Assignment 2

November 11, 2017

1 CSE 252A Computer Vision I Fall 2017

1.1 Assigment 2

This assignment contains theoretical and programming exercises. If you plan to submit hand written answers for theoretical exercises, please be sure your writing is readable and merge those in order with the final pdf you create out of this notebook.

1.2 Problem 1: Irradiance [4 pts]

Consider a rectangular surface with vertices (-3,-2,1); (3,-2,1); (-3,2,1) and (3,2,1). The radiance on the surface is given by $L.(x^2 + y^2 + 3)$ where L is a constant.

What is the irradiance arriving at the position (0,0,0) with the normal vector (0,0,1)?

1.3 Solution

Irradiance to radiance:

$$E(x) = L(x, \theta, \phi)$$

Then we get

$$E(x) = L(x^2 + y^2 + 3)\cos\theta d\omega$$

$$d\omega = \frac{d_{A1} * \cos\theta}{r^2}$$

Then the expression for E(x) becomes $E(x) = L(x^2 + y^2 + 3) * \frac{\cos^2\theta}{r^2} dA_1$
Using trigonometry, we get $\cos^2\theta = \frac{1}{x^2+y^2+1}$ Thus the expression for E(x) becomes

$$E(x) = L(x^2 + y^2 + 3) * \frac{1}{(x^2 + y^2 + 1) * r^2} dA_1$$

This is the irradiance over a small patch dA_1, and the total irradiance by the rectangular area is the integral of E(x) over x = -3 to x = 3 and y = -2 to y -2. Therefore,

$$E(total) = \int_{-2}^{2}\int_{-3}^{3} \frac{L(x^2 + y^2 + 3)}{(x^2 + y^2 + 1) * r^2} dx$$

1.4 Problem 2: Irradiance [4 pts]

Consider a cylinder with radius r and height h whose base is centered at z=0 along the xy-plane. The walls of the cylinder have a constant radiance L and the top of the cylinder has constant radiance 2L.

What is the irradiance E at the point (0,0,0) assuming that the surface at (0,0,0) has a normal vector of (0,0,1)?

1.4.1 Solution

Part 1: irradiance from the top:

$$E_{top}(total) = \int_0^{2\pi} \int_0^{\tan^{-1}\frac{r}{h}} 2L\cos\theta\sin\theta d\theta d\phi = \pi L \int_0^{\tan^{-1}\frac{r}{h}} \sin 2\theta d2\theta = \pi L[1-\cos(2\tan^{-1}\frac{r}{h})]$$

Part 2: irradiance from the wall:

$$E_{wall}(total) = \int_0^{2\pi} \int_{\tan^{-1}\frac{r}{h}}^{\frac{\pi}{2}} L\cos\theta\sin\theta d\theta d\phi = \frac{1}{2}\pi L \int_{\tan^{-1}\frac{r}{h}}^{\frac{\pi}{2}} \sin 2\theta d2\theta = \frac{1}{2}\pi L[1+\cos(2\tan^{-1}\frac{r}{h})]$$

Total irradiance:

$$E = (total) = \frac{1}{2}\pi L[1+\cos(2\tan^{-1}\frac{r}{h})] + \pi L[1-\cos(2\tan^{-1}\frac{r}{h})] = \frac{1}{2}\pi L[3-\cos(2\tan^{-1}\frac{r}{h})]$$

1.5 Problem 3: Diffused Objects and Shading Fields [4 pts]

We see a diffuse sphere centered at the origin, with radius one and albedo ρ, in an orthographic camera, looking down the z-axis.

This sphere is illuminated by a distant point light source whose direction is (0,0,1). There is no other illumination.

Show that the shading field in the camera is $\rho\sqrt{1-x^2-y^2}$.

1.5.1 Solution

The surface is $s(x,y) = (x, y, \sqrt{1-x^2-y^2})$
The normal vector is:

$$V = \frac{\partial s(x,y)}{\partial x} \times \frac{\partial s(x,y)}{\partial y} = (\frac{x}{\sqrt{1-x^2-y^2}}, \frac{y}{\sqrt{1-x^2-y^2}}, 1)$$

Thus,the unit normal is:

$$N = \frac{V}{|V|} = (x, y, \sqrt{1-x^2-y^2})$$

and the shading must be

$$\rho N \cdot S(x)$$

which yields

$$L = \rho(x, y, \sqrt{1-x^2-y^2}) \cdot (0,0,1) = \rho\sqrt{1-x^2-y^2}$$

1.6 Problem 4: Occlusion, Umbra and Penumbra [2 pts]

We have a square area source and a square occluder, both paraller to a plane.

The edge length of the source is half that of the occluder, and they are vertically above one another with their centers aligned.

1). What is the shape of the umbra?

2). What is the shape of the outside boundary of the penumbra?

1.6.1 Solution

1) square

2) square

1.7 Problem 5: Photometric Stereo, Specularity Removal, Light Direction [20 pts]

The goal of this part of the assignment is to implement a couple of different algorithms that reconstruct a surface using the concept of photometric stereo.

Additionally, you will implement the specular removal technique of Mallick et al., which enables photometric stereo reconstruction of certain non-Lambertian materials.

You can assume a Lambertian reflectance function once specularties are removed, but the albedo is unknown and non-constant in the images.

Your program will take in multiple images as input along with the light source direction (and color when necessary) for each image.

Lastly, you will have to implement a code to calculate the light direction on images of specular objects.

1.7.1 Part 1: [6 pts]

Implement the photometric stereo technique described in Forsyth and Ponce 5.4 and the lecture notes.

Your program should have two parts:

a) Read in the images and corresponding light source directions, and estimate the surface normals and albedo map.

b) Reconstruct the depth map from the normals. You can first try the naive scanline-based shape by integration method described in the book. If this does not work well on real images, you can use the implementation of the Horn integration technique given below in horn_integrate function.

Try using only im1, im2 and im4 first. Display your outputs as mentioned below.

Then use all four images. (Most accurate).

For each of the above cases you must output:

1). The estimated albedo map

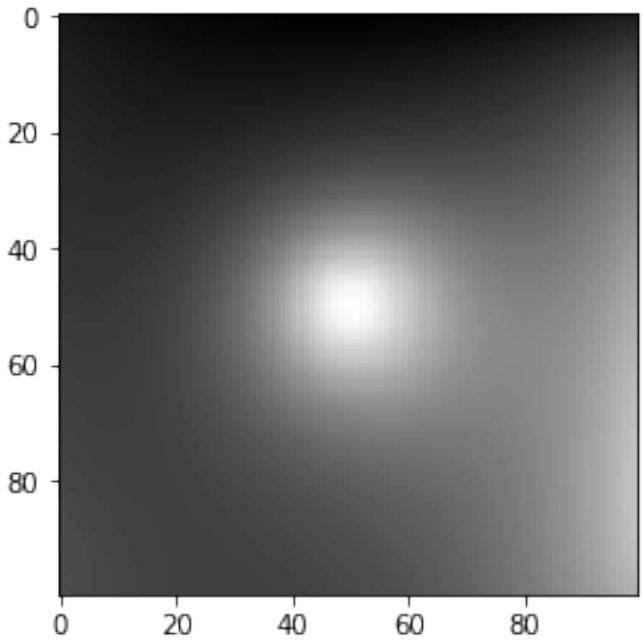
2). The estimated surface normals by either showing a). Needle map OR b). Three images showing components of surface normal

3). A wireframe of depth map

Note: You will find all the data for this part in synthetic_data.pickle

In [4]:
```
## Example: How to read and access data from a pickle
import pickle
import matplotlib.pyplot as plt

pickle_in = open("synthetic_data.pickle")
data = pickle.load(pickle_in)

#data is a dict which stores each element as a key-value pair.
print data.keys()

#To access the value of an entity, refer it by its key.
plt.imshow(data['im1'],cmap = 'gray')
plt.show()
```

```
['__version__', 'l4', '__header__', 'im1', 'im3', 'im2', 'l2', 'im4', 'l1', '__globals__', 'l3
```



In [5]:
```
import numpy as np
from scipy.signal import convolve
from numpy import linalg

def horn_integrate(gx,gy,mask,niter):
    '''
    integrate_horn recovers the function g from its partial
    derivatives gx and gy.
    mask is a binary image which tells which pixels are
```

```
        involved in integration.
        niter is the number of iterations.
        typically 100,000 or 200,000,
        although the trend can be seen even after 1000 iterations.
        '''
        g = np.ones(np.shape(gx))

        gx = np.multiply(gx,mask)
        gy = np.multiply(gy,mask)

        A = np.array([[0,1,0],[0,0,0],[0,0,0]]) #y-1
        B = np.array([[0,0,0],[1,0,0],[0,0,0]]) #x-1
        C = np.array([[0,0,0],[0,0,1],[0,0,0]]) #x+1
        D = np.array([[0,0,0],[0,0,0],[0,1,0]]) #y+1

        d_mask = A+B+C+D

        den = np.multiply(convolve(mask,d_mask,mode='same'),mask)
        den[den==0] = 1
        rden = 1.0/den
        mask2 = np.multiply(rden,mask)

        m_a = convolve(mask,A,mode='same')
        m_b = convolve(mask,B,mode='same')
        m_c = convolve(mask,C,mode='same')
        m_d = convolve(mask,D,mode='same')

        term_right = np.multiply(m_c,gx) + np.multiply(m_d,gy)
        t_a = -1.0*convolve(gx,B,mode='same')
        t_b = -1.0*convolve(gy,A,mode='same')
        term_right = term_right+t_a+t_b
        term_right = np.multiply(mask2,term_right)

        for k in range(niter):
            g = np.multiply(mask2,convolve(g,d_mask,mode='same'))\
                +term_right;

        return g
```

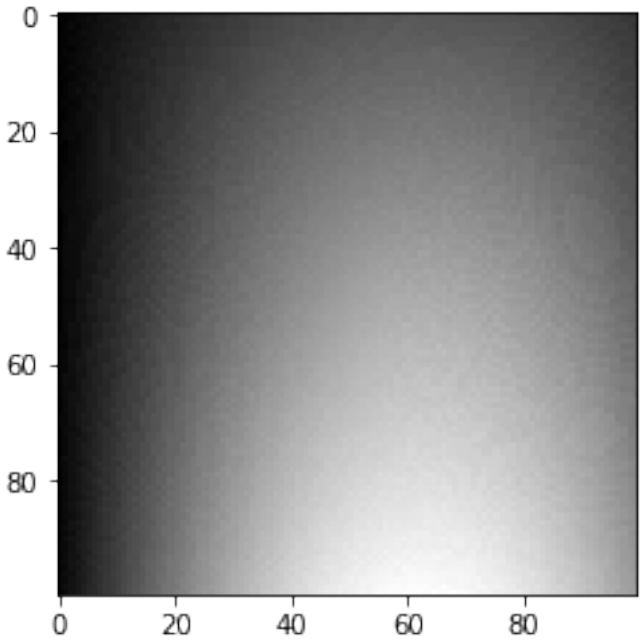
In [466]:
```
def photometric_stereo(images,lights):
    '''
    your implementaion
    '''
    img = np.array(images).T
    e = np.zeros((1,img.shape[2]))
    normals = np.zeros((np.shape(img)[0],np.shape(img)[1],3))
    albedo = np.zeros((np.shape(img)[0],np.shape(img)[1]))
    p = np.zeros((np.shape(img)[0],np.shape(img)[1]))
```

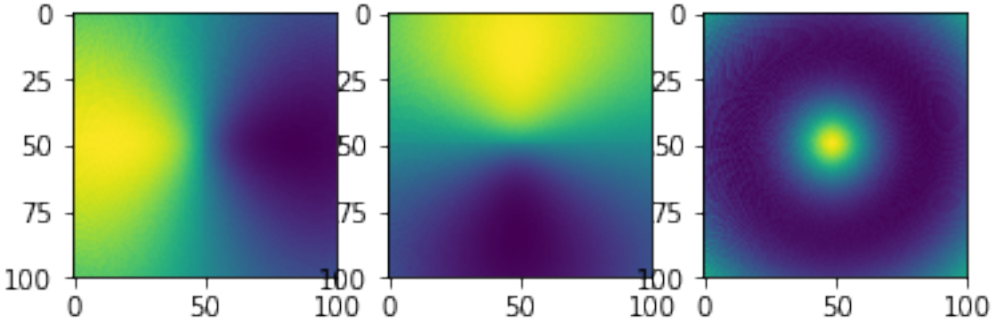
```
    q = np.zeros((np.shape(img)[0],np.shape(img)[1]))
    lights = lights.T
    for i in range(img.shape[2]):
        lights[2,i] = -lights[2,i]
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            for k in range(img.shape[2]):
                e[:,k] = float(img[i][j][k])
            A = np.dot(lights.T,lights)
            tmp = np.dot(linalg.pinv(A),lights.T)
            b = np.dot(e,tmp)
            albedo[i,j] = np.sqrt(np.dot(b,b.T))
            normals[i,j,:] = b / albedo[i][j]
            p[i,j] = normals[i,j,0]/float(normals[i,j,2])
            q[i,j] = normals[i,j,1]/float(normals[i,j,2])
    H = np.zeros((np.shape(img)[0],np.shape(img)[1]))
    H[0,0] = p[0,0]
    for i in range(1,img.shape[1]):
        H[0,i] = H[0,i-1] + p[0,i]
    for i in range(0,img.shape[1]):
        for j in range(1,img.shape[0]):
            H[j,i] = H[j-1,i] + q[j,i]
    mask = np.ones((np.shape(img)[0],np.shape(img)[1]))
    H_horn = horn_integrate(p,q,mask,10000)
    return albedo,normals,H,H_horn
```

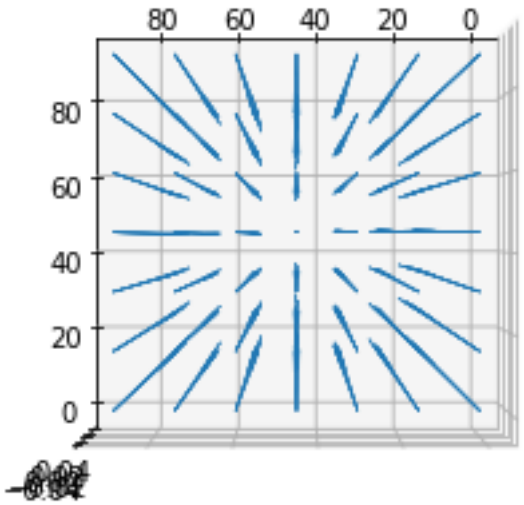
In [427]:
```
from mpl_toolkits.mplot3d import Axes3D

pickle_in = open("synthetic_data.pickle","rb")
data = pickle.load(pickle_in)

lights = np.vstack((data['l1'],data['l2'],data['l3'],data['l4']))

#lights = np.vstack((data['l1'],data['l2'],data['l4']))
#Hint: be careful about the light-source location and direction of light.
#lights right now stores light-source locations

images = []
images.append(data['im1'])
images.append(data['im2'])
images.append(data['im3'])
images.append(data['im4'])

albedo, normals, depth, horn = photometric_stereo(images, lights)

#-------------------------------------------------------------------------
#Following code is just a working example so you don't get stuck with any
#of the graphs required. You may want to write your own code to align the
```

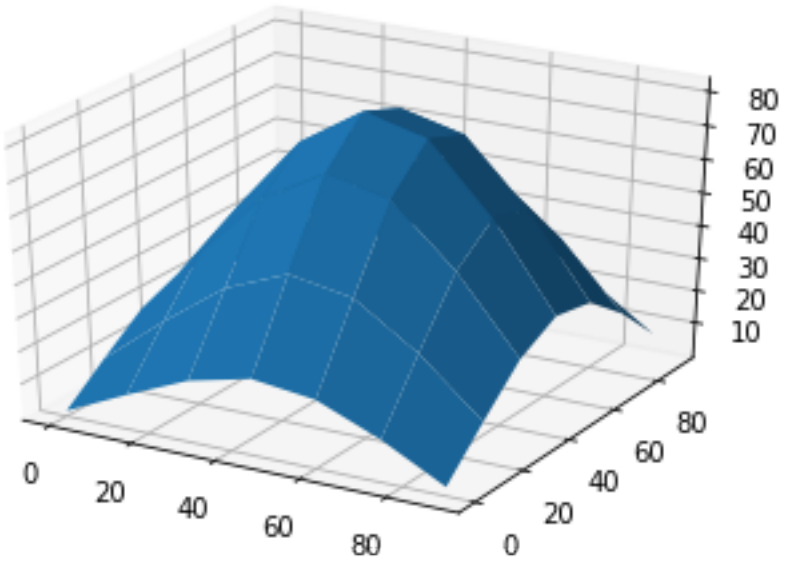
```
#results in a better layout
#------------------------------------------------------------------------

fig = plt.figure()
albedo_max = albedo.max()
albedo = albedo/albedo_max
plt.imshow(albedo, cmap='gray')
plt.show()

#showing normals as three seperate channels
figure = plt.figure()
ax1 = figure.add_subplot(131)
ax1.imshow(normals[..., 0])
ax2 = figure.add_subplot(132)
ax2.imshow(normals[..., 1])
ax3 = figure.add_subplot(133)
ax3.imshow(normals[..., 2])
plt.show()

#showing normals as quiver
X, Y, Z = np.meshgrid(np.arange(0,np.shape(normals)[0], 15), np.arange(0,np.shape(no
NX = normals[..., 0][::15,::15]
NY = normals[..., 1][::15,::15]
NZ = normals[..., 2][::15,::15]
fig = plt.figure(figsize=(5,5))
ax = fig.gca(projection='3d')
ax.view_init(azim=90, elev=-90)
plt.quiver(X,Y,Z,NX,NY,NZ, facecolor='r', linewidth=.5)
plt.show()

#plotting wireframe depth map
H = depth[::15,::15]
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(X[...,0],Y[...,0], H)
plt.show()

H = horn[::15,::15]
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(X[...,0],Y[...,0], H)
plt.show()
```

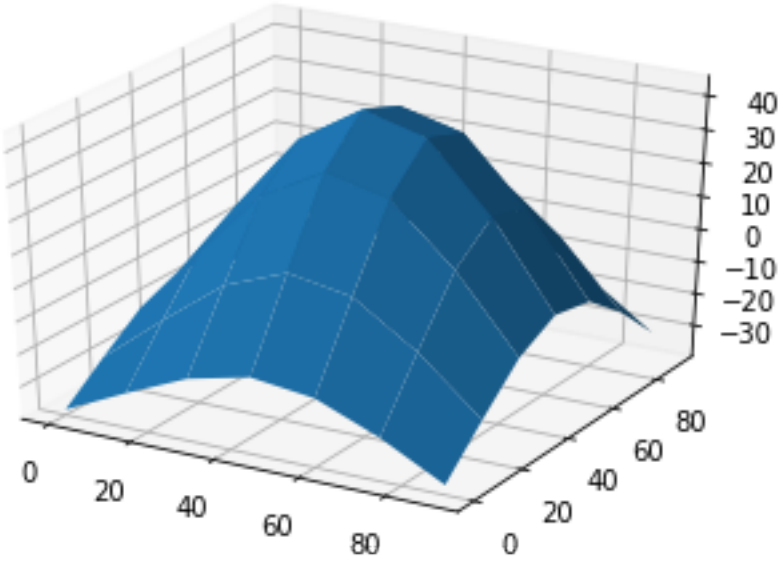
```
(3, 4)
```


0
20
40
60
80
0
20
40
60
80


0
25
50
75
100
0
50
100
0
25
50
75
100
0
50
100
0
25
50
75
100
0
50
100


80
60
40
20
0
80
60
40
20
0


80
70
60
50
40
30
20
10
0
20
40
60
80
0
20
40
60
80



1.7.2 Part 2: [4 pts]

Implement the specularity removal technique described in Beyond Lambert: Reconstructing Specular Surfaces Using Color (by Mallick, Zickler, Kriegman, and Belhumeur; CVPR 2005).

Your program should input an RGB image and light source color and output the corresponding SUV image.

Try this out first with the specular sphere images and then with the pear images.

For each specular sphere and pear images, include

1). The original image (in RGB colorspace).

2). The recovered S channel of the image.

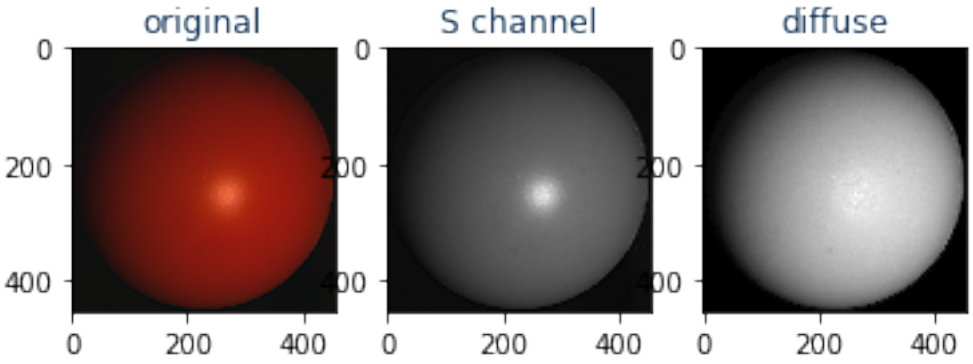
3). The recovered diffuse part of the image - Use $G = \sqrt{U^2 + V^2}$ to represent the diffuse part.

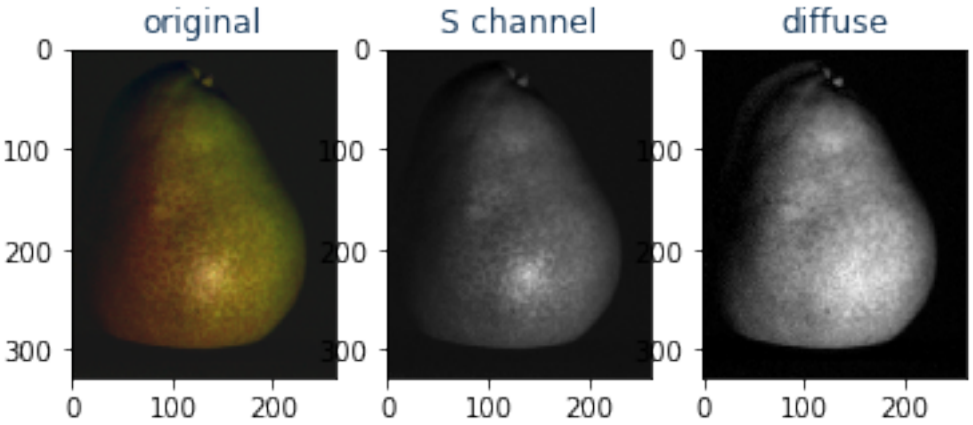
Note: You will find all the data for this part in specular_sphere.pickle and specular_pear.pickle

```
In [460]: def get_rot_mat(rot_v, unit=None):
              '''
              takes a vector and returns the rotation matrix required to align the unit vector
              '''
              if unit is None:
                  unit = [1.0, 0.0, 0.0]

              rot_v = rot_v/np.linalg.norm(rot_v)
              uvw = np.cross(rot_v, unit) #axis of rotation

              rcos = np.dot(rot_v, unit) #cos by dot product
              rsin = np.linalg.norm(uvw) #sin by magnitude of cross product

              #normalize and unpack axis
```

```
        if not np.isclose(rsin, 0):
            uvw = uvw/rsin
        u, v, w = uvw

        # Compute rotation matrix
        R = (
            rcos * np.eye(3) +
            rsin * np.array([
                [ 0, -w,  v],
                [ w,  0, -u],
                [-v,  u,  0]
            ]) +
            (1.0 - rcos) * uvw[:,None] * uvw[None,:]
        )

        return R

    def RGBToSUV(I_rgb,rot_vec):
        '''
        your implementation which takes an RGB image and a vector encoding the orientati
        '''
        I = (I_rgb - I_rgb.min()) / (I_rgb.max() - I_rgb.min())
        R = get_rot_mat(rot_vec)
        I_suv = np.zeros(np.shape(I_rgb))
        I_uv = np.zeros((np.shape(I_rgb)[0],np.shape(I_rgb)[1]))
        for i in range(I_rgb.shape[0]):
            for j in range(I_rgb.shape[1]):
                I_suv[i,j,:] = np.dot(R,[I_rgb[i,j,0],I_rgb[i,j,1],I_rgb[i,j,2]])
        for i in range(I_rgb.shape[0]):
            for j in range(I_rgb.shape[1]):
                I_uv[i,j] = np.sqrt(np.dot(I_suv[i,j,1],I_suv[i,j,1])+np.dot(I_suv[i,j,2

        plt.subplot(1,3,1)
        plt.title('original',color='#123456')
        plt.imshow(I)
        plt.subplot(1,3,2)
        plt.title('S channel',color='#123456')
        plt.imshow(I_suv[:,:,0],cmap='gray')
        plt.subplot(1,3,3)
        plt.title('diffuse',color='#123456')
        plt.imshow(I_uv,cmap='gray')
        plt.show()
        G = I_uv
        return (S,G)

In [461]: pickle_in = open("specular_sphere.pickle","rb")
          data = pickle.load(pickle_in)
          pickle_in1 = open("specular_pear.pickle","rb")
```

```
data1 = pickle.load(pickle_in1)
#sample input
S,G = RGBToSUV(data['im1'],np.hstack((data['c'][0][0],data['c'][1][0],data['c'][2][0]
S,G = RGBToSUV(data1['im1'],np.hstack((data1['c'][0][0],data1['c'][1][0],data1['c'][
```





1.7.3 Part 3: [3 pts]

Combine parts 1 and 2 by running your photometric stereo code on the diffuse components of the specular sphere and pear images.

For comparison, run your photometric stereo code on the original images (converted to grayscale) as well. You should notice erroneous "bumps" in the resulting reconstructions, as a result of violating the Lambertian assumption.

For each specular sphere and pear image sets, using all the four images, include:

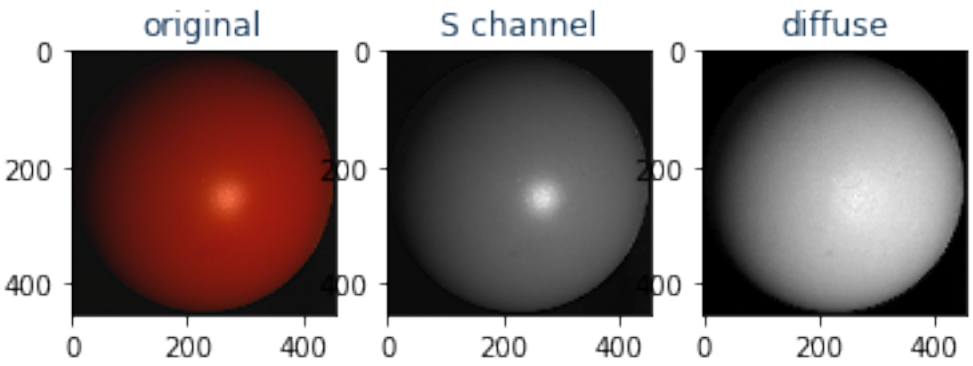
1). The estimated albedo map (original and diffuse)

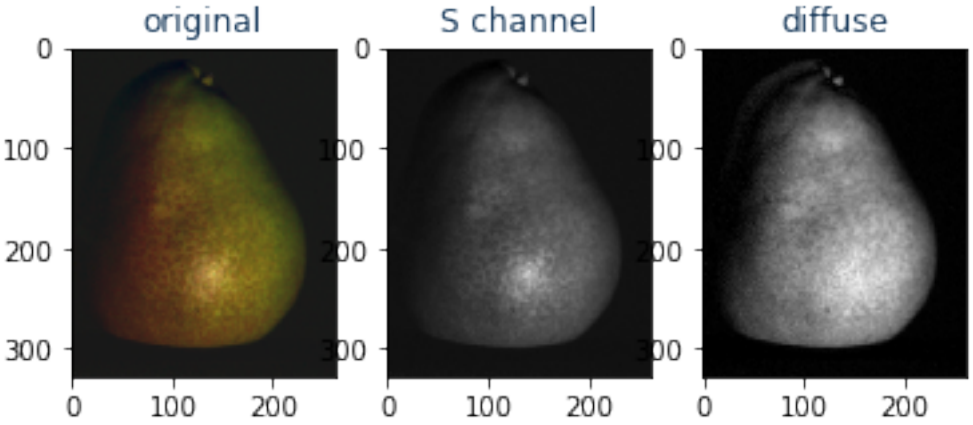
2). The estimated surface normals (original and diffuse) by either showing a). Needle map OR b). Three images showing components of surface normal

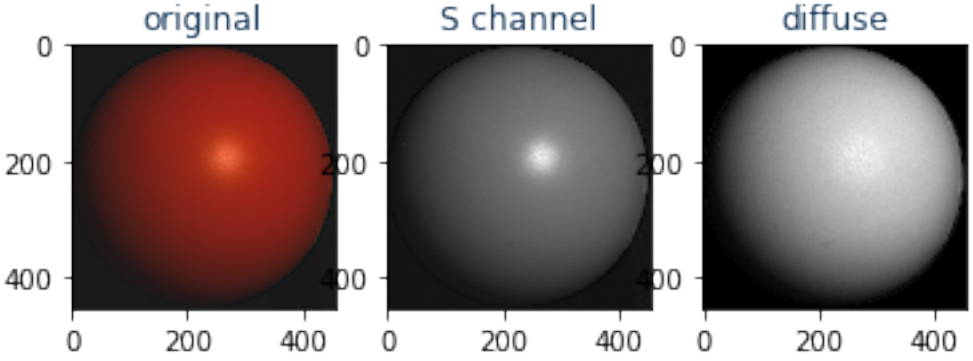
3). A wireframe of depth map (original and diffuse)

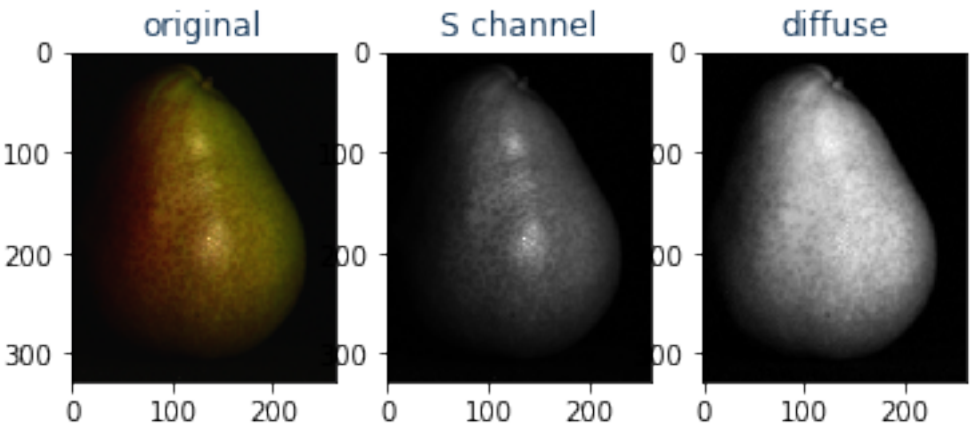
```
In [468]: def plott(albedo,normals,depth,horn):
              fig = plt.figure()
              albedo_max = albedo.max()
              albedo = albedo/albedo_max
              plt.imshow(albedo, cmap='gray')
              plt.show()

              #showing normals as three seperate channels
              figure = plt.figure()
              ax1 = figure.add_subplot(131)
              ax1.imshow(normals[..., 0])
              ax2 = figure.add_subplot(132)
              ax2.imshow(normals[..., 1])
              ax3 = figure.add_subplot(133)
              ax3.imshow(normals[..., 2])
              plt.show()

              #showing normals as quiver
              X, Y, Z = np.meshgrid(np.arange(0,np.shape(normals)[0], 15), np.arange(0,np.shape
              NX = normals[..., 0][::15,::15]
              NY = normals[..., 1][::15,::15]
              NZ = normals[..., 2][::15,::15]
              fig = plt.figure(figsize=(5,5))
              ax = fig.gca(projection='3d')
              ax.view_init(azim=90, elev=-90)
              plt.quiver(X,Y,Z,NX,NY,NZ, facecolor='r', linewidth=.5)
              plt.show()

              #plotting wireframe depth map
              H = depth[::15,::15]
              fig = plt.figure()
              ax = fig.gca(projection='3d')
              ax.plot_surface(X[...,0],Y[...,0], H)
              plt.show()

              H = horn[::15,::15]
              fig = plt.figure()
              ax = fig.gca(projection='3d')
              ax.plot_surface(X[...,0],Y[...,0], H)
              plt.show()

In [469]: def rgb2gray(rgb):
              r, g, b = rgb
              gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
              return gray

In [470]: #------------------------------------------------------------------------
          #you may reuse the code for photometric_stereo here
```

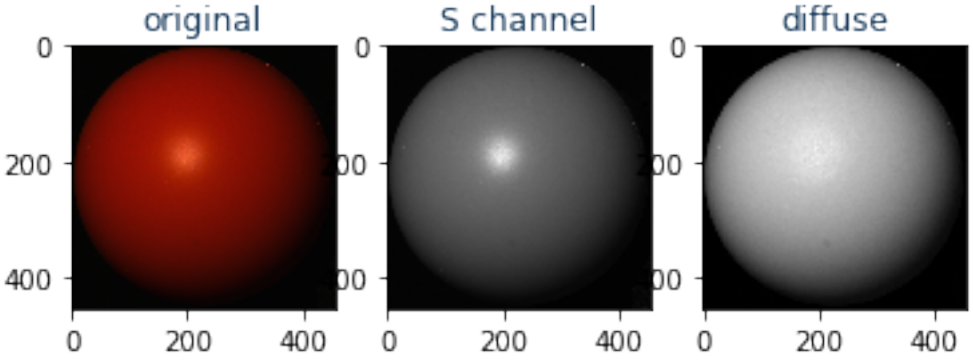
```
#write your code below to process the data and send it to photometric_stereo
#and display the albedo, normals and depth maps
#-------------------------------------------------------------------------
pickle_in = open("specular_sphere.pickle","rb")
data = pickle.load(pickle_in)
lights = np.vstack((data['l1'],data['l2'],data['l3'],data['l4']))
pickle_in1 = open("specular_pear.pickle","rb")
data1 = pickle.load(pickle_in1)
lights1 = np.vstack((data1['l1'],data1['l2'],data1['l3'],data1['l4']))
diffuse_1 = []
diffuse_2 = []
ori_1 = []
ori_2 = []
num = ['im1', 'im2', 'im3', 'im4']
for n in num:
    S,G = RGBToSUV(data[n],np.hstack((data['c'][0][0],data['c'][1][0],data['c'][2][0]
    diffuse_1.append(G)
    gray = np.zeros(data[n].shape[:2])
    for i in range(gray.shape[0]):
        for j in range(gray.shape[1]):
            gray[i][j] = rgb2gray(data[n][i][j])
    ori_1.append(gray)
    S,G = RGBToSUV(data1[n],np.hstack((data1['c'][0][0],data1['c'][1][0],data1['c'][2
    diffuse_2.append(G.T)
    gray = np.zeros(data1[n].shape[:2])
    for i in range(gray.shape[0]):
        for j in range(gray.shape[1]):
            gray[i][j] = rgb2gray(data1[n][i][j])
    ori_2.append(gray.T)
albedo, normals, depth, horn = photometric_stereo(diffuse_1, lights)
print "Sphere Diffuse"
plott(albedo,normals,depth,horn)
albedo, normals, depth, horn = photometric_stereo(ori_1, lights)
print "Sphere Original"
plott(albedo,normals,depth,horn)

albedo, normals, depth, horn = photometric_stereo(diffuse_2, lights1)
print "Pear Diffuse"
plott(albedo,normals,depth.T,horn.T)
albedo, normals, depth, horn = photometric_stereo(ori_2, lights1)
print "Pear Original"
plott(albedo,normals,depth.T,horn.T)
```

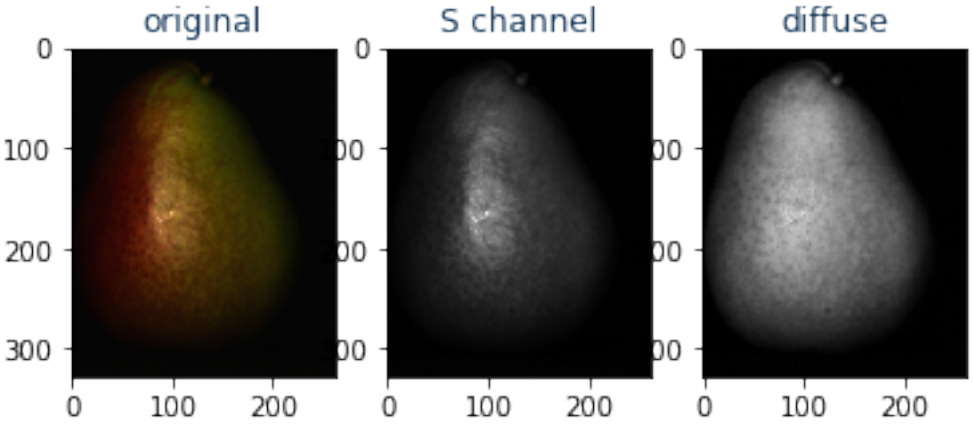

original
S channel
diffuse
0
200
400

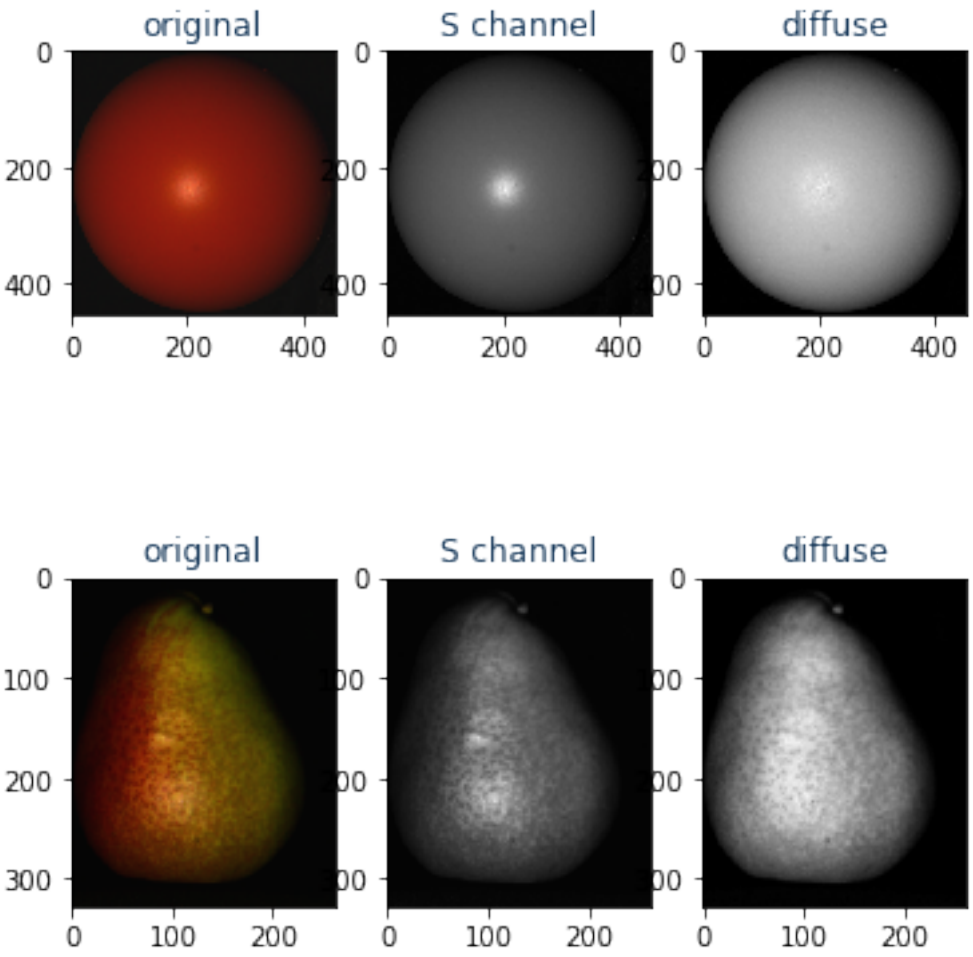

original
S channel
diffuse
0
100
200
300


original
S channel
diffuse
0
200
400

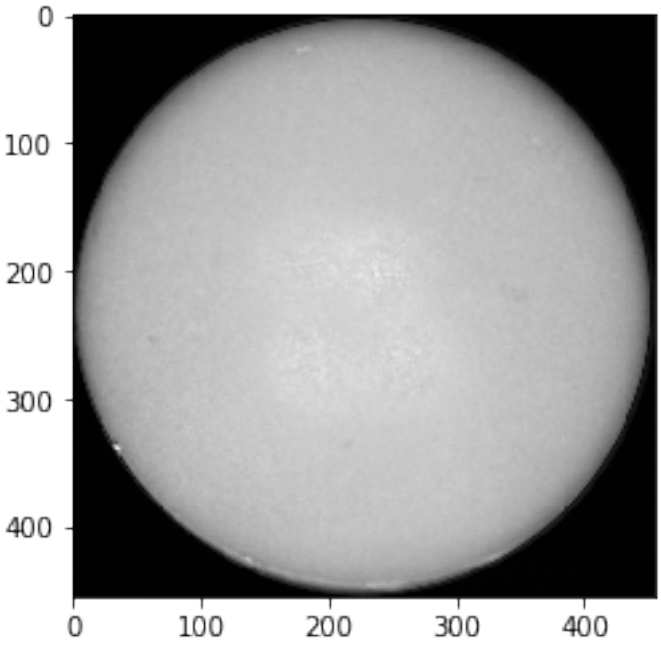

original
S channel
diffuse

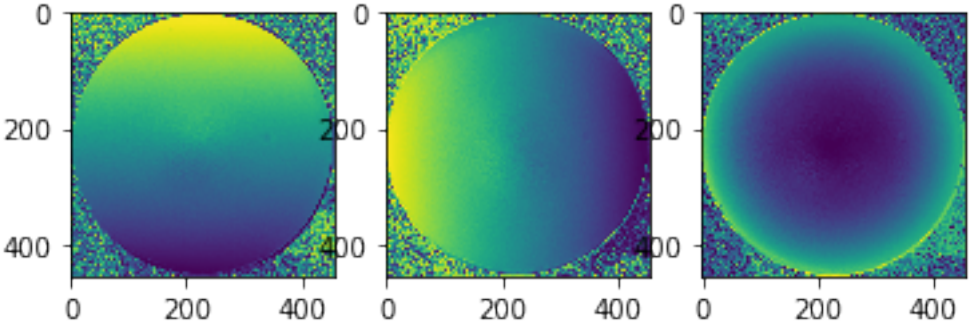

original
S channel
diffuse

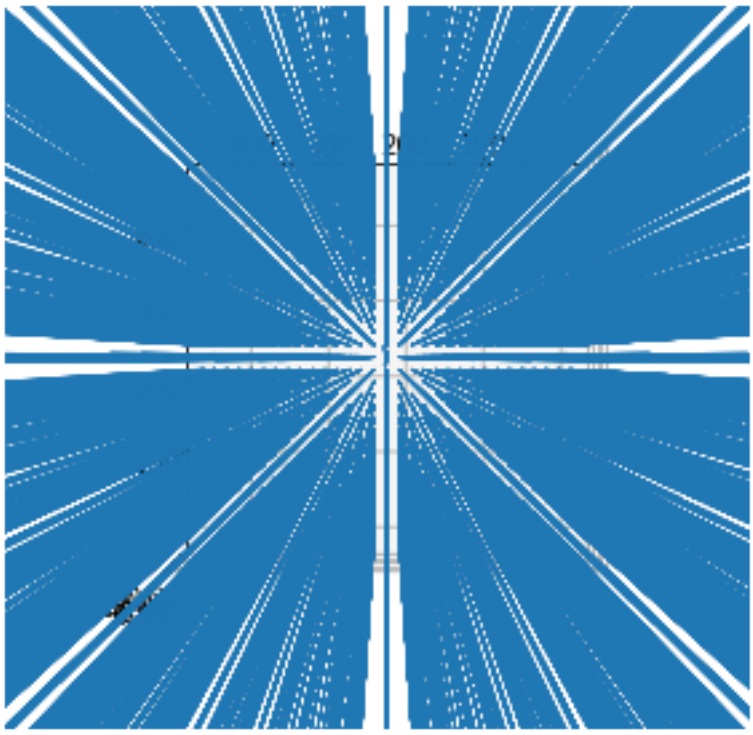

original
S channel
diffuse

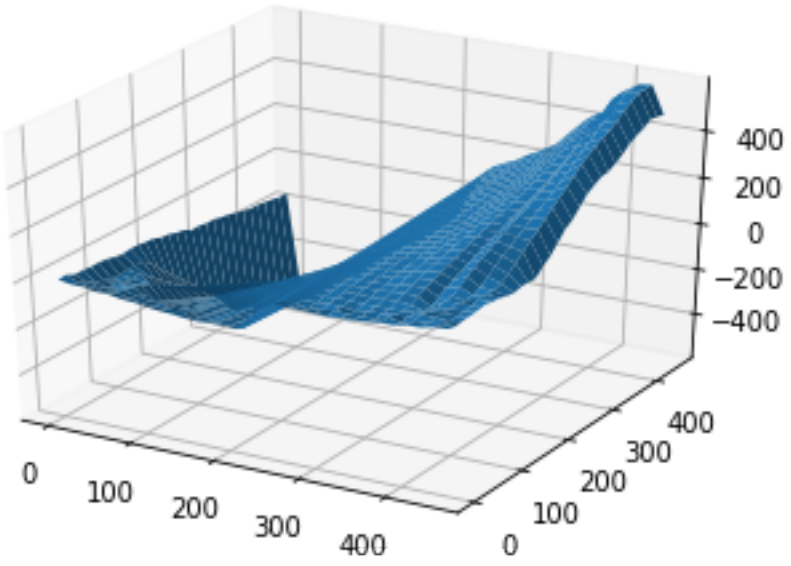


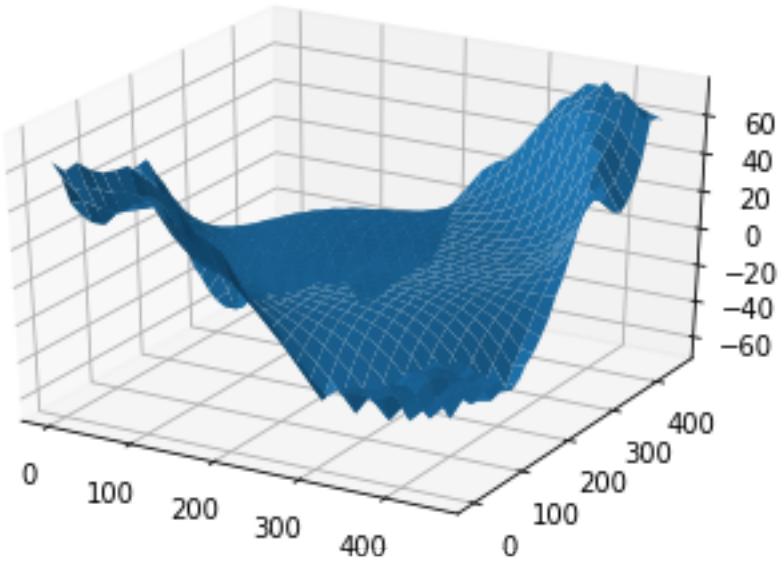
Sphere Diffuse


0
100
200
300
400
0
100
200
300
400

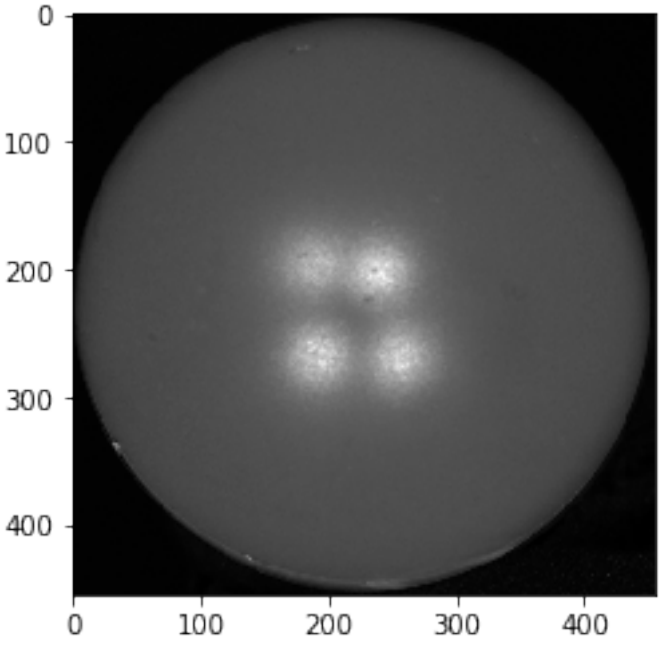

0
200
400
0
200
400
0
200
400
0
200
400
0
200
400
0
200
400

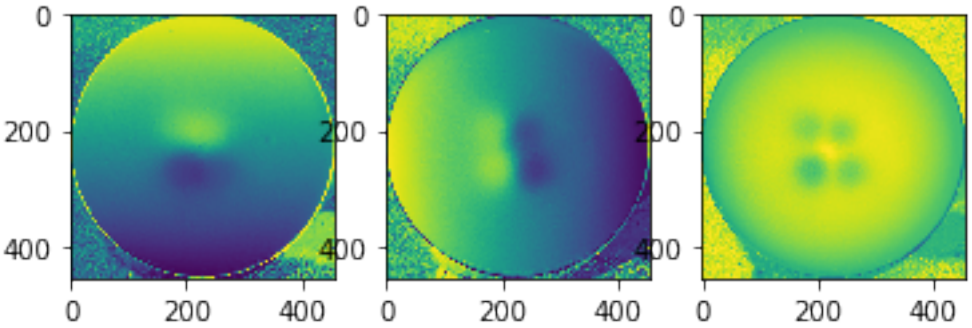


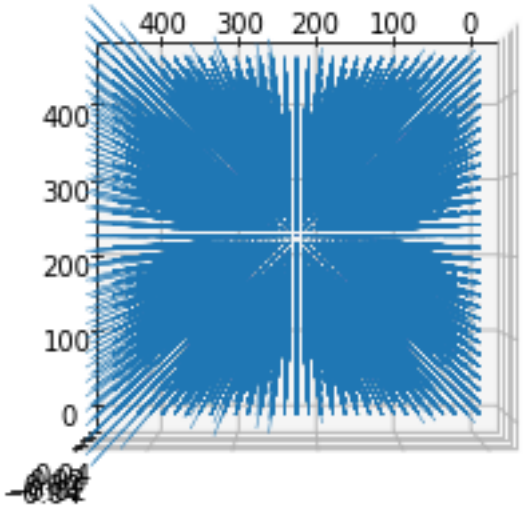

400
200
0
−200
−400
0
100
200
300
400
0
100
200
300
400

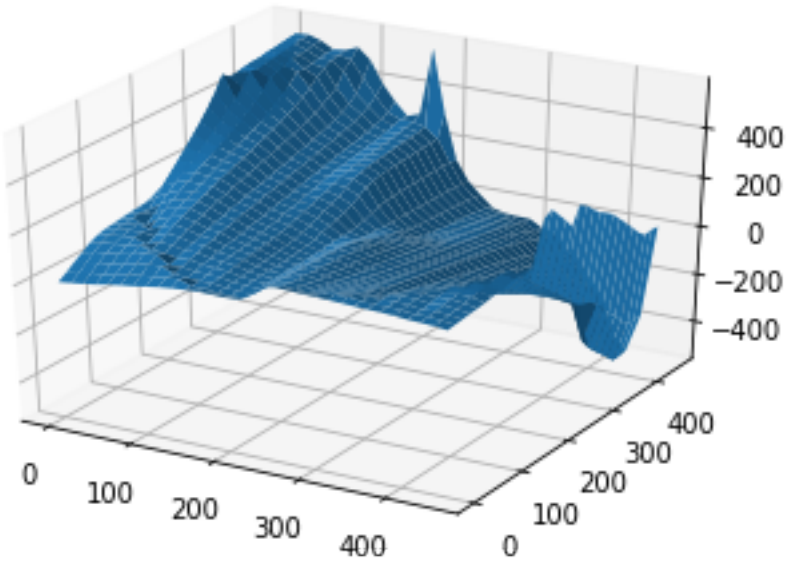


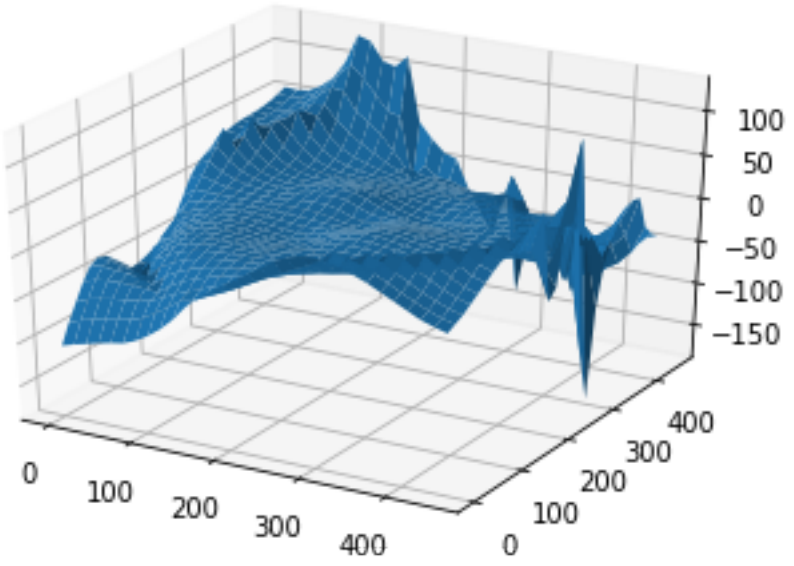
Sphere Original



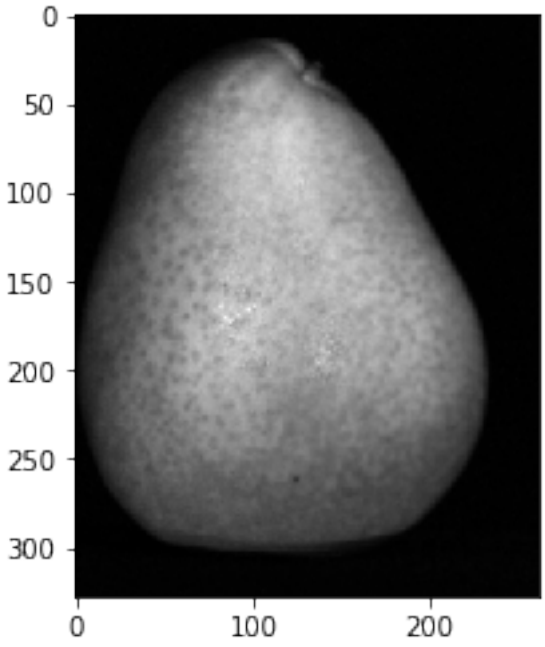

0
200
400
0
200
400
0
200
400
0
200
400
0
200
400
0
200
400

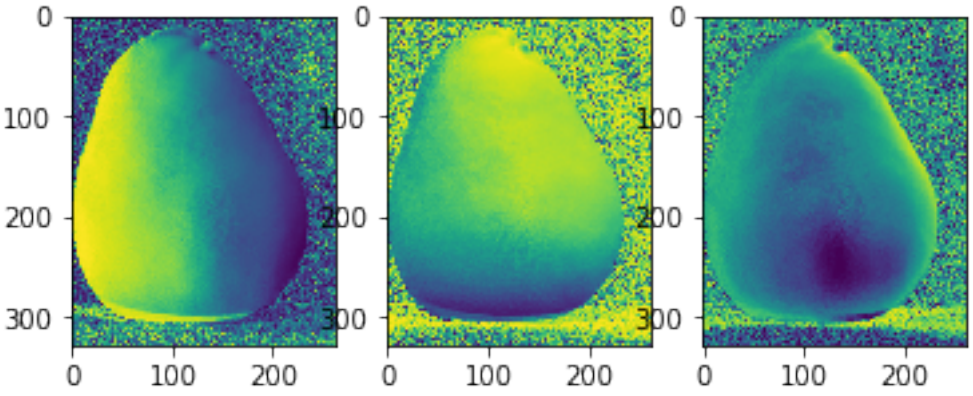

400
300
200
100
0
400
300
200
100
0

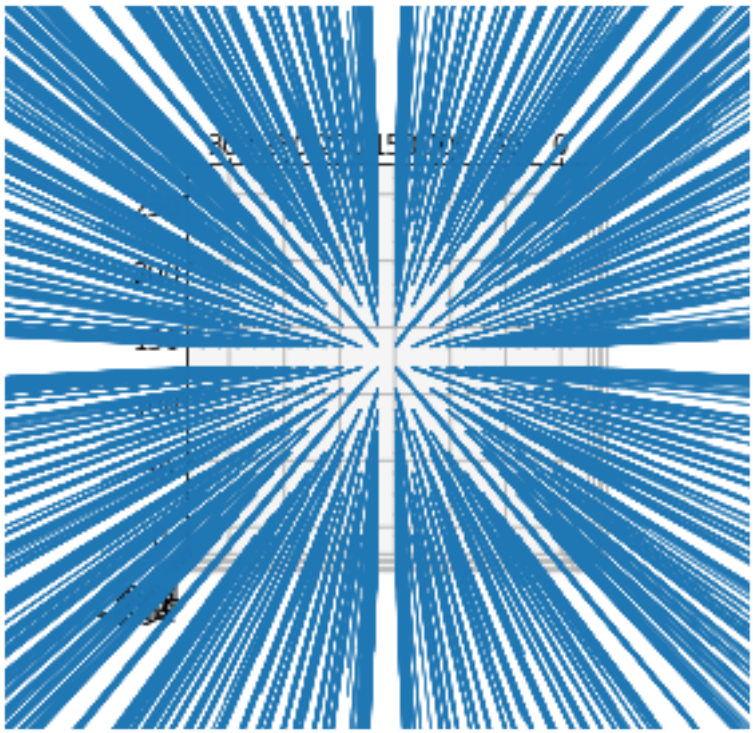


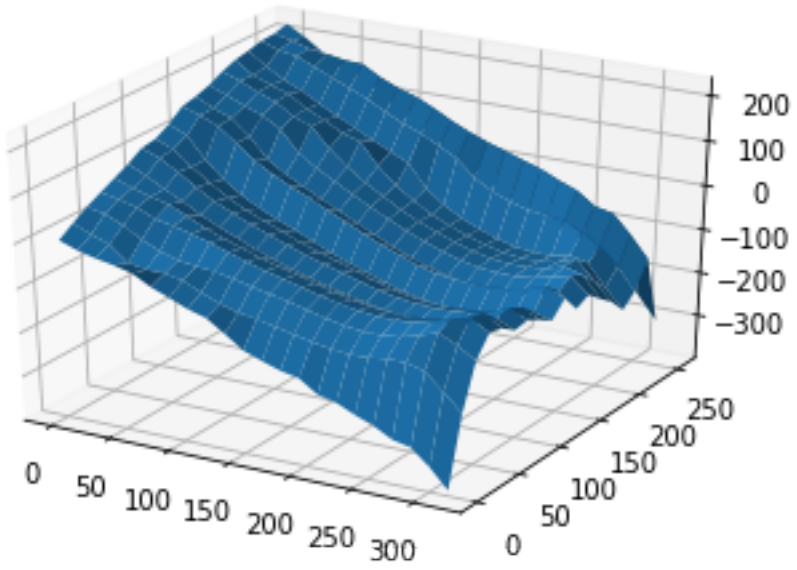


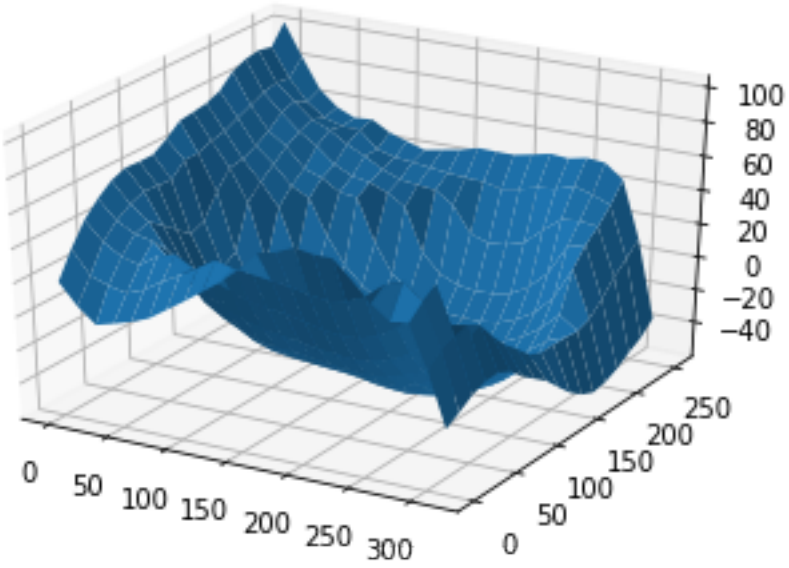
Pear Diffuse


0
50
100
150
200
250
300
0
100
200

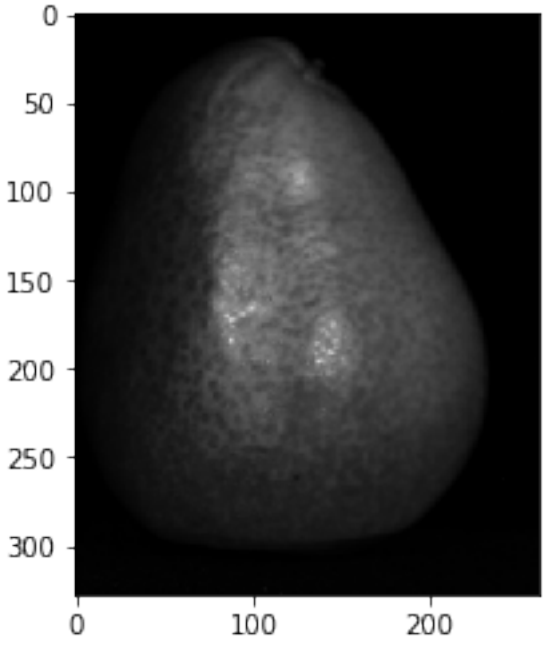

0
100
200
300
0
100
200
0
100
200
300
0
100
200
0
100
200
300
0
100
200

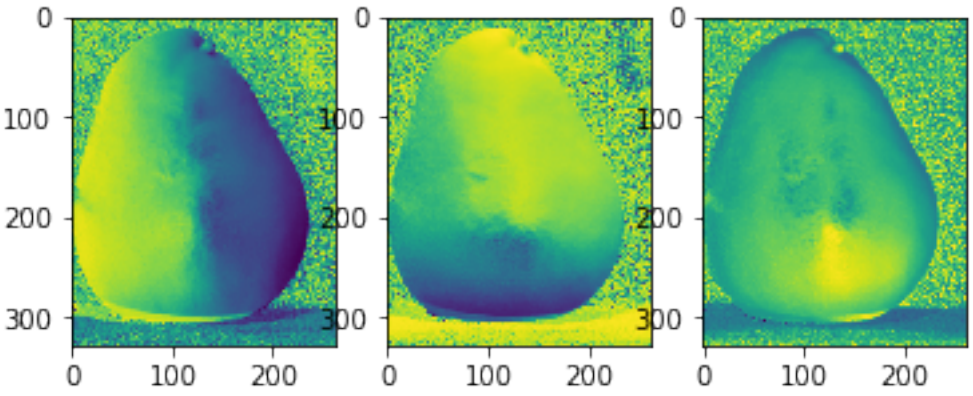


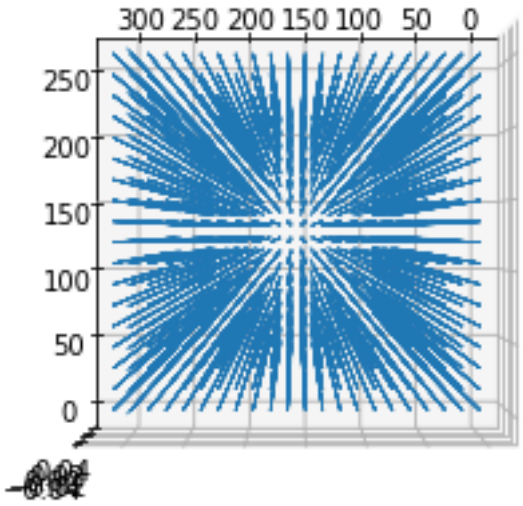

200
100
0
−100
−200
−300
0
50
100
150
200
250
300
0
50
100
150
200
250

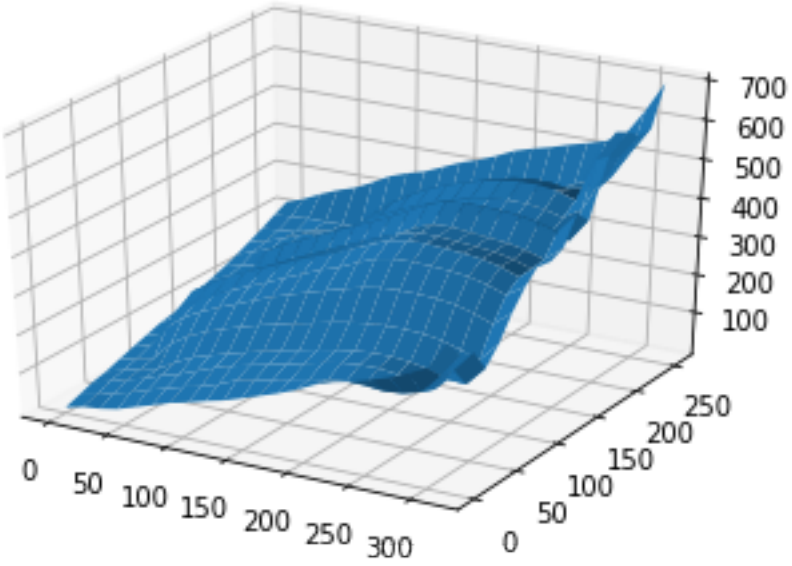


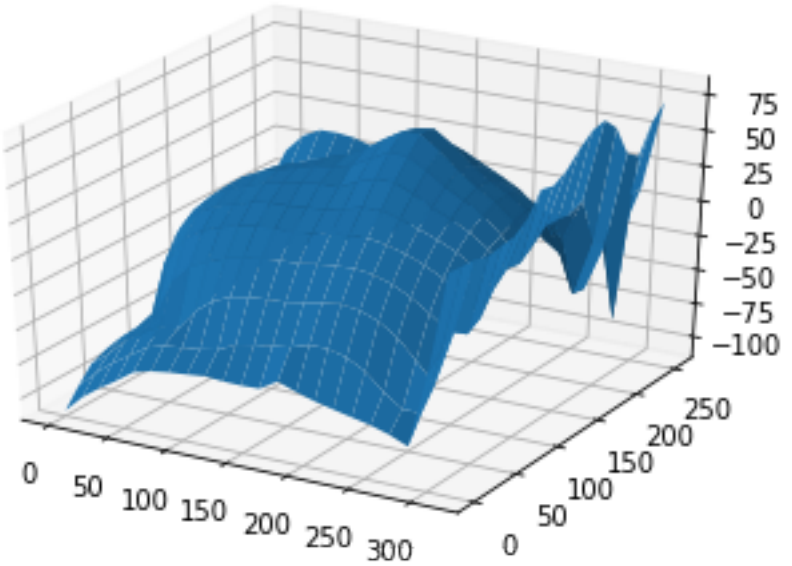
Pear Original




0
100
200
300
0
100
200


300 250 200 150 100 50 0
250
200
150
100
50
0





1.7.4 Part 4: [7 pts]

Often times the light direction is unknown. In these cases it is required to estimate the light direction using a highly specular ball.

a). Derive the equations/working-logic to find the light direction L, using the center of the ball, $C = (x, y)^T$, and the brightest point in the image $B = (x, y)^T$. You may find this equation

useful: $L = 2N(R.N) - R$ where R is the reflection direction and N is the surface normal. The camera is looking down the z-axis. You may submit a hand-written/typed derivation for this part OR make a seperate markdown cell for it and type it there.

b). Complete the function light_direction(), compute_center(), compute_brightest(), compute_radius() with a code that calculates the light direction in the 8 sphere images in sphere.zip.

c). Using all the images, include a wireframe of depth map

1.7.5 Solution

Assume the center of the ball as $C(C_x, C_y, C_z)$,the coordinate of the brightest point is $B(b_x, b_y, b_z)$
From $C = (x, y)^T$ we can get $C_x, C_y, C_z = 0$
The equation of a sphere with center $C(C_x, C_y, C_z)$ and radius r is:

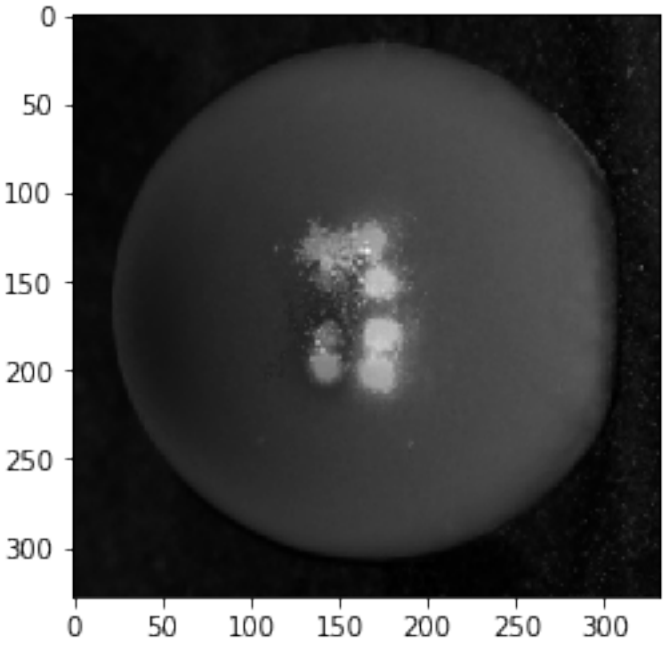
$$(b_x - C_x)^2 + (b_y - C_y)^2 + (b_z - C_z)^2 = r^2$$

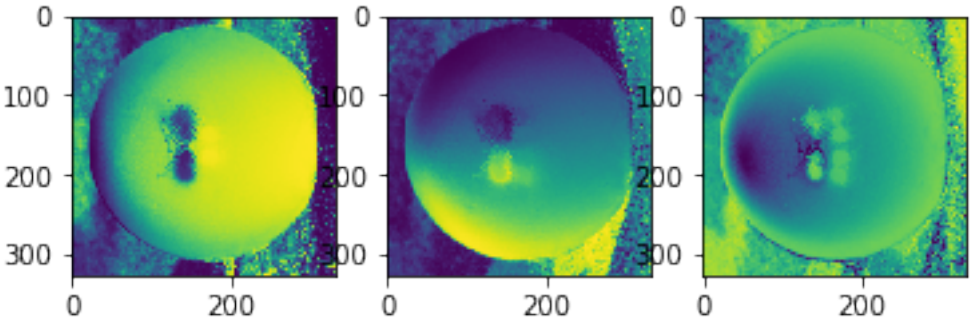
Therefore, $b_z = \sqrt{r^2 - (b_x - C_x)^2 - (b_y - C_y)^2}$
Thus, $N = [b_x - C_x, b_y - C_y, b_z]$
Consider camera is looking down the z-axis,$R = [0, 0, 1]$

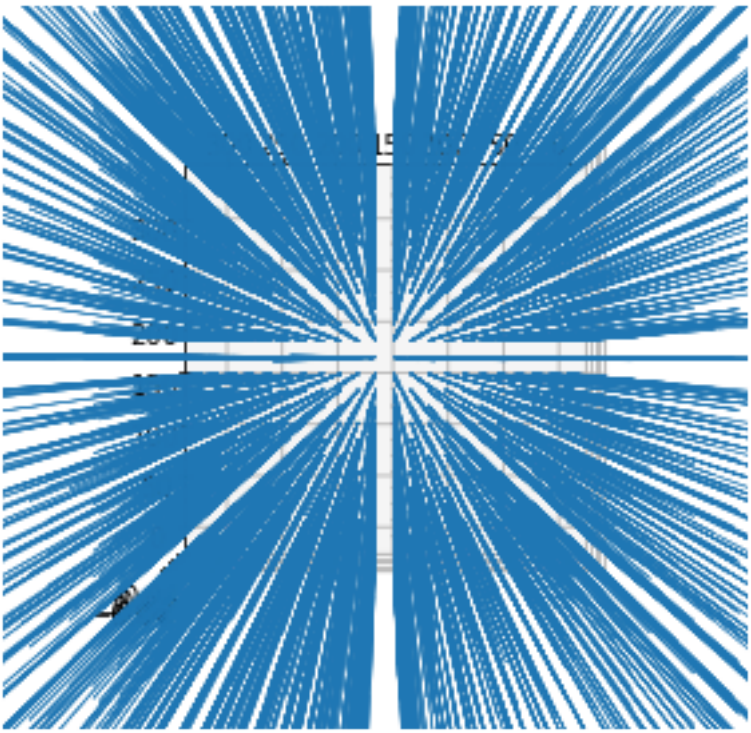
```
In [420]: def compute_radius(x_c, y_c, mask):
              r = 0
              c_1 = mask[x_c].tolist().count(255)
              c_2 = mask[y_c].tolist().count(255)
              r = (c_1+c_2)/4.0
              return r

          def compute_center(mask):
              x = 0
              y = 0
              r = [sum(row) for row in mask]
              c = [sum(row) for row in mask.T]
              row_left = 0
              row_right = 0
              col_up = 0
              col_bot = 0
              for i,rr in enumerate(r):
                  if rr>0 and row_left == 0:row_left = i
                  if rr>0 and row_left != 0:row_right = i
              for i,cc in enumerate(c):
                  if cc>0 and col_up == 0:col_up = i
                  if cc>0 and col_up != 0:col_bot = i
              x = (row_left + row_right)/2
              y = (col_up + col_bot)/2
              return x,y

          def compute_brightest(img):
              x = 0
              y = 0
```

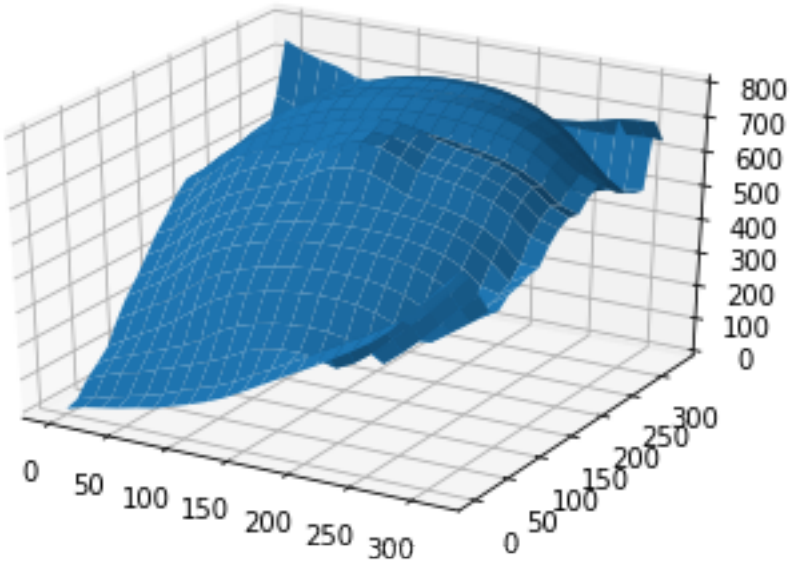
```
    bnw = np.zeros(img.shape)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if img[i][j] == 255:
                bnw[i][j] = 255
            else: bnw[i][j]=0
    x,y=compute_center(bnw)
    return x,y

def light_direction(img, mask):
    R = np.array([0, 0, 0])
    cx,cy = compute_center(mask)
    r = compute_radius(cx,cy,mask)
    bx,by = compute_brightest(img)
    N = np.zeros(3)

    bz = np.sqrt(r**2 - (bx-cx)**2 - (by-cy)**2) #x^2+y^2+z^2=r^2
    N = np.array([bx-cx, by-cy, bz])
    R = np.array([0,0,1])

    L = 2*np.dot(N, R)*N - R
    L = L/np.linalg.norm(L)
    x = L[0]
    y = L[1]
    z = L[2]
    return x,y,z
```

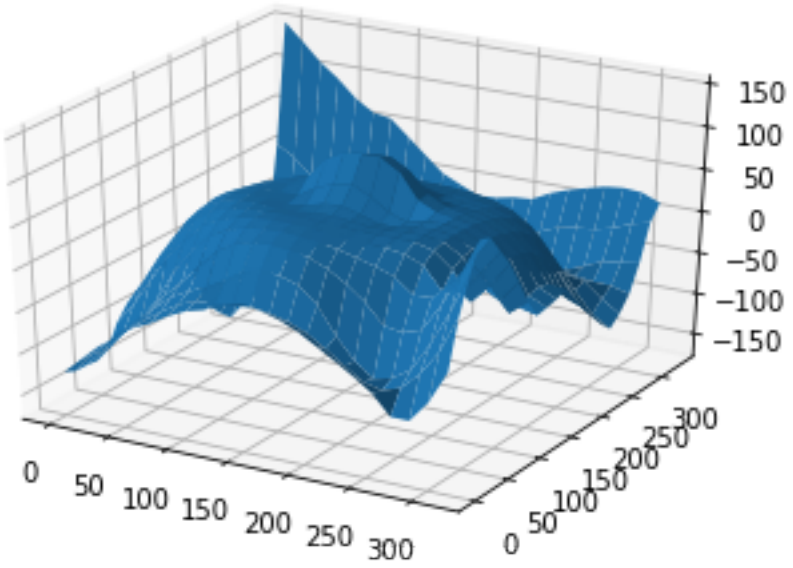
In [476]:
```
from scipy.misc import imread
images = []
lights = np.zeros([8,3])
for i in range(8):
    img = imread('./sphere/sphere' + str(i+1) + '.png',flatten=True)
    images.append(img)
    light = np.array(light_direction(img, msk))
    lights[i,:] = np.array([light[0],light[1],light[2]])
albedo, normals, depth, horn = photometric_stereo(images, lights)
plott(albedo,normals,depth.T,horn.T)
#####Do above for all the 8 images#####
#####Code to get the depth map to be written by self#####
```


0
50
100
150
200
250
300
0
50
100
150
200
250
300


0
100
200
300
0
200
0
100
200
300
0
200
0
100
200
300
0
200




800
700
600
500
400
300
200
100
0
0
50
100
150
200
250
300
0
50
100
150
200
250
300


150
100
50
0
−50
−100
−150
0
50
100
150
200
250
300
0
50
100
150
200
250
300