Home Work 3: Assignment Report

Titash Mandal (tm2761@nyu.edu, N13592626)

Reconstruct a surface from its 2D image using the concept of Photometric Stereo

1. SCOPE OF THE EXPERIMENT AND CONCEPT OF SHAPE FROM SHADING.

We focus on the use of point light source to reconstruct the geometry of an object and this technique will also require the use of multiple images to provide results.

2. Concept of Shape from Shading.

It is possible to reconstruct a patch of surface from a series of pictures of that surface taken under different illuminants. First, we need a camera model. For simplicity, we choose a camera situated so that the point (x, y, z) in space is imaged to the point (x, y) in the camera (orthographic projection). To measure the shape of the surface, we need to obtain the depth to the surface. Moving a point light source, we can create a modification of the shading on the acquired image which allows us to compute the surface normal for a given patch. Once we have the set of surface normal vector describing the whole object, we can estimate the possible geometry of the object.

The observed (measured/captured) intensity of an image depends on four main factors:

- **ILLUMINATION-**This is defined by the position, direction and spectral energy distribution of the light rays.
- **SURFACE REFLECTIVITY OF THE OBJECT-** This is known in shape from shading literature as albedo, the proportion of the incident light or radiation that is reflected by a surface, typically that of a planet or moon.
- **SURFACE GEOMETRY OF AN OBJECT**-We want to recover given the object's 2D gray-scale image.
- **CAMERA CAPTURING THE OBJECT**- This is defined by its intrinsic and extrinsic parameters.
- **LAMBERTIAN SURFACES**- We are concerned with objects which have Lambertian surfaces which are whiteish and matte and are not very shiny.

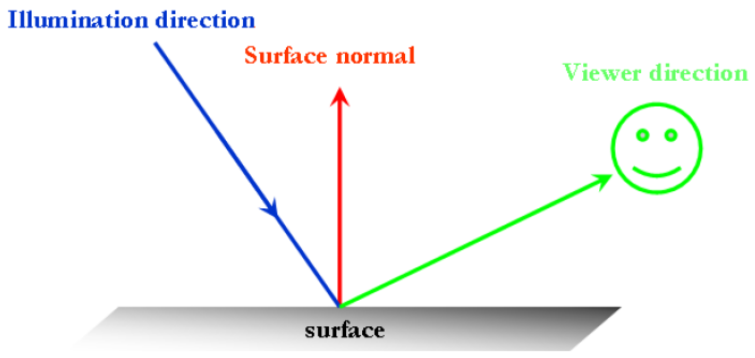


Fig 1.1- A 3D surface is illuminated by a light source defined by its direction, and captured by a camera (viewer) defined by its direction where the surface normal is the unit vector perpendicular on the surface.

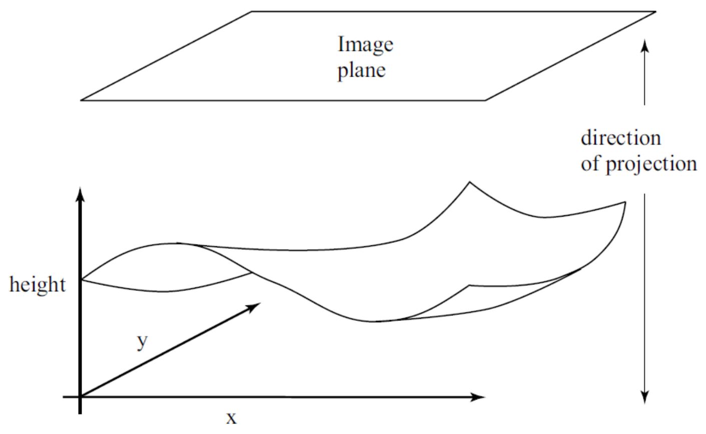
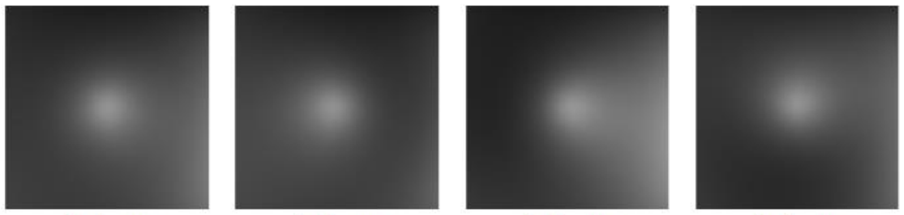


Fig-1.2 - A Monge patch is a representation of a piece of surface as a height function. For the photometric stereo example, we assume that an orthographic camera—one that maps (x, y, z) in space to (x, y) in the camera—is viewing a Monge patch. This means that the shape of the surface can be represented as a function of position in the image

3. Experimental Procedure

3.1 Data Capture

We are provided with a set of four images with a bright area in the center and darker surrounding areas. The images of Lambertian surfaces are taken by changing the directions of the point light source and the values of the direction of the source of light are given to us in vector format. We need to divide the images by 255 to scale them back to a floating-point range of [0.0 · · · 1.0].

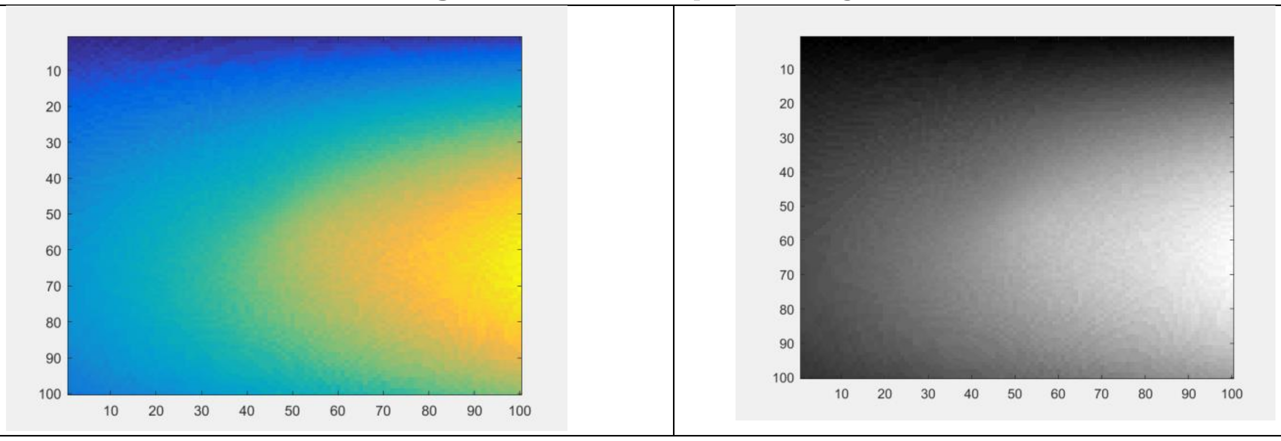


im1.png $\begin{bmatrix}0\\0\\1\end{bmatrix}$ im2.png $\begin{bmatrix}-0.2\\0\\1\end{bmatrix}$ im3.png $\begin{bmatrix}0.2\\0\\1\end{bmatrix}$ im4.png $\begin{bmatrix}0\\-0.2\\1\end{bmatrix}$

Fig-2.1 - Images with the light source directions are provided to us.

3.2 Computing the albedo map for the images

Fig-2.2 The albedo map of the images



- **Albedo** is a measure for reflectance or optical brightness and is measured on a scale from zero (corresponding to a black body that absorbs all incident radiation) to one (corresponding to a white body that reflects all incident radiation).
- If we look at the second and the third image where the light source is symmetrical, we have [-0:2; 0; 1] and [0:2; 0; 1] for the second and the third image respectively.
- When we compare, we can see that the third image is brighter on the right side of the image and the second image is brighter on its left side.
- We can also add that this albedo map make some sense with respect to the dark areas which appear to have a low albedo value and the center of the image that has a higher albedo that agrees with the images.

3.3 Reconstruct and estimate the surface normal vectors

We adopt a local shading model and assume that there is no ambient illumination so that the brightness at a point x on the surface is,

$$\mathbf{B(x)} = \boldsymbol{\rho}\,\mathbf{(x)N(x)} \cdot \mathbf{S_1},$$

where **N** is the unit surface normal and $\mathbf{S_1}$ is the source vector. Now we assume that the response of the camera is linear in the surface radiosity, and so have that the value of a pixel at (x, y) is

$$\begin{aligned}\mathbf{I(x, y)} &= kB(\mathbf{x}) \\ &= kB(x, y) \\ &= k\,\rho\,(x, y)\mathbf{N}(x, y) \cdot \mathbf{S_1} \\ &= \mathbf{g}(x, y) \cdot \mathbf{V_1},\end{aligned}$$

where $\mathbf{g(x, y)} = \boldsymbol{\rho}\,\mathbf{(x, y)N(x, y)}$ and $\mathbf{V_1 = kS_1}$, where k is the constant connecting the camera response to the input radiance. In these equations, **g(x, y)** describes the surface, and $\mathbf{V_1}$ is a property of the illumination and of the camera. We have a dot product between a vector field g(x, y) and a vector V_1, which could be measured; with enough of these dot products, we could reconstruct g and so the surface. Now if we have n sources, for each of which V_i known, we stack each of these V into a known matrix V_i, where

$$\mathcal{V} = \begin{pmatrix} \boldsymbol{V}_1^T \\ \boldsymbol{V}_2^T \\ \cdots \\ \boldsymbol{V}_n^T \end{pmatrix}.$$

For each image point, we stack the measurements into a vector

$$i(x, y) = \{I1(x, y), I2(x, y), \ldots, In(x, y)\}T .$$

Now we have,

$$\mathbf{i}(x, y) = V.\mathbf{g}(x, y),$$

and g is obtained by solving this linear system—or rather, one linear system per point in the image. Typically, n > 3, so that a least-squares solution is appropriate. We can extract the albedo from a measurement of g because N is the unit normal. This means that $|g(x, y)| = \rho\,(x, y)$. This provides a check on our measurements as well. We can extract the surface normal from g because the normal is a unit vector. This method gives us the needle map of the 2-D image.

Observation - If we analyse this result, we can clearly see the shape of a kind of bi dimensional Gaussian kernel. Then the brightest area on the pictures is the center of the image which clearly correspond to the summit of this shape and the normals are clearly oriented to the light source.

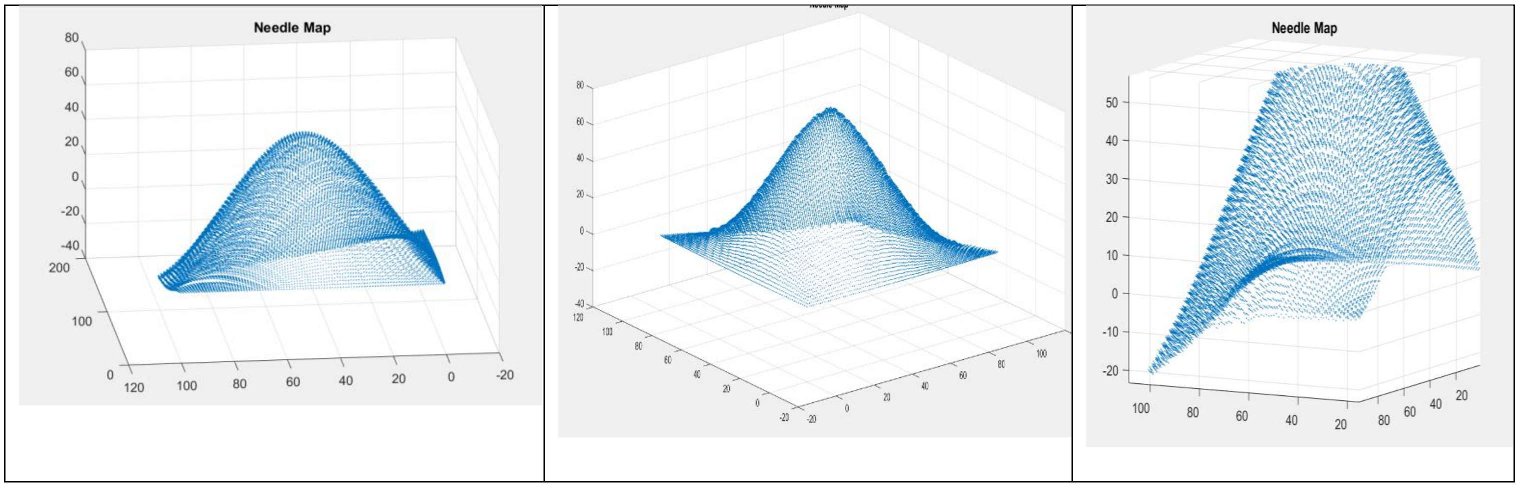
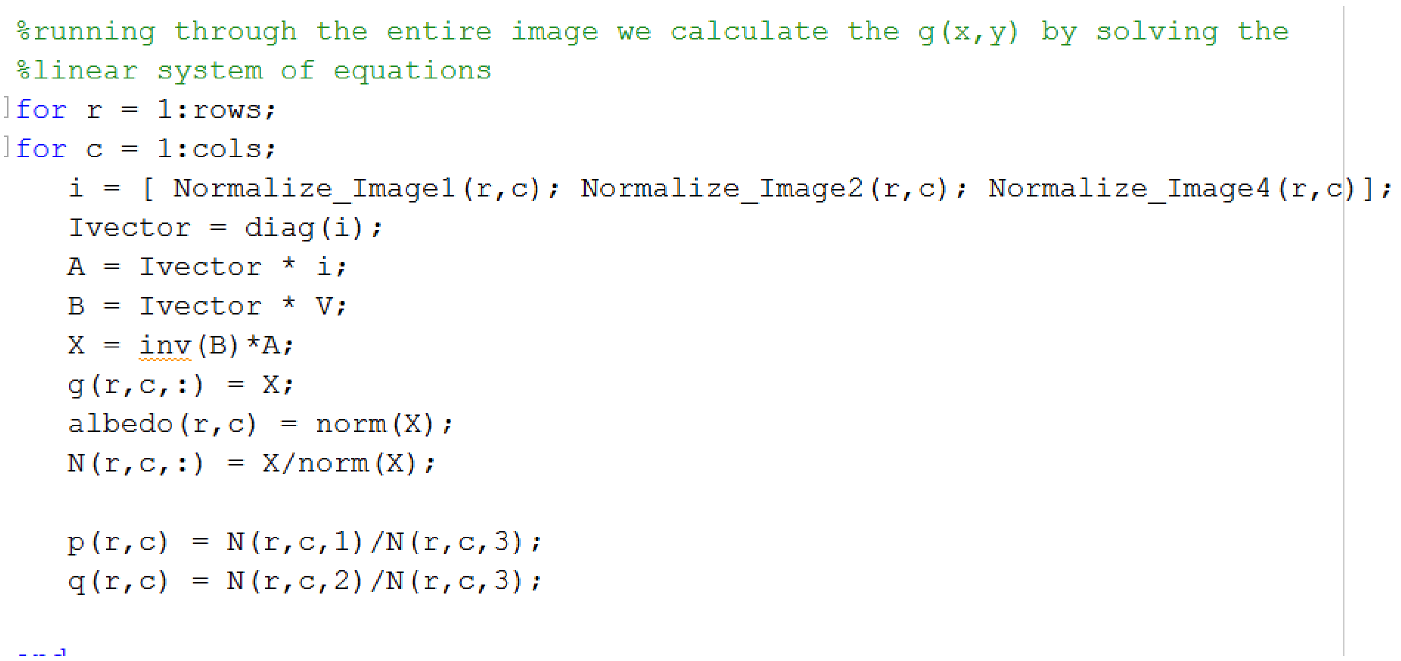


Fig-3.3- Different views of the computed normal vector needle map of the synth_images

3.4 Recovering the surface from the normal vectors.

The partial derivative gives the change in surface height with a small step in either the x or the y direction. We can get the surface by summing these changes in height along some path. In our experiment, we follow the linear least square approach.



```
%running through the entire image we calculate the g(x,y) by solving the
%linear system of equations
for r = 1:rows;
for c = 1:cols;
   i = [ Normalize_Image1(r,c); Normalize_Image2(r,c); Normalize_Image4(r,c)];
   Ivector = diag(i);
   A = Ivector * i;
   B = Ivector * V;
   X = inv(B)*A;
   g(r,c,:) = X;
   albedo(r,c) = norm(X);
   N(r,c,:) = X/norm(X);

   p(r,c) = N(r,c,1)/N(r,c,3);
   q(r,c) = N(r,c,2)/N(r,c,3);
```

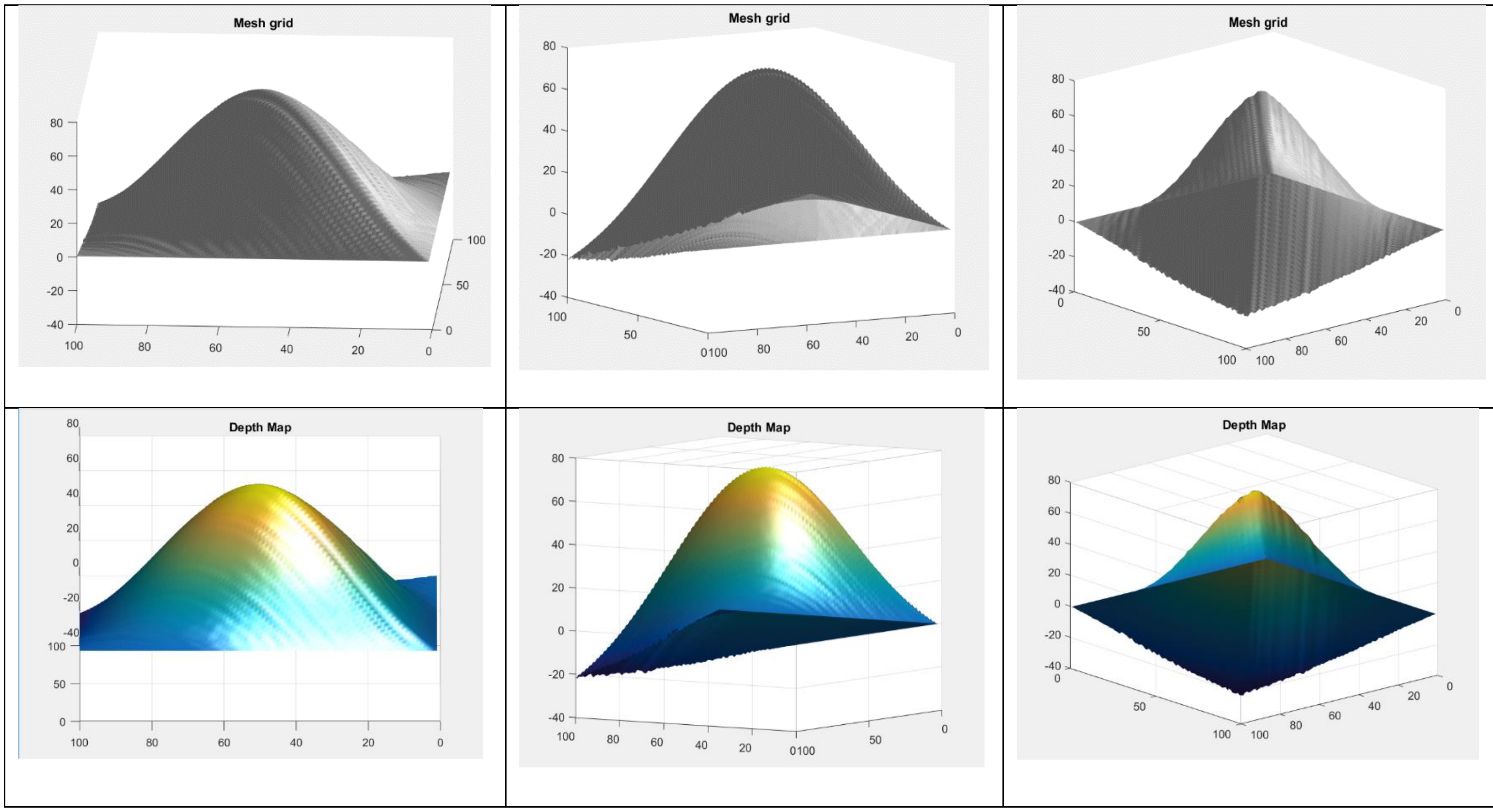


Fig-3.4 The different views of the reconstructed surface based on integration of normal vectors taking images 1,2,4.

3.5 Discussion of the surface reconstruction obtained from the surface normal

We know that in shadow,

$$I(x,y) = 0$$

The reconstruction with the shadow area is not mathematically possible, the algorithm fails to compute the surface normal and to reconstruct the surface. Hence, we use a different trick to also consider areas in shadow for surface reconstruction.

- We construct a diagonal matrix consisting of the inputs of the images.

$$\mathcal{I}(x,y) = \begin{pmatrix} I_1(x,y) & \dots & 0 & 0 \\ 0 & I_2(x,y) & \dots & 0 \\ \dots & & & \\ 0 & 0 & \dots & I_n(x,y) \end{pmatrix}$$

- We multiply the LHS and RHS of the brightness equation with this diagonal matrix to get an equation like this.

$$\mathcal{I}\boldsymbol{i} = \mathcal{I}\mathcal{V}\boldsymbol{g}(x,y)$$

$$\underbrace{\begin{pmatrix} I_1^2(x,y) \\ I_2^2(x,y) \\ .. \\ I_n^2(x,y) \end{pmatrix}}_{\text{Known}} = \underbrace{\begin{pmatrix} I_1(x,y) & 0 & .. & 0 \\ 0 & I_2(x,y) & .. & .. \\ .. & .. & .. & 0 \\ 0 & .. & 0 & I_n(x,y) \end{pmatrix}}_{\text{Known}} \underbrace{\begin{pmatrix} \mathbf{V}_1^T \\ \mathbf{V}_2^T \\ .. \\ \mathbf{V}_n^T \end{pmatrix}}_{\text{Known}} \underbrace{\mathbf{g}(x,y)}_{\text{Unknown}}$$

Solving this linear sets of equations, relevant elements of the left vector and the matrix are zero at points that are in shadow. If we look at the reconstruction made with only three images, we can see that they are not completely accurate.

This is probably due to dark shadow areas on the images that prevent the algorithm from reconstructing the normal because our system is not correctly defined.

<u>THE MATLAB CODE FOR THE EXPERIMENT IS AS FOLLOWS.</u>

```
clc;
clear;
close;

%read the four images
Image4=imread('D:\NYU_SEM_2\Computer Vision\Assignment-3\synth-images\im4.png');
Image3=imread('D:\NYU_SEM_2\Computer Vision\Assignment-3\synth-images\im3.png');
Image2=imread('D:\NYU_SEM_2\Computer Vision\Assignment-3\synth-images\im2.png');
Image1=imread('D:\NYU_SEM_2\Computer Vision\Assignment-3\synth-images\im1.png');
imshow(Image4);

%convert the image values into double
Double_img1 = double(Image1);
Double_img2 = double(Image2);
Double_img3 = double(Image3);
Double_img4 = double(Image4);

%Normalize the images so that the intensity values are between 0-1
Normalize_Image1 =(Double_img1)./255;
Normalize_Image2 =(Double_img2)./255;
Normalize_Image3 =(Double_img3)./255;
Normalize_Image4 =(Double_img4)./255;


%Input the Source vectors corresponding to each image in matrix as given to us
v1 = [0 0 1];
v2 = [-0.2 0 1];
v3 = [0.2 0 1];
v4 = [0 -0.2 1];
%stack the input vectors of the 4 images in another vector
V = [v1; v2; v4];

%calculate the number of rows and columns of the images. It is the same
%value for the four images.
rows = size(Double_img1,1);
cols = size(Double_img1,2);

%initialize the g(x,y) vector where g(x,y)= albedo(x,y).N(x,y)
%initialzie the p , q and the normal Vector N
g = zeros(rows,cols,3);
albedo = zeros(rows,cols);
p = zeros(rows,cols);
q = zeros(rows,cols);
N = zeros(rows,cols, 3);

%running through the entire image we calculate the g(x,y) by solving the
%linear system of equations
for r = 1:rows;
for c = 1:cols;
  i = [ Normalize_Image1(r,c); Normalize_Image2(r,c); Normalize_Image4(r,c)];
  Ivector = diag(i);
  A = Ivector * i;
  B = Ivector * V;
  X = inv(B)*A;
  g(r,c,:) = X;
  albedo(r,c) = norm(X);
  N(r,c,:) = X/norm(X);

  p(r,c) = N(r,c,1)/N(r,c,3);
  q(r,c) = N(r,c,2)/N(r,c,3);

end
end

%To calculate the albedo of the given image whose values should be
%between 0-1
maximumAlbedo = max(max(albedo) );
if(maximumAlbedo > 0)
albedo = albedo/maximumAlbedo;
end

%To calculate Depth Map
depth=zeros(rows,cols);
for i = 2:size(Double_img1,1)
for j = 2:size(Double_img1,2)
depth(i,j) = depth(i-1,j-1)+q(i,j)+p(i,j);
end
end

figure(15);
surfl(-depth);
colormap(gray);
title('Mesh grid ');
grid off;
shading interp

%– Creating a new shaded image by choosing a new light source position vector and calculating
%the dot product of surface normals and light source vector (both normalized)
%at each pixel, resulting in a new image showing a light source not originally used for
%reconstruction

[ X, Y ] = meshgrid( 1:rows, 1:cols );
figure;
surf( X, Y, -depth, 'EdgeColor', 'none' );
camlight left;
title('Depth Map');
lighting phong

%Constructing the Needle Map which shows the normals pointing on the
%surface
figure
spacing = 1;
[X Y] = meshgrid(1:spacing:rows, 1:spacing:cols);
```

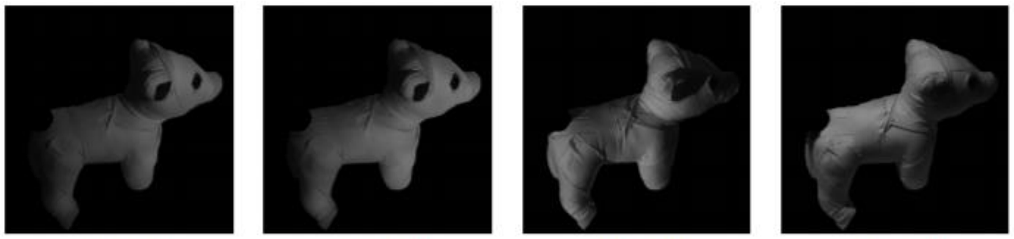
```
quiver3(X,Y,-depth,N(:,:,1),N(:,:,2),N(:,:,3))
hold on;
title('Needle Map');
hold off
```

4. Surface reconstruction from image of a dog.

Similar steps are followed for the images of the dog.

4.1- Data Capture

The four images of the bandaged dog forming a Lambertian surface is given to us. The direction of the light source vectors is also given to us. The light source vectors first need to be normalized before using them. We also need to divide the images by 255 to scale them back to a floating-point range of [0.0 · · · 1.0]. We get satisfactory results with images 2, 3 and 4.



dog1.tif	dog2.tif	dog3.tif	dog4.tif
$\begin{bmatrix}16\\19\\30\end{bmatrix}$	$\begin{bmatrix}13\\16\\30\end{bmatrix}$	$\begin{bmatrix}-17\\10.5\\26.5\end{bmatrix}$	$\begin{bmatrix}9\\-25\\4\end{bmatrix}$

Fig-4.1 Images with the light source directions are provided to us.

4.2 Computing the albedo map for the images

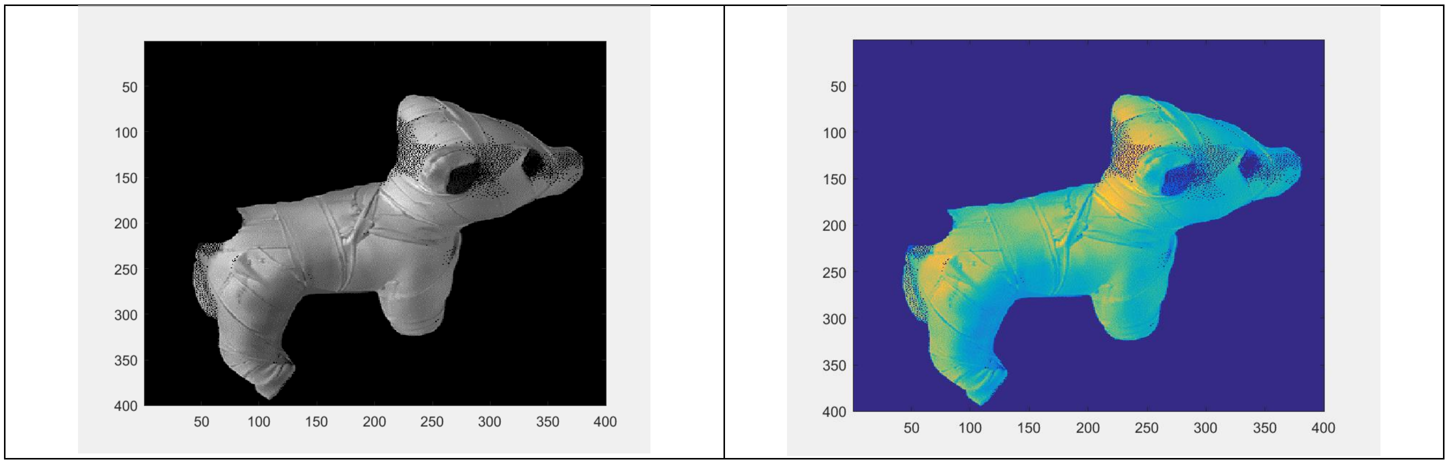


Fig-4.2 The albedo map of the images

The above figure shows the albedo map of the image. Places on the Lambertian surface which have a high reflectance property, are white while other regions are darker. We have used the MATLAB function imagesc that displays the data in image array C as an image that uses the full range of colors in the colormap. Each element of C specifies the color for 1 pixel of the image. The resulting image is an m-by-n grid of pixels where m is the number of columns and n is the number of rows in C.

4.3 Reconstruct and estimate the local normal vectors

The needle map which shows the normals to the surface mesh is first constructed. The surface normals point towards the direction of the light source illumination and help us reconstruct the 3-D surface from the images.

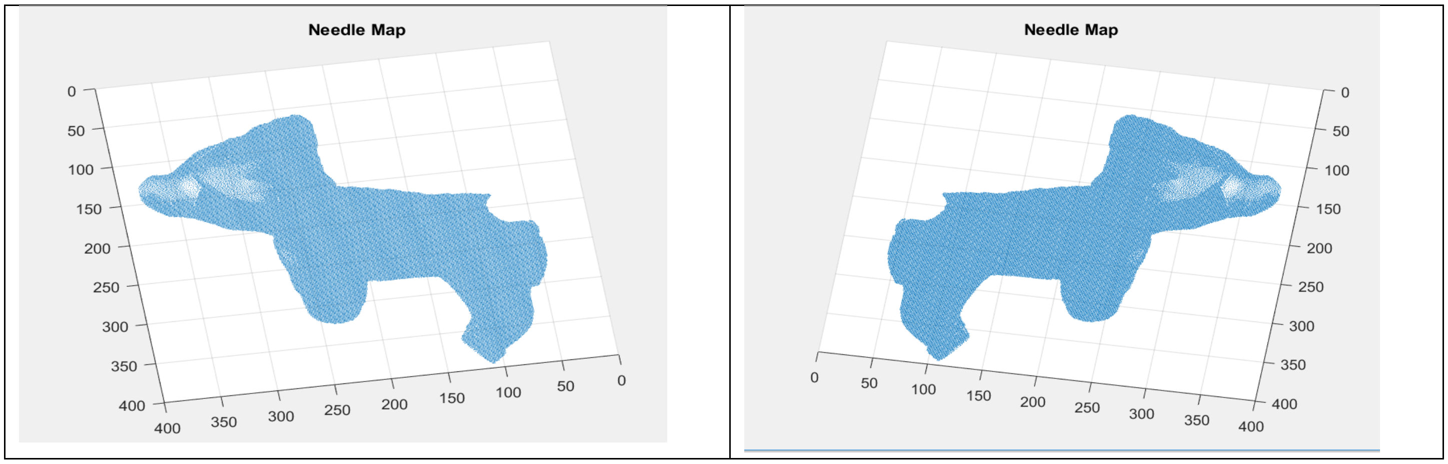


Fig-4.3 The needle map of the images of the dog surface.

4.4- Recovering the surface from the normal vectors.

The surface normals are integrated to obtain the surface from the needle map. Following is the result obtained.

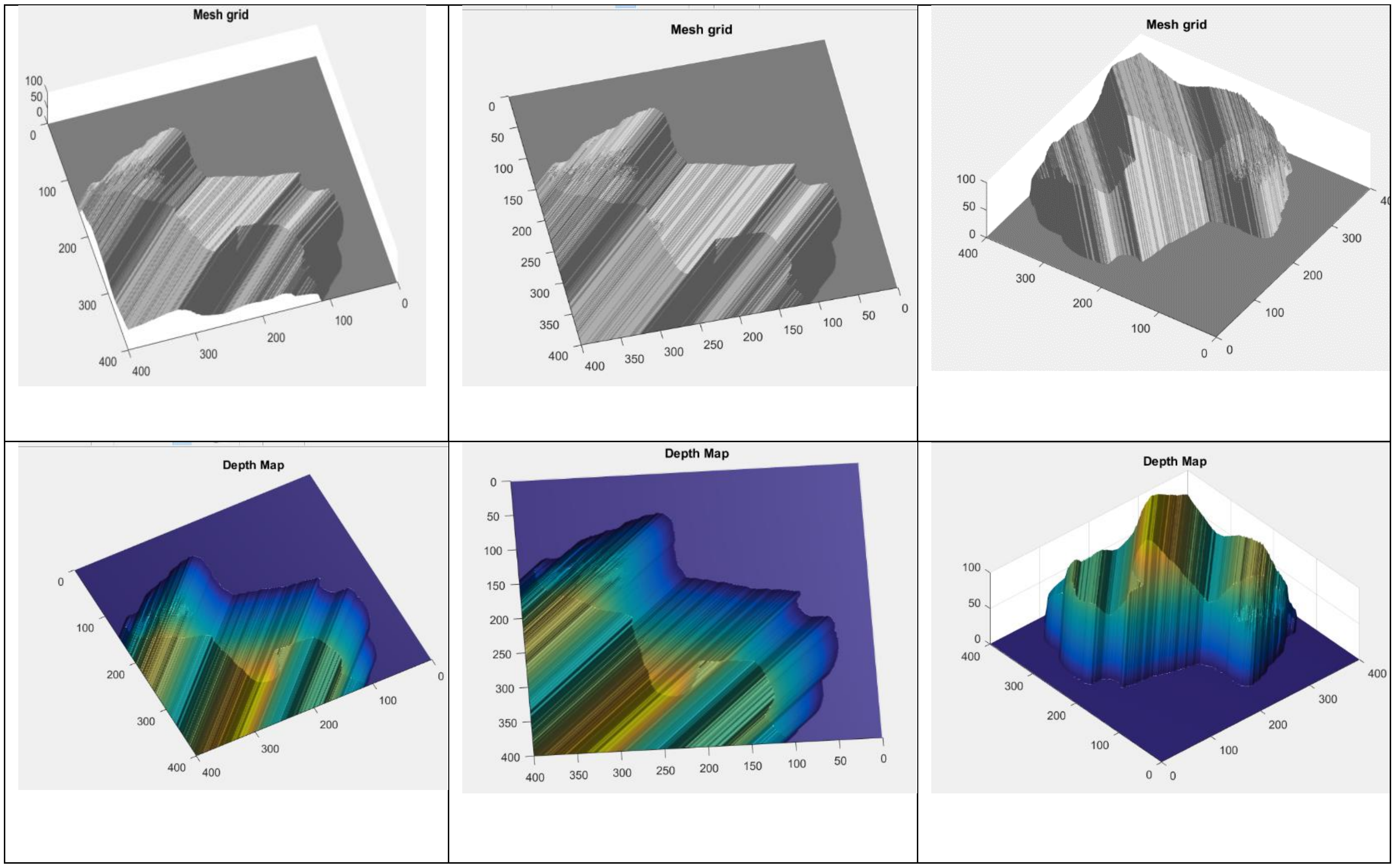


Fig-4.4 The different views of the reconstructed surface based on integration of normal vectors taking images 2,3,4.

Observation- We can see that the 3-D reconstruction of the dog image has a lot of errors because the surface reconstructed does not clearly look like the image of the dog. This can be due to the shading regions where the intensity of light falling on the surface is 0. When errors get summed up due to integration of the gradients p, q to form the surface, the errors get added up giving a less perfect solution. This error can be rectified by using many other methods to reconstruct the surface from the 2-D image. The different methods to do so are as follows:

- Engineering approach: Path integration (Forsyth & Ponce)
- In general: Calculus of Variation Approaches
- Horn: Characteristic Strip Method
- Kimmel, Siddiqi, Kimia, Bruckstein: Level set method

<u>THE MATLAB CODE IS:</u>

```
close;


%read the four images
Image4=(imread('D:\NYU_SEM_2\Computer Vision\dog-png\dog3.png'));
Image3=(imread('D:\NYU_SEM_2\Computer Vision\dog-png\dog3.png'));
Image2=(imread('D:\NYU_SEM_2\Computer Vision\dog-png\dog3.png'));
Image1=(imread('D:\NYU_SEM_2\Computer Vision\dog-png\dog3.png'));
```

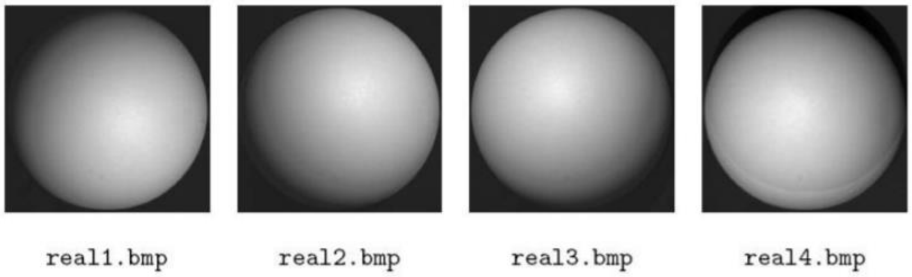
```
%Converting the rgb image to a gray scale image to have scaled intensity values.
Gray_Image4=mat2gray(Image4);
Gray_Image3=mat2gray(Image3);
Gray_Image2=mat2gray(Image2);
Gray_Image1=mat2gray(Image1);


%Normalize the source vectors
v1 = [16;19;30];
Norm_v1=v1/norm(v1,2);
v2 = [13;16;30];
Norm_v2=v2/norm(v2,2);
v3 = [-17;10.5;26.5];
Norm_v3=v3/norm(v3,2);
v4 = [9;-25;4];
Norm_v4=v4/norm(v4,2);

%stack the input vectors of the 4 images in another vector
V = [v2'; v3'; v4'];

%calculate the number of rows and columns of the images. It is the same
%value for the four images.
rows = size(Gray_Image4,1);
cols = size(Gray_Image4,2);

%initialize the g(x,y) vector where g(x,y)= albedo(x,y).N(x,y) initialzie the p , q and the normal Vector N
g = zeros(rows,cols,3);
albedo = zeros(rows,cols);
p = zeros(rows,cols);
q = zeros(rows,cols);
N = zeros(rows,cols, 3);

%running through the entire image we calculate the g(x,y) by solving the
%linear system of equations
for r = 1:rows;
for c = 1:cols;
  i = [ Gray_Image1(r,c); Gray_Image3(r,c); Gray_Image4(r,c)];
  Ivector = diag(i);
  A = Ivector * i;
  B = Ivector * V;
  rank_of_B=rank(B);
  if (rank_of_B<3)
     continue;
  end
      X = B\A;
      g(r,c,:) = X;
      albedo(r,c) = norm(X);
      N(r,c,:) = X/norm(X);
      p(r,c) = N(r,c,1)/N(r,c,3);
      q(r,c) = N(r,c,2)/N(r,c,3);
```

```
end
end

%To caculate the albedo of the given image whose values should be
%between 0-1
maximumAlbedo = max(max(albedo) );
if(maximumAlbedo > 0)
albedo = albedo/maximumAlbedo;
end

%To calculate Depth Map
for i = 2:size(Image4,1)
for j = 2:size(Image4,2)
depth(i,j) = depth(i-1,j-1)+q(i,j)+p(i,j);
end
end
figure(15);
surfl(-depth);
colormap(gray);
title('Mesh grid ');
grid off;
shading interp;

%– Creating a new shaded image by choosing a new light source position vector and calculating
%the dot product of surface normals and light source vector (both normalized)
%at each pixel, resulting in a new image showing a light source not originally used for
%reconstruction

[ X, Y ] = meshgrid( 1:rows, 1:cols );
figure;
surf( X, Y, -depth, 'EdgeColor', 'none' );
camlight left;
title('Depth Map');
lighting phong

%Constructing the Needle Map which shows the normals pointing on the surface
figure
spacing = 1;
[X Y] = meshgrid(1:spacing:rows, 1:spacing:cols);
quiver3(X,Y,-depth,N(:,:,1),N(:,:,2),N(:,:,3))
hold on;
title('Needle Map');
hold off
```

5. Surface reconstruction from image of a Sphere.

5.1- Data Capture

The four images of the sphere forming a Lambertian surface was given to us. The direction of the light source vectors is also given to us. The light source vectors first need to be normalized before using them. We also need to divide the images by 255 to scale them back to a floating-point range of [0.0 · · · 1.0]. We get satisfactory results with images 2, 3 and 4.



$$\begin{bmatrix}0.38359\\0.236647\\0.89266\end{bmatrix}\quad\begin{bmatrix}0.372825\\-0.303914\\0.87672\end{bmatrix}\quad\begin{bmatrix}-0.250814\\-0.34752\\0.903505\end{bmatrix}\quad\begin{bmatrix}-0.203844\\0.096308\\0.974255\end{bmatrix}$$

Fig-5. 1 Images with the light source directions are provided to us.

5.2 Computing the albedo map for the images

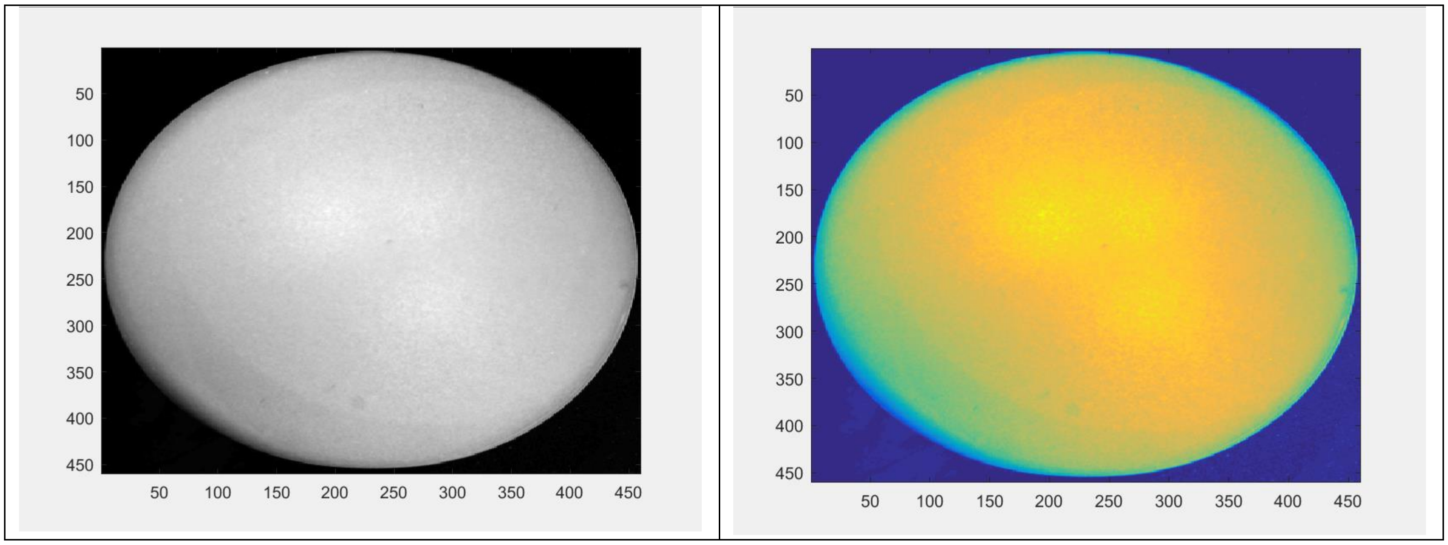


Fig-5.2 The albedo Map of the images

The above figure shows the albedo map of the image. Places on the Lambertian surface which have a high reflectance property, are white while other regions are darker. We have used the MATLAB function imagesc that displays the data in image array C as an image that uses the full range of colors in the colormap. Each element of C specifies the color for 1 pixel of the image. The resulting image is an m-by-n grid of pixels where m is the number of columns and n is the number of rows in C.

5.3 Reconstruct and estimate the local normal vectors

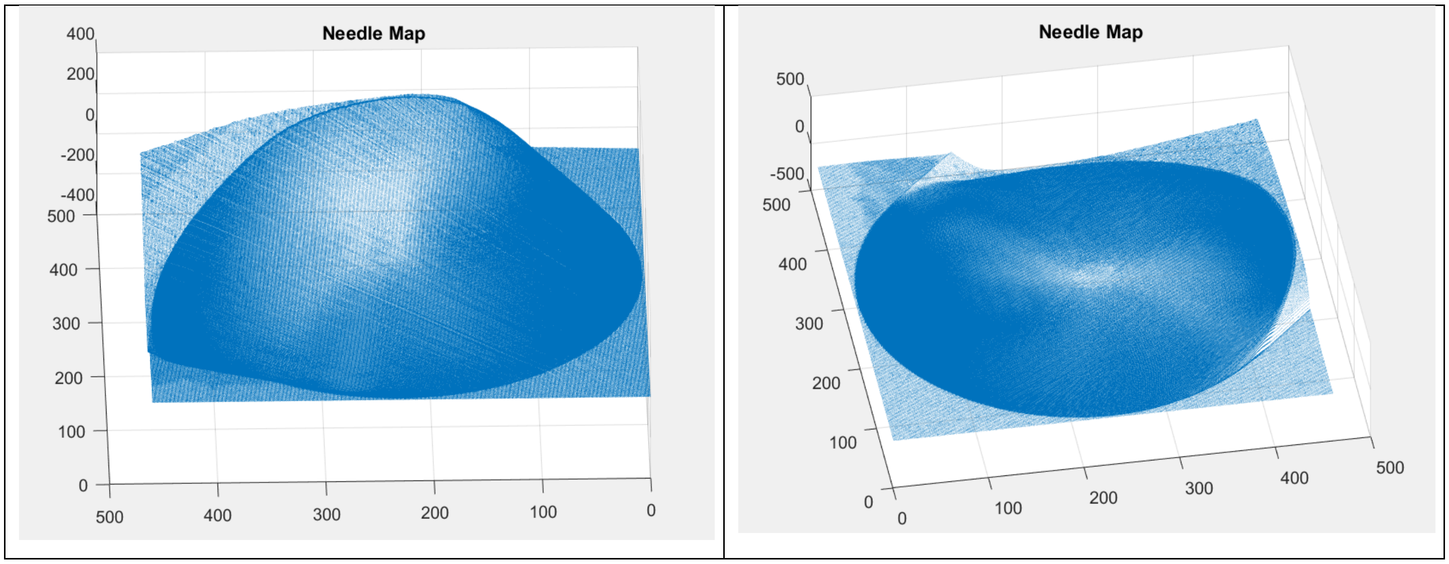


Fig-5.3 The needle map from the images of the sphere surface.

5.3- Recovering the surface from the normal vectors.

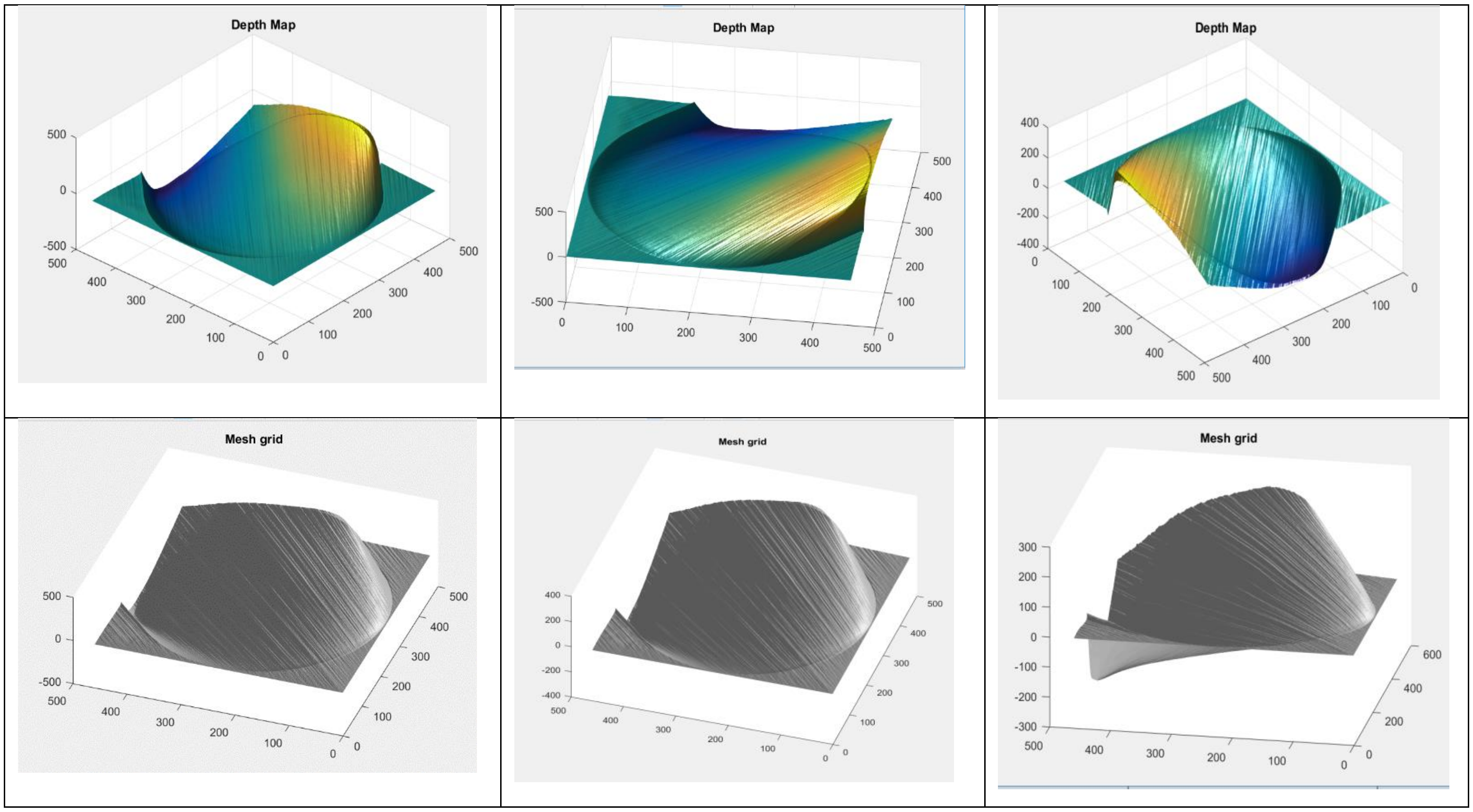


Fig-5.4 The different views of the reconstructed sphere surface based on integration of normal vectors taking images 2,3,4.

Observation- If we look at the reconstruction made with only three images, we can see that they are not really accurate. This is probably due to dark shadow areas on several images that prevent the algorithm from reconstructing the normals because our system is not correctly defined. We can derive the three dimensional surface from the 2D images but because we have used the matrix equalization method to also include areas under shadow, we can see that the errors are visible in the final surface.

MATLAB CODE FOR THE SPHERE SURFACE RECONSTRUCTION IS :

```
close;

%read the four images
Image4=double(imread('D:\NYU_SEM_2\Computer Vision\Assignment-3\sphere-images (2)\real1.bmp'));
Image3=double(imread('D:\NYU_SEM_2\Computer Vision\Assignment-3\sphere-images (2)\real2.bmp'));
Image2=double(imread('D:\NYU_SEM_2\Computer Vision\Assignment-3\sphere-images (2)\real3.bmp'));
Image1=double(imread('D:\NYU_SEM_2\Computer Vision\Assignment-3\sphere-images (2)\real4.bmp'));

% %Normalize the images so that the intensity values are between 0-1
 Normalize_Image1 =(Image1)./255;
 Normalize_Image2 =(Image2)./255;
 Normalize_Image3 =(Image3)./255;
 Normalize_Image4 =(Image4)./255;


%Normalize the source vectors
v1 = [0.38359;0.236647;0.89266];
Norm_v1=v1/norm(v1,2);
v2 = [0.372825;-0.303914;0.87672];
Norm_v2=v2/norm(v2,2);
v3 = [-0.250814;-0.34752;0.903505];
Norm_v3=v3/norm(v3,2);
v4 = [-0.203844;0.096308;0.974255];
Norm_v4=v4/norm(v4,2);

%stack the input vectors of the 4 images in another vector
V = [Norm_v2'; Norm_v3'; Norm_v4'];

%calculate the number of rows and columns of the images. It is the same
%value for the four images.
rows = size( Normalize_Image1,1);
cols = size( Normalize_Image1,2);

%initialize the g(x,y) vector where g(x,y)= albedo(x,y).N(x,y)
%initialzie the p , q and the normal Vector N
g = zeros(rows,cols,3);
albedo = zeros(rows,cols);
p = zeros(rows,cols);
q = zeros(rows,cols);
N = zeros(rows,cols, 3);

%running through the entire image we calculate the g(x,y) by solving the
%linear system of equations
for r = 1:rows;
for c = 1:cols;
  i = [  Normalize_Image2(r,c); Normalize_Image3(r,c); Normalize_Image4(r,c)];
  Ivector = diag(i);
  A = Ivector * i;
  B = Ivector * V;
  rank_of_B=rank(B);
  if (rank_of_B<3)
     continue;
  end
      X = B\A;
      g(r,c,:) = X;
      albedo(r,c) = norm(X);
      N(r,c,:) = X/norm(X);
      p(r,c) = N(r,c,1)/N(r,c,3);
      q(r,c) = N(r,c,2)/N(r,c,3);

end
end

%To caculate the albedo of the given image whose values should be
%between 0-1
maximumAlbedo = max(max(albedo) );
if(maximumAlbedo > 0)
albedo = albedo/maximumAlbedo;
end

%To calculate Depth Map
for i = 2:size(Image4,1)
for j = 2:size(Image4,2)
depth(i,j) = depth(i-1,j-1)+q(i,j)+p(i,j);
end
end



figure(15);
surfl(-depth);
colormap(gray);
title('Mesh grid ');
grid off;
shading interp

%– Creating a new shaded image by choosing a new light source position vector and calculating
%the dot product of surface normals and light source vector (both normalized)
%at each pixel, resulting in a new image showing a light source not originally used for
%reconstruction
[ X, Y ] = meshgrid( 1:rows, 1:cols );
figure;
% Surf- creates a three-dimensional surface plot.
%The function plots the values in matrix Z as heights above a grid in the x-y plane defined by X and Y.
surf( X, Y, -depth, 'EdgeColor', 'none' );
camlight left;
title('Depth Map');
lighting phong

%Constructing the Needle Map which shows the normals pointing on the
%surface
figure
spacing = 1;
[X Y] = meshgrid(1:spacing:rows, 1:spacing:cols);
quiver3(X,Y,-depth,N(:,:,1),N(:,:,2),N(:,:,3))
hold on;
title('Needle Map');
```

```
hold off
```

6. Bonus question: Reconstruction of the surface by integration with the Frankot and Chellappa method.

- The reconstruction of surfaces from estimated gradients, called surface-from-gradients (SfG), is an essential step for the methods such as shape-from-shading (SfS) and photometric stereo. SfS can estimate a dense and noisy gradient field, which is used to calculate the height field by integration.
- Mathematically, a gradient field generated from a three-dimensional surface should be integrable – that is the integral along any closed path should be equal to zero and **integration results** should be **independent to the selection of path choice.**
- However, in practices, the gradient fields produced by PS (or SfS) are rarely integrable due to the inevitable noises generated during the estimation process.
- To integrate a noisy gradient, we have tried to reconstruct the 3D surface from the images by the implementation of Frankot and Chellappa'a algorithm for constructing an integrable surface from gradient information.
- In this approach , they enforce integrability by projecting a dense non-integrable field of gradients onto a set of integrable slopes using the Fourier basis functions.
- An integrable surface is defined in terms of an orthonormal set of gradients field, and then the gradient field is partially projected onto the gradient space to obtain a partially integrable field.
- However, a common problem of approaches enforcing integrability is that important sharp features may be lost when the integrability constraints are not satisfied on the input gradients.

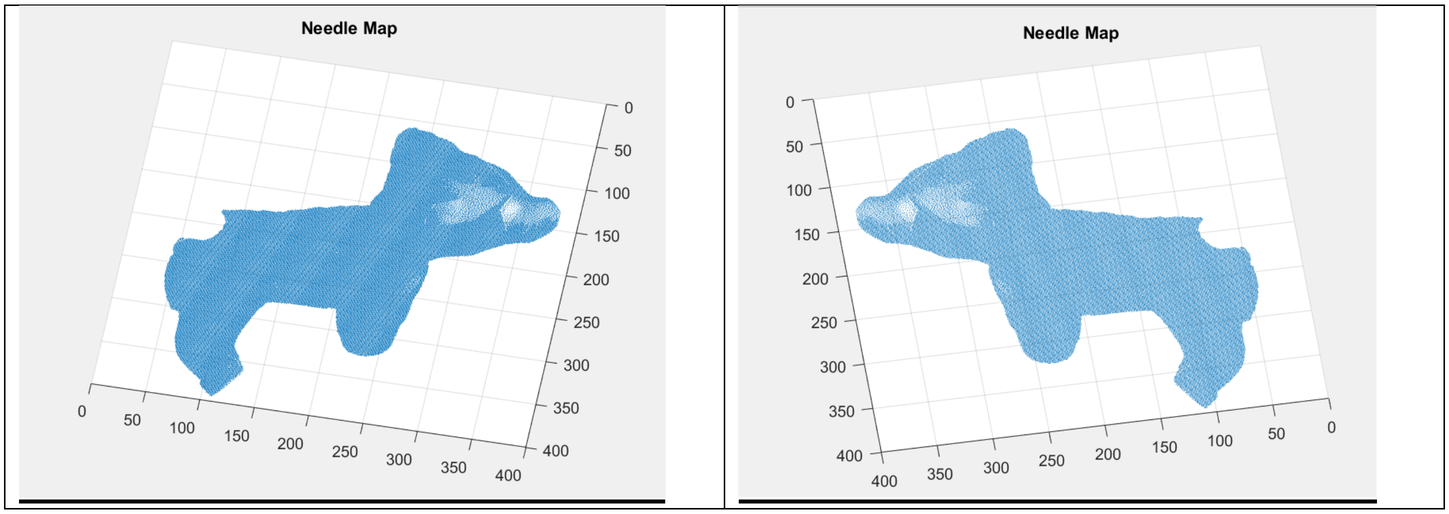


Fig-6.1 The needle map from the images of dog constructed by Frankot and Chellappa'a algorithm

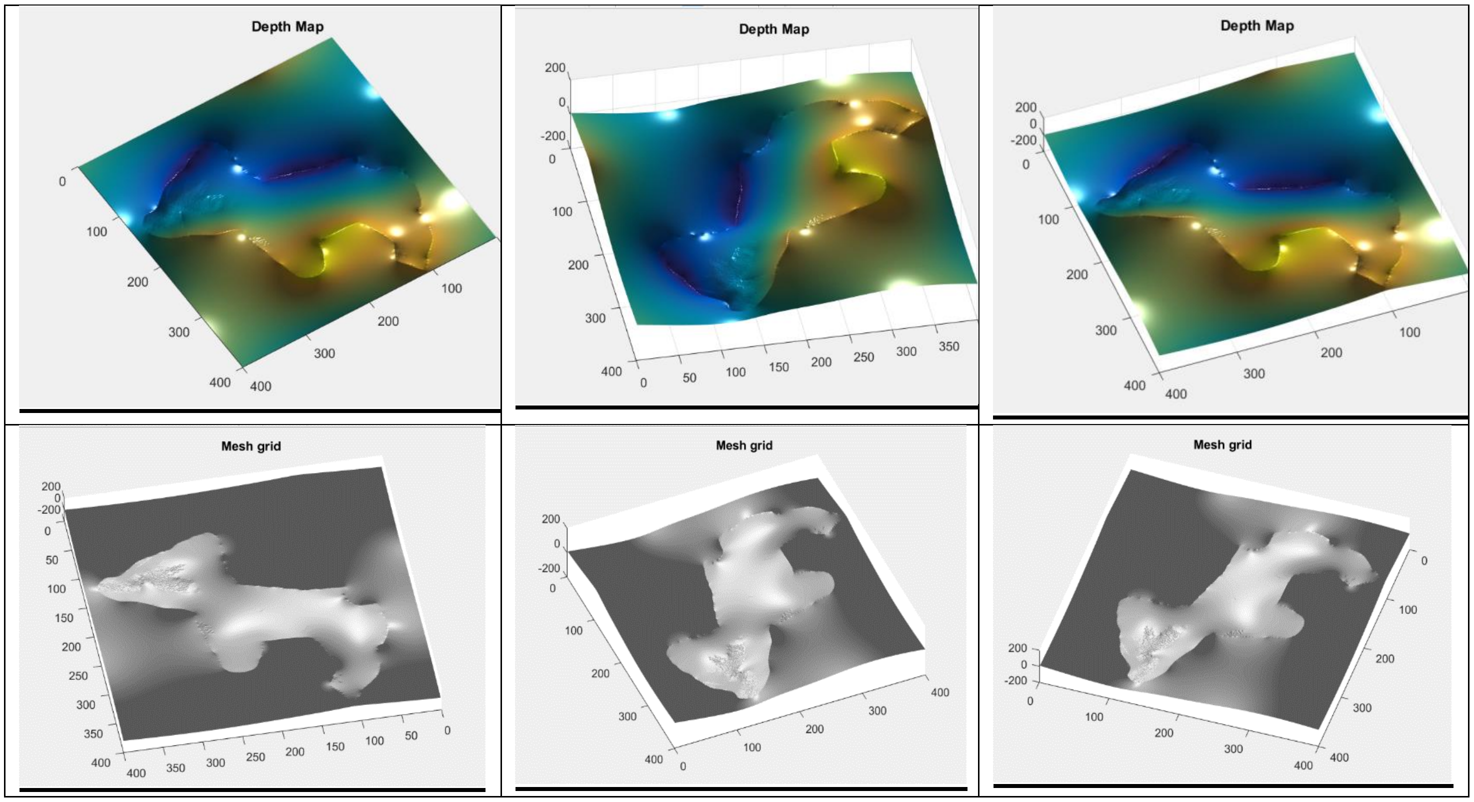


Fig-6.2 Different views of the 3-D reconstructed surface of the dog

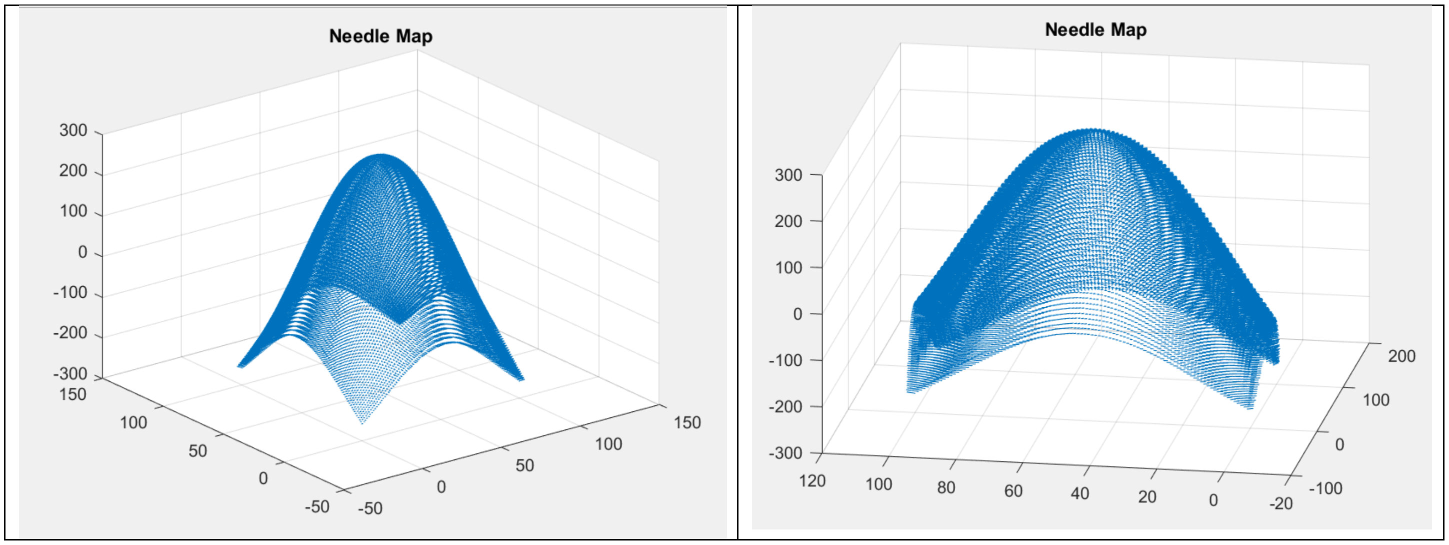


Fig-6.3 The needle map of surface normal from the synth_images.

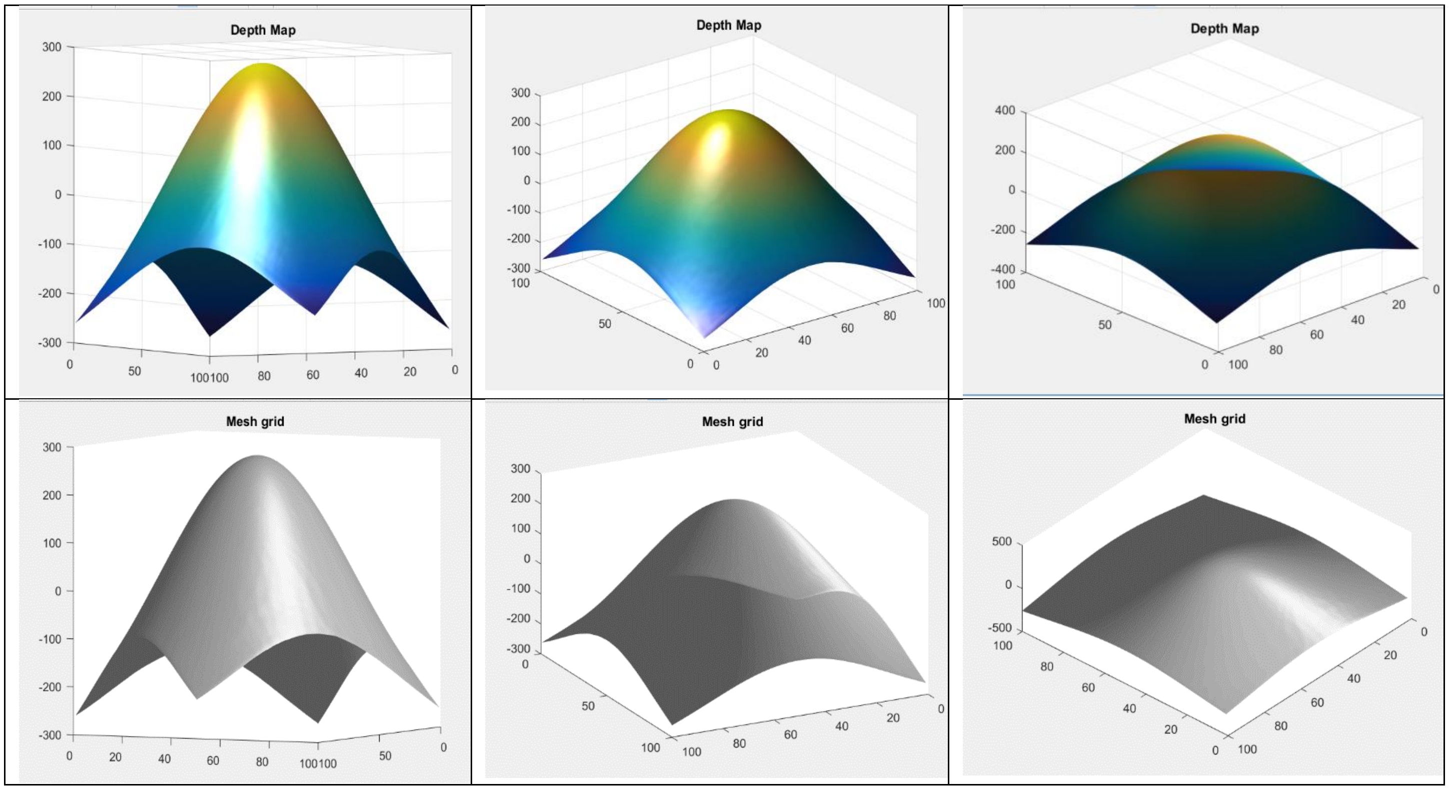


Fig-6.4 Different views of the 3-D reconstructed surface of the synth_image

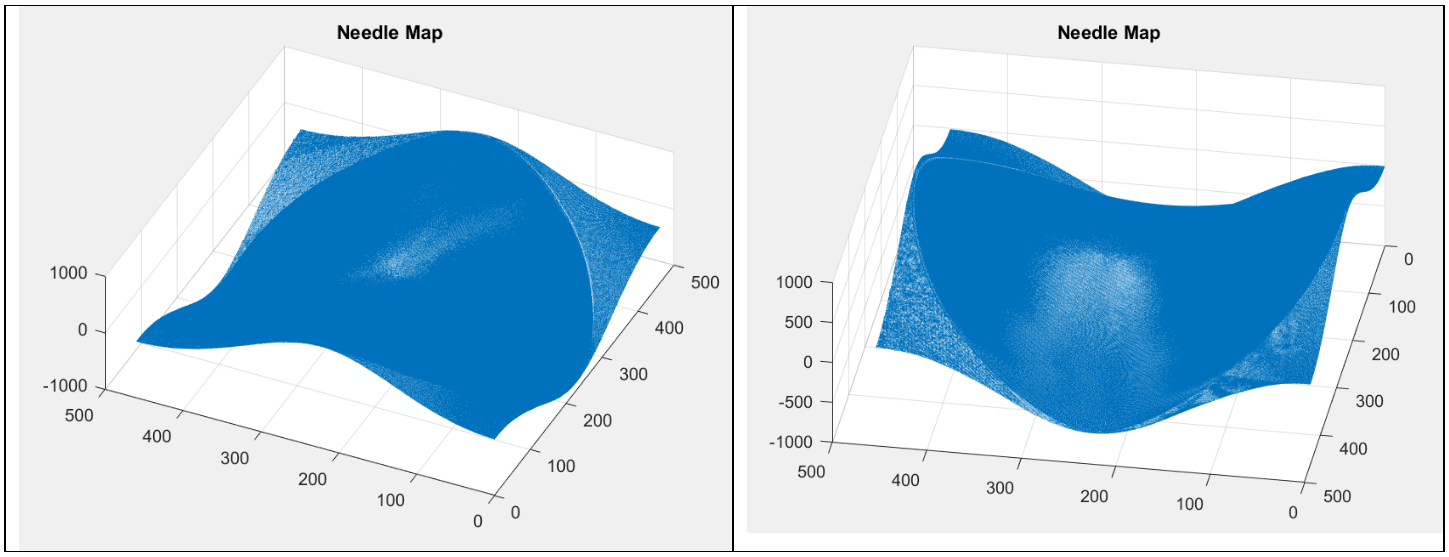


Fig-6.5 The needle map of surface normal from the images of the sphere surface

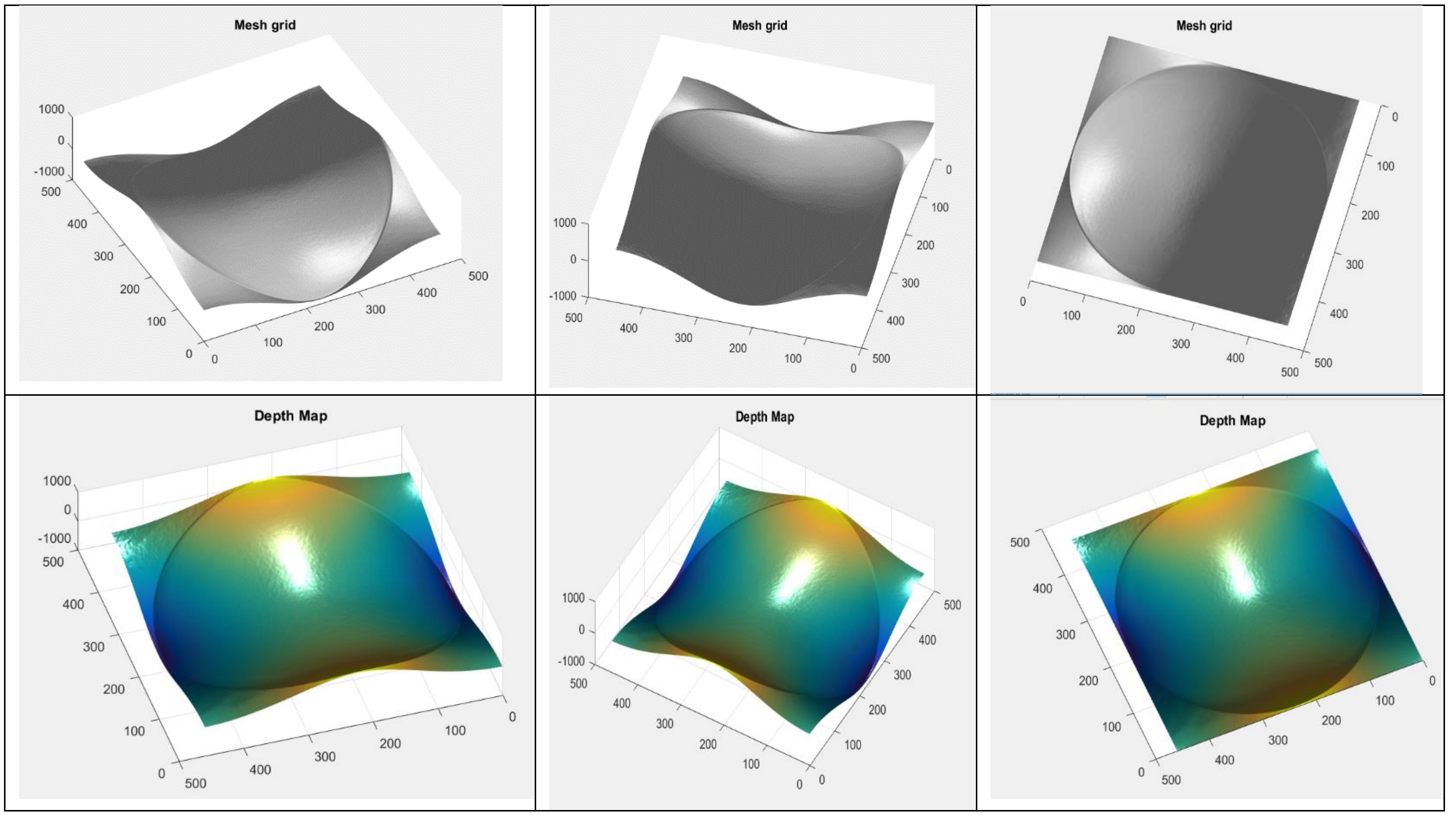


Fig-6.6 Different views of the 3-D reconstructed surface of the sphere

6.5 MATLAB CODE FOR SURFACE RECONSTRUCTION OF THE DOG IMAGE USING THE ALGORITHM OF FRANKOT AND CHELLAPPA

```
close;

%read the four images
Image4=(imread('D:\NYU_SEM_2\Computer Vision\dog-png\dog3.png'));
Image3=(imread('D:\NYU_SEM_2\Computer Vision\dog-png\dog3.png'));
Image2=(imread('D:\NYU_SEM_2\Computer Vision\dog-png\dog3.png'));
Image1=(imread('D:\NYU_SEM_2\Computer Vision\dog-png\dog3.png'));

Gray_Image4=mat2gray(Image4);
Gray_Image3=mat2gray(Image3);
Gray_Image2=mat2gray(Image2);
Gray_Image1=mat2gray(Image1);

%Normalize the source vectors
v1 = [16;19;30];
Norm_v1=v1/norm(v1,2);
v2 = [13;16;30];
Norm_v2=v2/norm(v2,2);
v3 = [-17;10.5;26.5];
Norm_v3=v3/norm(v3,2);
v4 = [9;-25;4];
Norm_v4=v4/norm(v4,2);

%stack the input vectors of the 4 images in another vector
V = [v2'; v3'; v4'];
```

```
%calculate the number of rows and columns of the images. It is the same
%value for the four images.
rows = size(Gray_Image4,1);
cols = size(Gray_Image4,2);

%initialize the g(x,y) vector where g(x,y)= albedo(x,y).N(x,y)
%initialzie the p , q and the normal Vector N
g = zeros(rows,cols,3);
albedo = zeros(rows,cols);
p = zeros(rows,cols);
q = zeros(rows,cols);
N = zeros(rows,cols, 3);

%running through the entire image we calculate the g(x,y) by solving the
%linear system of equations
for r = 1:rows;
for c = 1:cols;
  i = [ Gray_Image1(r,c); Gray_Image3(r,c); Gray_Image4(r,c)];
  Ivector = diag(i);
  A = Ivector * i;
  B = Ivector * V;
  rank_of_B=rank(B);
  if (rank_of_B<3)
     continue;
  end
      X = B\A;
      g(r,c,:) = X;
      albedo(r,c) = norm(X);
      N(r,c,:) = X/norm(X);
      p(r,c) = N(r,c,1)/N(r,c,3);
      q(r,c) = N(r,c,2)/N(r,c,3);

end
end

%To caculate the albedo of the given image whose values should be
%between 0-1
maximumAlbedo = max(max(albedo) );
if(maximumAlbedo > 0)
albedo = albedo/maximumAlbedo;
end

%To calculate Depth Map
depth=frankotchellappa(p,q);
%{
for i = 2:size(Image4,1)
for j = 2:size(Image4,2)
depth(i,j) = depth(i-1,j-1)+q(i,j)+p(i,j);
end
end
%}

figure(15);
surfl(-depth);
colormap(gray);
```

```
title('Mesh grid ');
grid off;
shading interp

%– Creating a new shaded image by choosing a new light source position vector and calculating
%the dot product of surface normals and light source vector (both normalized)
%at each pixel, resulting in a new image showing a light source not originally used for
%reconstruction

[ X, Y ] = meshgrid( 1:rows, 1:cols );
figure;
surf( X, Y, -depth, 'EdgeColor', 'none' );
camlight left;
title('Depth Map');
lighting phong

%Constructing the Needle Map which shows the normals pointing on the
%surface
figure
spacing = 1;
[X Y] = meshgrid(1:spacing:rows, 1:spacing:cols);
quiver3(X,Y,-depth,N(:,:,1),N(:,:,2),N(:,:,3))
hold on;
title('Needle Map');
hold off
```

Function: frankotchellappa()

```
function z = frankotchellappa(dzdx,dzdy)

  if ~all(size(dzdx) == size(dzdy))
    error('Gradient matrices must match');
  end

  [rows,cols] = size(dzdx);

  % The following sets up matrices specifying frequencies in the x and y
  % directions corresponding to the Fourier transforms of the gradient
  % data.  They range from -0.5 cycles/pixel to + 0.5 cycles/pixel. The
  % fiddly bits in the line below give the appropriate result depending on
  % whether there are an even or odd number of rows and columns

  [wx, wy] = meshgrid(([1:cols]-(fix(cols/2)+1))/(cols-mod(cols,2)), ...
        ([1:rows]-(fix(rows/2)+1))/(rows-mod(rows,2)));

  % Quadrant shift to put zero frequency at the appropriate edge
  wx = ifftshift(wx); wy = ifftshift(wy);

  DZDX = fft2(dzdx);   % Fourier transforms of gradients
  DZDY = fft2(dzdy);

  % Integrate in the frequency domain by phase shifting by pi/2 and
  % weighting the Fourier coefficients by their frequencies in x and y and
  % then dividing by the squared frequency.  eps is added to the
```

```
   % denominator to avoid division by 0.

   Z = (-j*wx.*DZDX -j*wy.*DZDY)./(wx.^2 + wy.^2 + eps);  % Equation 21

   z = real(ifft2(Z));  % Reconstruction
```

6.6 We can perform 3D surface reconstruction from the 4 images taken all four images together. We get a better result if we follow this approach.

MATLAB CODE FOR SURFACE RECONSTRUCTION WITH FOUR IMAGES.

```
%Take input of the image data
Img1 = double(imread('D:\NYU_SEM_2\Computer Vision\Assignment-3\synth-images\im1.png'));
Img2 = double(imread('D:\NYU_SEM_2\Computer Vision\Assignment-3\synth-images\im2.png'));
Img3 = double(imread('D:\NYU_SEM_2\Computer Vision\Assignment-3\synth-images\im3.png'));
Img4 = double(imread('D:\NYU_SEM_2\Computer Vision\Assignment-3\synth-images\im4.png'));

%Normalize the image
 NormalsImg1 =(Img1)./255;
 NormalsImg2 =(Img2)./255;
 NormalsImg3 =(Img3)./255;
 NormalsImg4 =(Img4)./255;
%stack the source vectors
   v1 = [0 0 1]';
   v2 = [-0.2 0 1]';
   v3 = [0.2 0 1]';
   v4 = [0 -0.2 1]';

%Combine all the Lightsource into a matrix
LightSource = [v1';v2'; v3'; v4'];

%Get the size of an image
[NRows, NCols] = size(NormalsImg1);

%Initialize G, Albedo, Normals and p,q
g = zeros(NRows,NCols,3);
albedo = zeros(NRows,NCols);
Normals = zeros(NRows,NCols, 3);
p =  zeros(NRows,NCols);
q =  zeros(NRows,NCols);

%Algorithm to calculate the Depth (Z)
for i = 1:NRows;
   for j = 1:NCols;

      %Compute small i
      iv = [NormalsImg1(i,j); NormalsImg2(i,j); NormalsImg3(i,j); NormalsImg4(i,j)]/255;

      %Compute I to Multiply on both side of equation i(x,y) = V * g(x,y)
      I = diag(iv);
      A = I * iv;
      B = I * LightSource;

      %Compute G, ALbedo(Rho) and Normals where Rho(x,y) = |g(x,y)| and N(x,y) = g(x,y)/|g(x,y)|
      temp = B\A;
      g(i,j,:) = temp;
```

```
        albedo(i,j) = norm(temp);
        Normals(i,j,:) = temp/norm(temp);

        %Compute P and Q where P = dz/dx and Q = dz/dy
        p(i,j) = Normals(i,j,1)/Normals(i,j,3);
        q(i,j) = Normals(i,j,2)/Normals(i,j,3);

    end
end

%Normalize the albedo
MaxAlbedo = max(max(albedo));
if( MaxAlbedo > 0)
    albedo = albedo/MaxAlbedo;
end

%Calculate the depth (Z values) using normal method
Depth=zeros(NRows,NCols);
for i = 2:size(Img1,1)
    for j = 2:size(Img1,1)
        Depth(i,j) = Depth(i-1,j-1)+q(i,j)+p(i,j);
    end
end

%Albedo Figure
figure(1);
imagesc(albedo);
figure(2)
imagesc(albedo);
colormap(gray);


%Depth gray image
figure(3);
surfl(-Depth);
colormap(gray);
grid off;
shading interp

%Normal vectors
[X, Y] = meshgrid( 1:NRows, 1:NCols );
figure(4);
quiver3(X,Y,-Depth, Normals(:,:,1),Normals(:,:,2),Normals(:,:,3))

%Wireframe of depth map
[X, Y] = meshgrid( 1:NRows, 1:NCols );
figure(5);
quiver3(X,Y,-Depth, Normals(:,:,1),Normals(:,:,2),Normals(:,:,3))
hold on;
surf( X, Y, -Depth, 'EdgeColor', 'none' );
camlight left;
lighting phong;
hold off;

%Calculate depth using Frankotchellappa function
```

```
FunctionalDepth = frankotchellappa(p,q);

%Depth gray image
figure(6);
surfl(-FunctionalDepth);
colormap(gray);
grid off;
shading interp

%Normal vectors
[X, Y] = meshgrid( 1:NRows, 1:NCols );
figure(7);
quiver3(X,Y,-FunctionalDepth, Normals(:,:,1),Normals(:,:,2),Normals(:,:,3))

%Wireframe of depth map
[X, Y] = meshgrid( 1:NRows, 1:NCols );
figure(8);
quiver3(X,Y,-FunctionalDepth, Normals(:,:,1),Normals(:,:,2),Normals(:,:,3))
hold on;
surf( X, Y, -FunctionalDepth, 'EdgeColor', 'none' );
camlight left;
lighting phong;
hold off;

end
```

CONCLUSION-

In this assignment, we learned important techniques from shape from shading. We reconstructed the shape of an object with Lambertian surface using multiple images where the light source position is changed and according to the object's reflecting properties, the albedo, the values of the source vectors are estimated. Theoretically we only need 3 images to solve our equation system and reconstruct the original object. But most of the time there are dark areas in the image where the intensity I(x,y)=0 which prevents us from getting complete information about the shape. The errors cause creation of 3D surfaces with errors which do not look perfect. We also noticed that with the integration technique that we used, a bunch of issues occurred. As we also mentioned, our integration method is simple and is based on normal integration along a curve. But this method does not really take in account some properties of the shape as differentiability. Indeed, an interesting property of shapes is, in most cases, to be differentiable and continuous. So, we need to enforce properties on the second derivative, which is not done in this implementation.
The checking constraint we used in our experiment is:

Assume that the measured value of the unit normal at some point (x, y) is $(a(x, y), b(x, y), c(x, y))$. Then

$$\frac{\partial f}{\partial x} = \frac{a(x,y)}{c(x,y)} \text{ and } \frac{\partial f}{\partial y} = \frac{b(x,y)}{c(x,y)}.$$

We have another check on our data set, because

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial^2 f}{\partial y \partial x},$$

so we expect that

$$\frac{\partial \left(\frac{a(x,y)}{c(x,y)} \right)}{\partial y} - \frac{\partial \left(\frac{b(x,y)}{c(x,y)} \right)}{\partial x}$$

should be small at each point. In principle it should be zero, but we would have to estimate these partial derivatives numerically and so should be willing to accept small values. This test is known as a test of *integrability*, which in vision applications always boils down to checking that mixed second partials are equal.

Thus, the final object surface was constructed using the techniques of shape from shading.