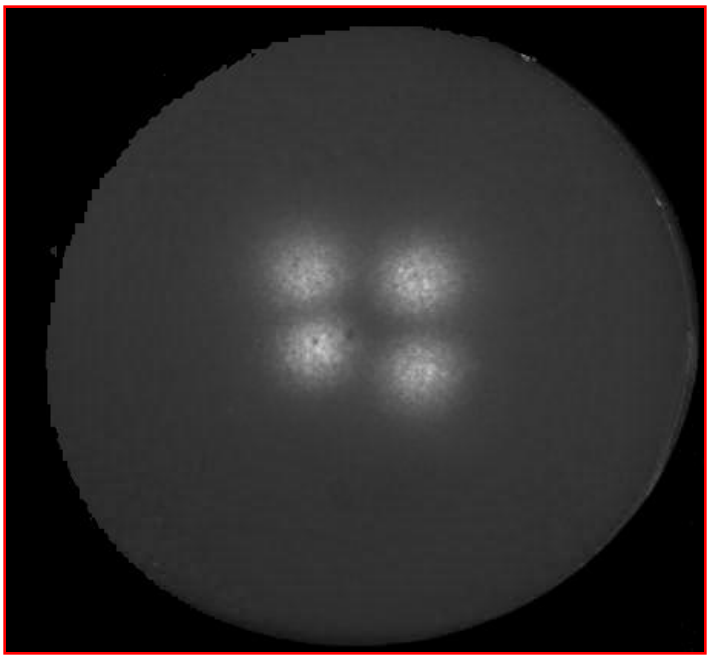
For this part, we implemented the specularity removal technique as specified in the paper *Beyond Lambert: Reconstructing Specular Surfaces Using* Color by Mallick, Zickler. It transforms RGB space into new space called SUV. This is achieved by aligning one axis with the source color, and two of three transformed channels (UV, diffuse image) are independent of specular reflectance. Below images shows original image, specular component and diffuse image.

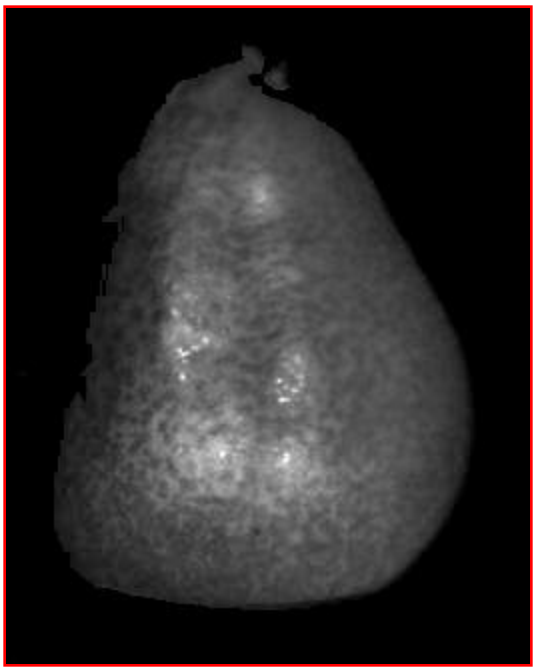




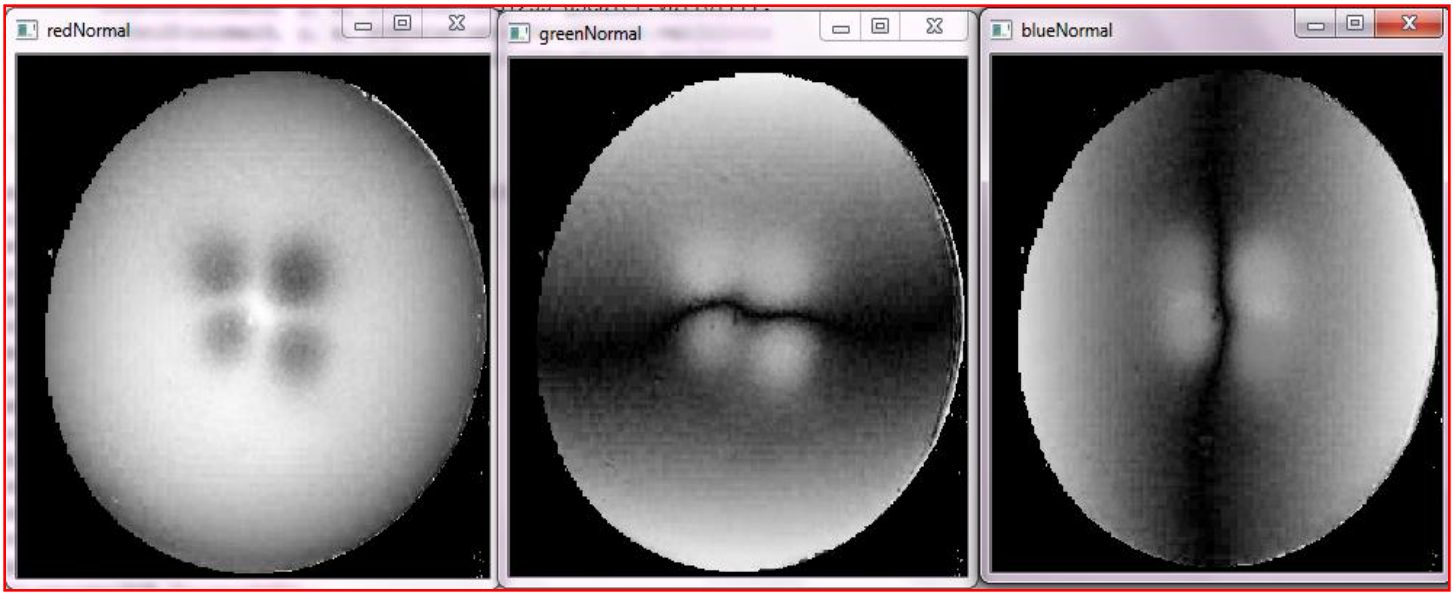



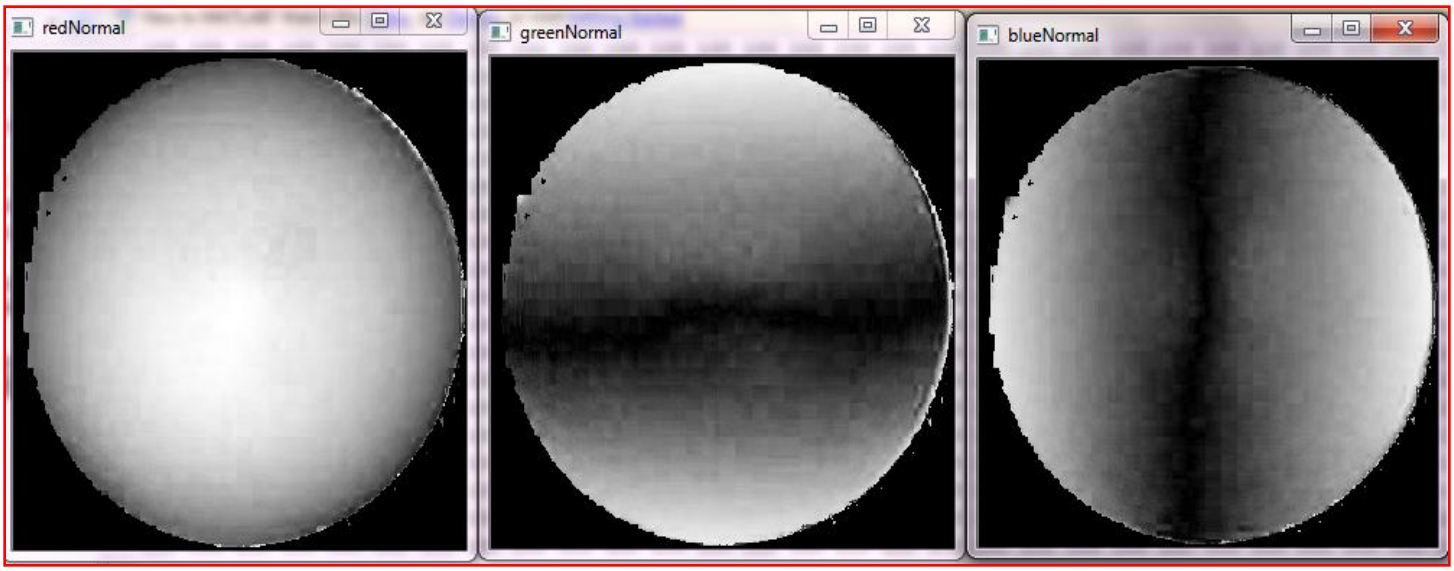

Part – C

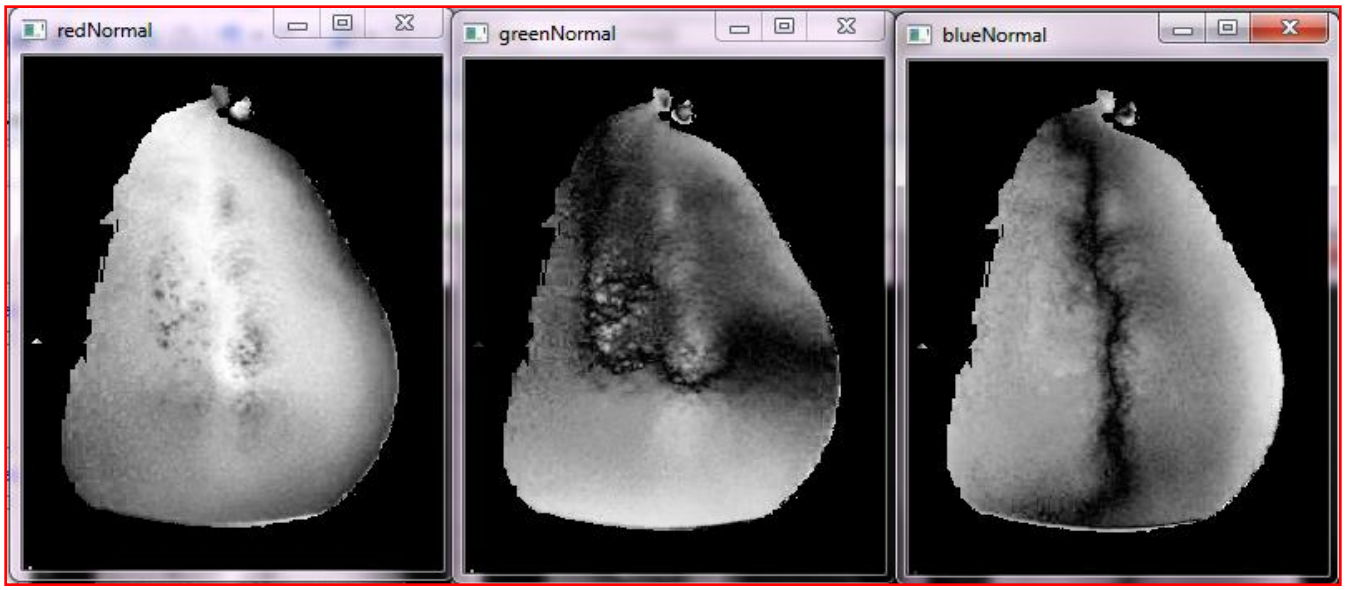
Part C involves combination of part A and B on diffuse images of pear and sphere and comparison of result with original gray images. Below are the images showing Albedo and surface normal for original and corresponding diffuse images.

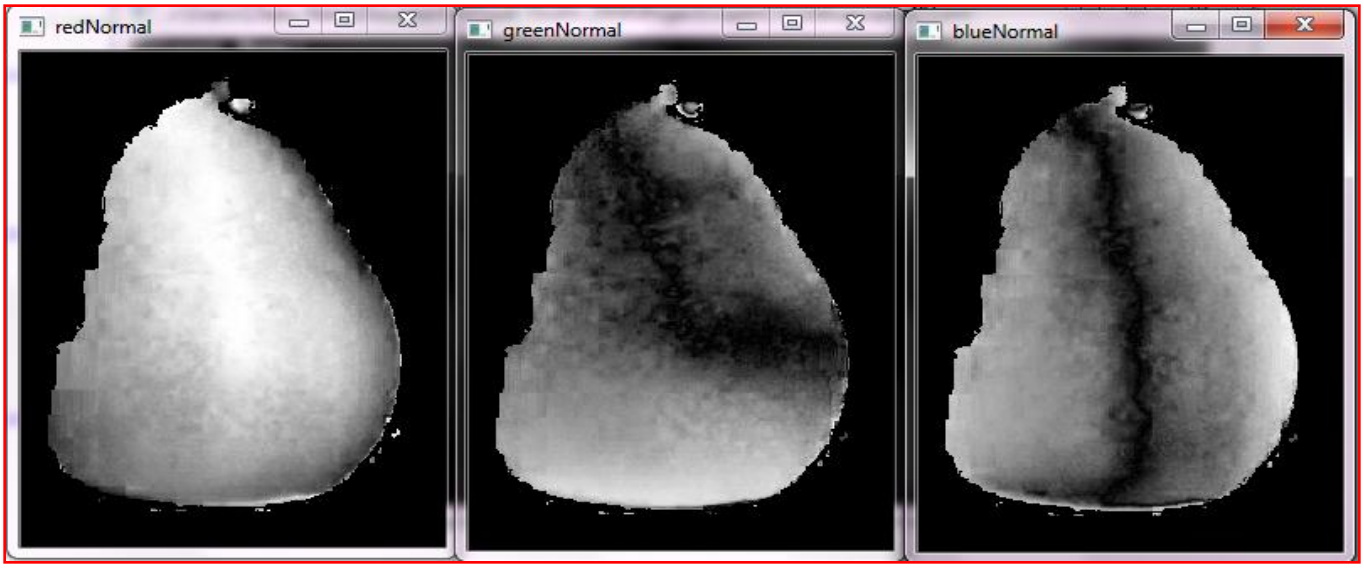




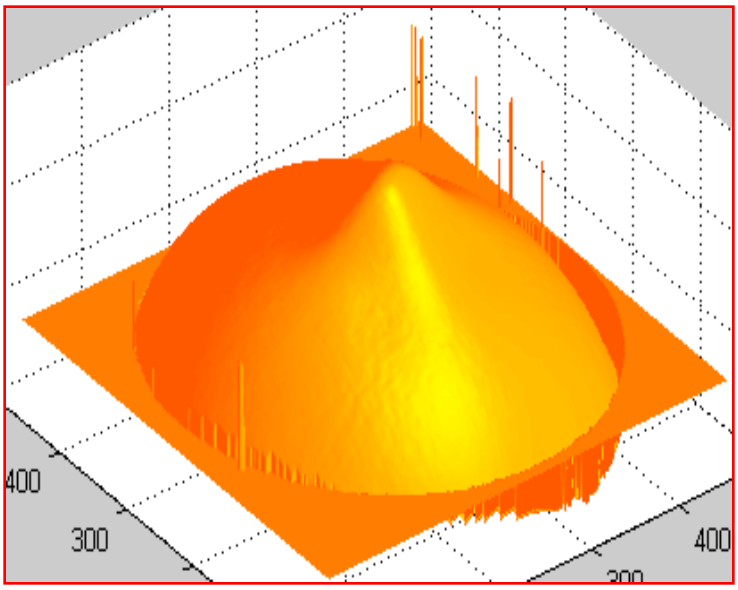

redNormal
greenNormal
blueNormal

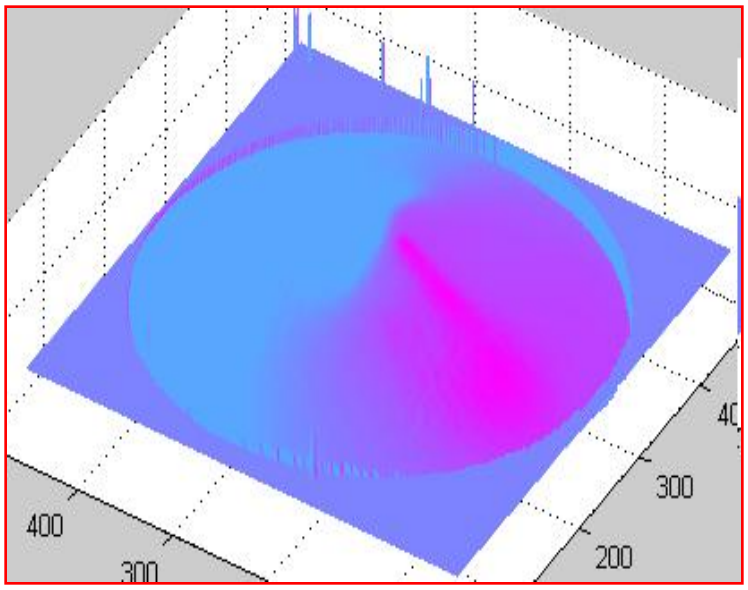

redNormal
greenNormal
blueNormal

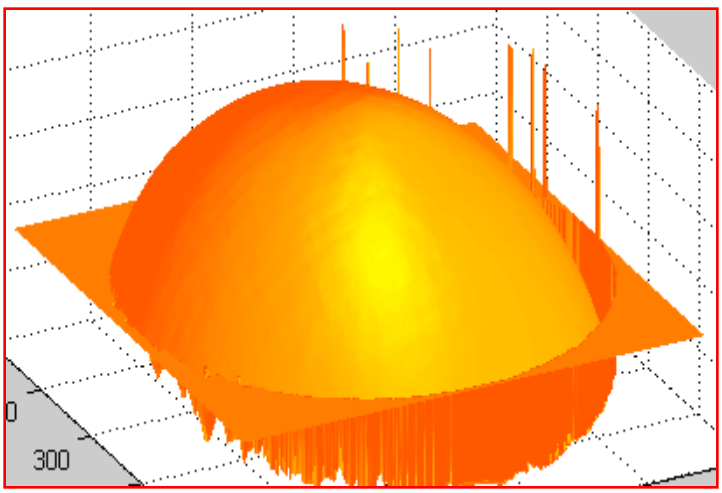

redNormal
greenNormal
blueNormal

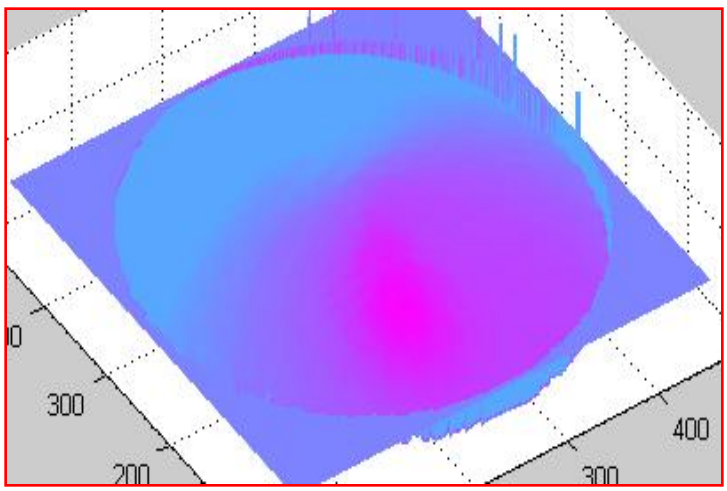


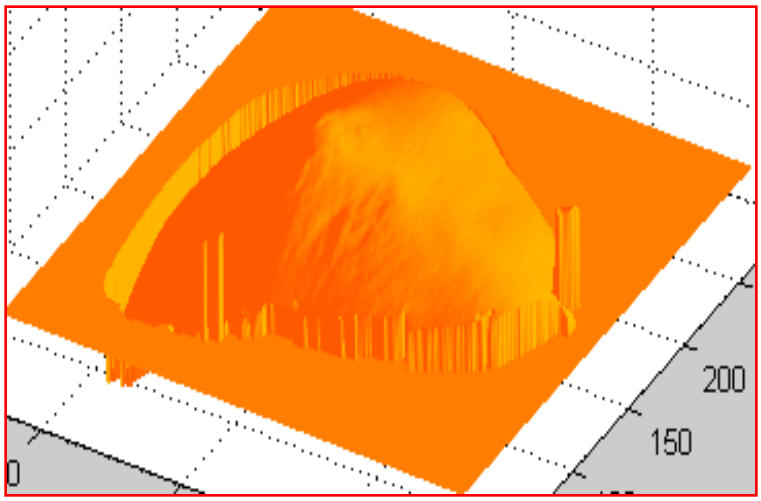
Below images shows reconstruction for original and diffused surface

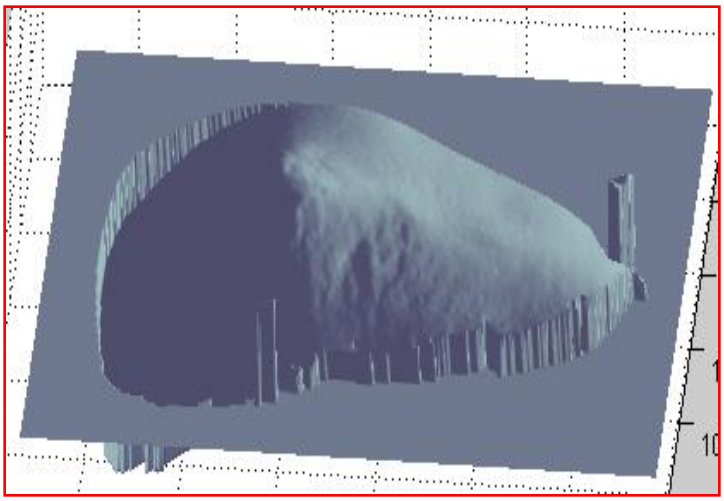


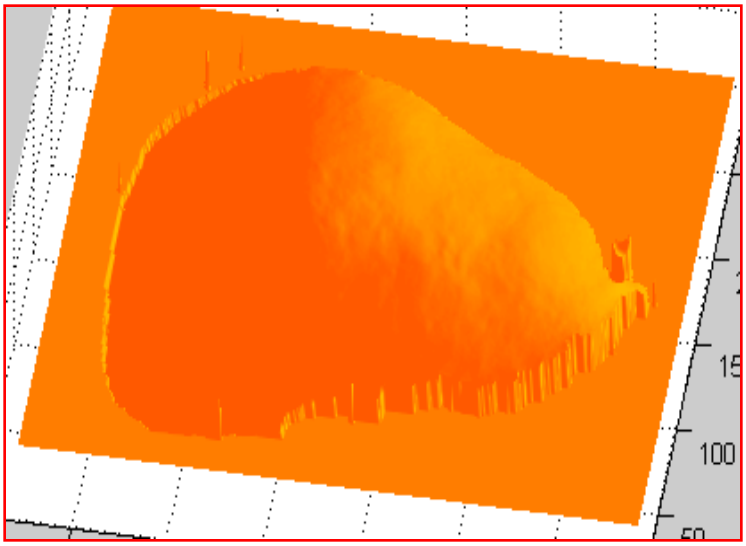


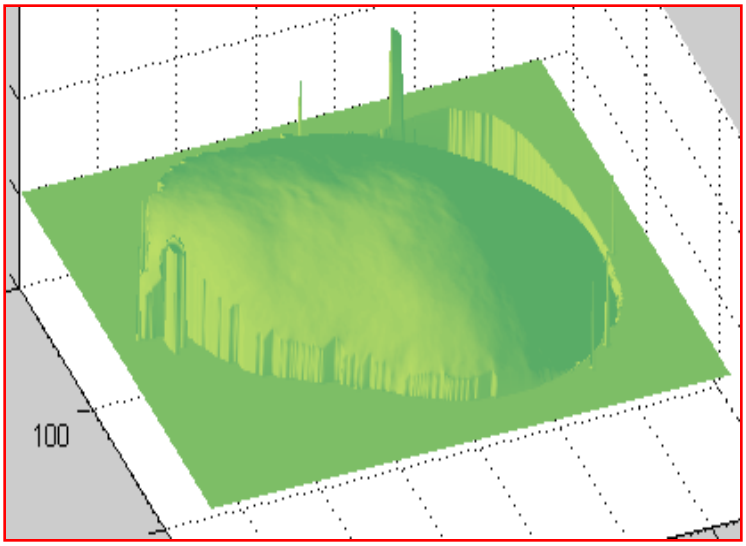












Compile & Run

1. Libraries are used:
    i. OpenCV 2.1 or higher (http://opencv.willowgarage.com/wiki/)
    ii. Csparse (http://www.cise.ufl.edu/research/sparse/CSparse/)
2. Program is developed in Visual Studio 2008 C++ with OpenCV and CSparse libraries.
3. Run main.cpp to compile the program.
4. Console window will prompt to enter name of dataset.