HW2 Coding Problems Solution

November 12, 2018

1 Problem 5: Photometric Stereo, Specularity Removal [14 pts]

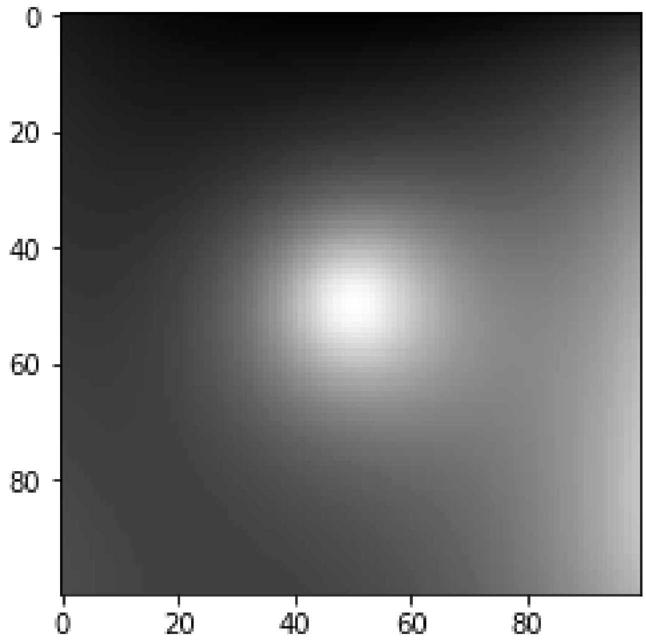
1.1 Part 1: [6 pts]

For each of the above cases you must output:

1. The estimated albedo map.
2. The estimated surface normals by showing both
   1. Needle map, and
   2. Three images showing components of surface normal.
3. A wireframe of depth map.

In [3]:
```
## Example: How to read and access data from a pickle
import pickle
import matplotlib.pyplot as plt
%matplotlib inline
pickle_in = open("synthetic_data.pickle", "rb")
data = pickle.load(pickle_in)
# data = pickle.load(pickle_in, encoding="latin1")

# data is a dict which stores each element as a key-value pair.
print("Keys: " + str(data.keys()))

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

print("Light source direction: " + str(data["l1"]))
```

```
Keys: ['l3', '__header__', '__globals__', 'im1', 'l4', 'im2', 'l2', 'im3', 'l1', '__version__'
Image:
```




```
Light source direction: [[0 0 1]]
```

```
In [4]: import numpy as np
        from scipy.signal import convolve
        from numpy import linalg

        def horn_integrate(gx, gy, mask, niter):
            '''
            horn_integrate 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 [5]: def photometric_stereo(images, lights, mask, horn_niter=100000):
        '''
        your implementaion
        '''
        num, height, width = images.shape
        # g = np.zeros((height, width, 3))
        normals = np.zeros((height, width, 3))
        albedo = np.zeros((height, width))
        gx = np.zeros((height, width))
        gy = np.zeros((height, width))
        V = lights
        Vplus = np.linalg.inv(V.T.dot(V)).dot(V.T)
        for x in range(height):
            for y in range(width):
                i = images[:, x, y]
                g = Vplus.dot(i)
                albedo[x, y] = np.linalg.norm(g)
                normals[x, y, :] = g / albedo[x, y]
                if mask[x, y]:
                    gx[x, y] = normals[x, y, 0] / normals[x, y, 2]
                    gy[x, y] = normals[x, y, 1] / normals[x, y, 2]

        H = np.zeros((height, width))
        for x in range(1, height):
```

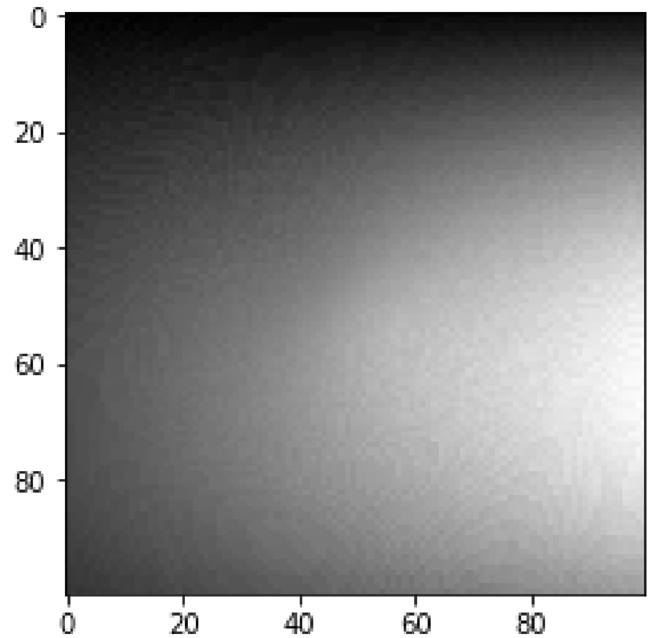
```
        H[x, 0] = H[x - 1, 0] + gy[x, 0]
    for y in range(1, width):
        H[:, y] = H[:, y - 1] + gx[:, y]

    H_horn = horn_integrate(gx,gy,mask,horn_niter)

    return albedo, normals, H, H_horn, gx, gy
```

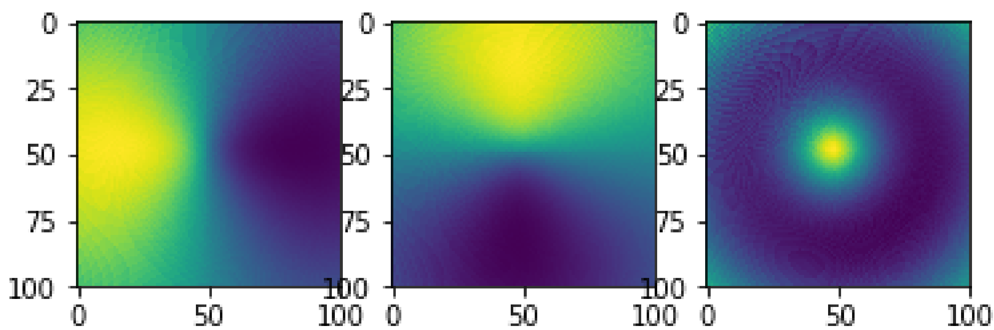
In [19]:
```
# --------------------------------------------------------------------------
# Following code is just a working example so you don't get stuck with any
# of the graphs required. You may want to write your own code to align the
# results in a better layout.
# --------------------------------------------------------------------------
from mpl_toolkits.mplot3d import Axes3D

def plot_all(albedo, normals, depth, horn, mask=None, stride=15, length=10):

    # showing albedo map
    fig = plt.figure()
    albedo_max = albedo.max()
    albedo = albedo / albedo_max
    plt.imshow(albedo, cmap="gray")
    plt.show()

    # showing normals as three separate 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, _ = np.meshgrid(np.arange(0,np.shape(normals)[0], 15),
                          np.arange(0,np.shape(normals)[1], 15),
                          np.arange(1))
    X = X[..., 0]
    Y = Y[..., 0]
    Z = horn[::stride,::stride].T
    if type(mask) != type(None):
        normals *= np.expand_dims(mask, axis=2)
    NX = normals[..., 0][::stride,::-stride].T
    NY = normals[..., 1][::-stride,::stride].T
    NZ = normals[..., 2][::stride,::stride].T
    fig = plt.figure(figsize=(5, 5))
    ax = fig.gca(projection='3d')
```

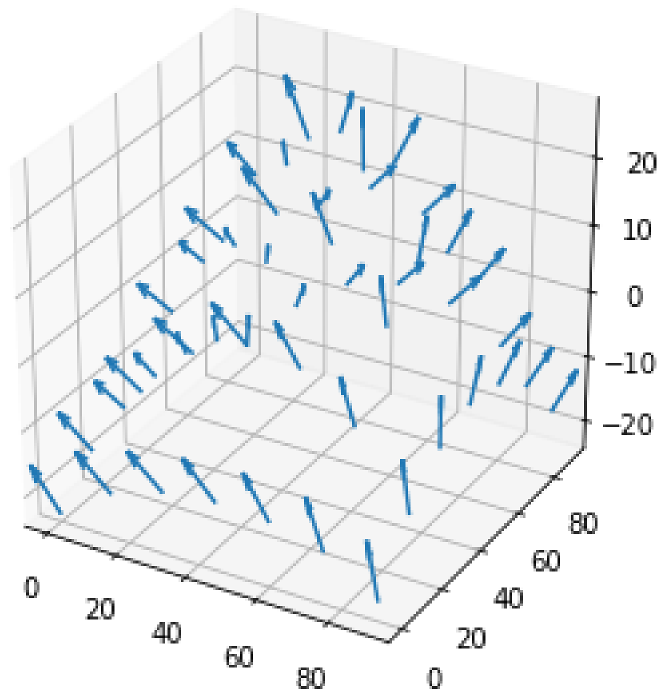
```
    plt.quiver(X,Y,Z,NX,NY,NZ, length=length)
    plt.show()

    # plotting wireframe depth map
    H = depth[::stride,::stride]
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    ax.plot_surface(X,Y, H.T)
    plt.show()

    H = horn[::stride,::stride]
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    ax.plot_surface(X,Y, H.T)
    plt.show()
```

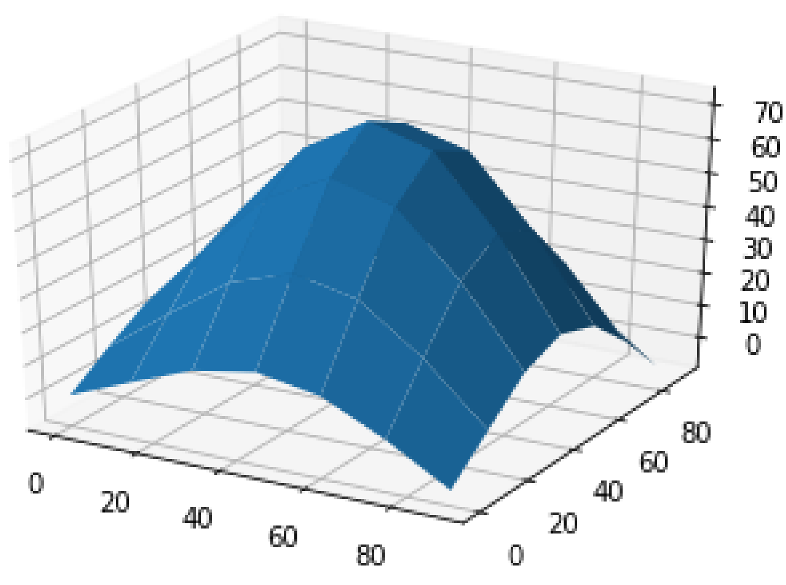
In [51]:
```
from mpl_toolkits.mplot3d import Axes3D

pickle_in = open("synthetic_data.pickle", "rb")
data = pickle.load(pickle_in)
# data = pickle.load(pickle_in, encoding="latin1")

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

images = []
images.append(data["im1"])
images.append(data["im2"])
# images.append(data["im3"])
images.append(data["im4"])
images = np.array(images)

mask = np.ones(data["im1"].shape)

albedo, normals, depth, horn, gx, gy = photometric_stereo(images, lights, mask, horn_
```

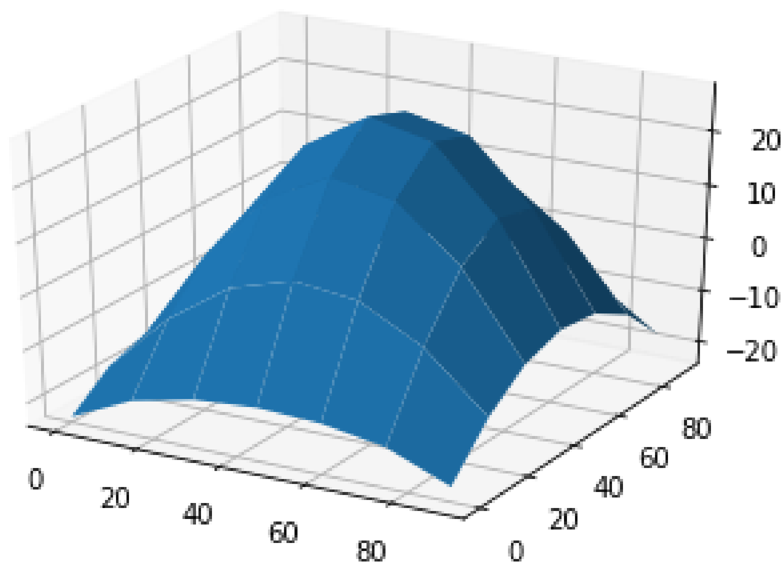
In [52]:
```
plot_all(albedo, normals, depth, horn)
```


0
20
40
60
80
0
20
40
60
80


0
25
50
75
100
0
50
100

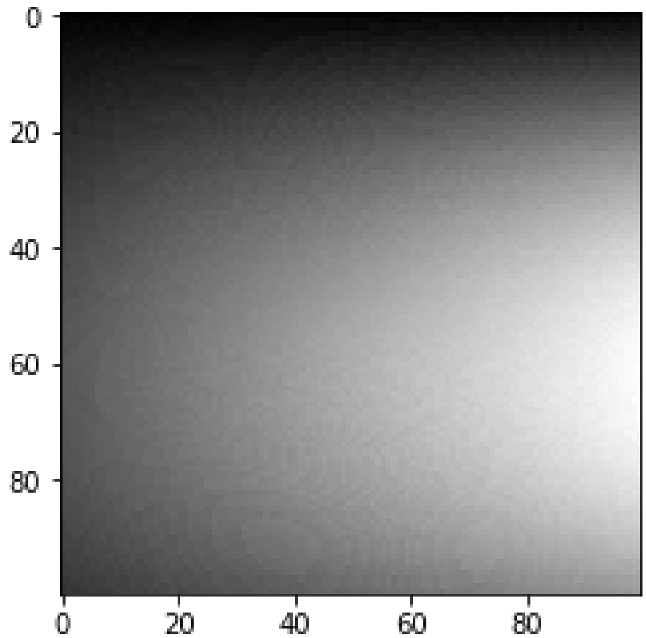

20
10
0
−10
−20
0
20
40
60
80
0
20
40
60
80

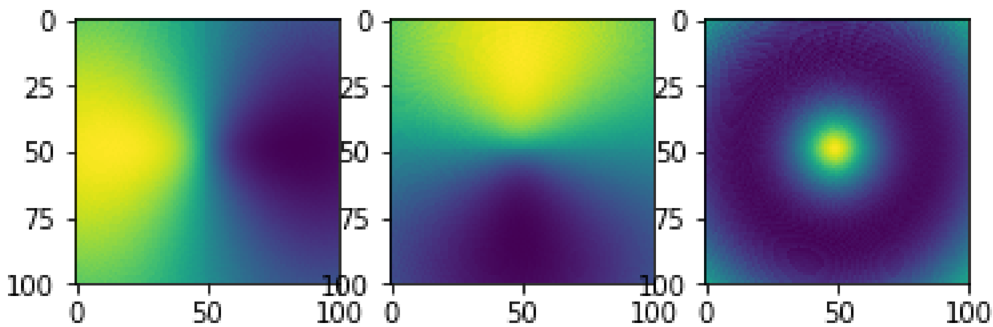

70
60
50
40
30
20
10
0
0
20
40
60
80
0
20
40
60
80

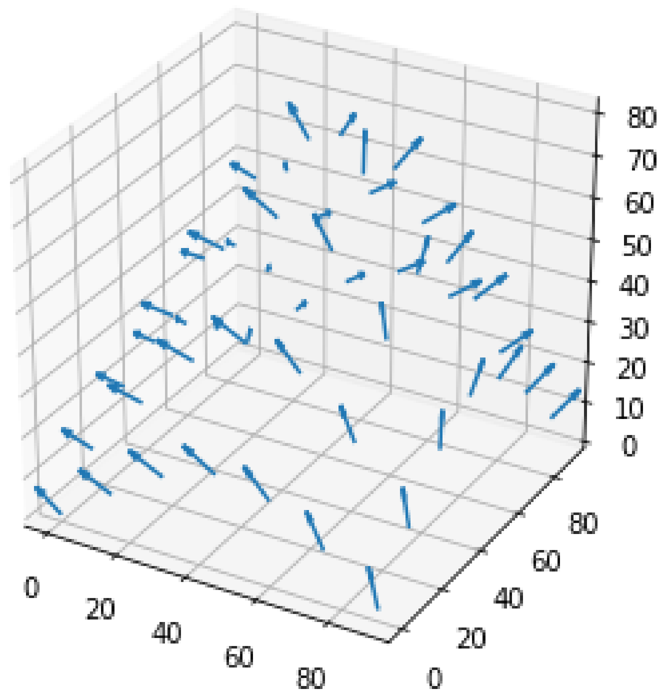


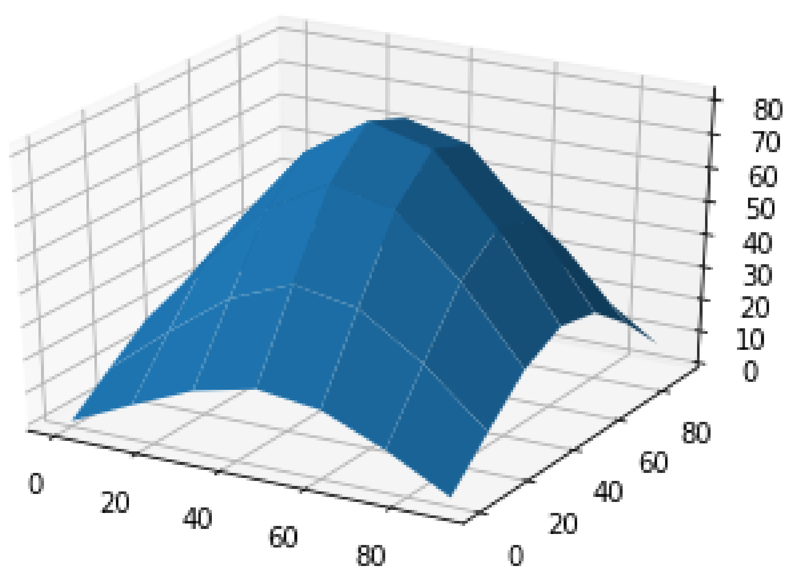

```
In [12]: from mpl_toolkits.mplot3d import Axes3D

         pickle_in = open("synthetic_data.pickle", "rb")
         data = pickle.load(pickle_in)
         # data = pickle.load(pickle_in, encoding="latin1")

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

         images = []
         images.append(data["im1"])
         images.append(data["im2"])
         images.append(data["im3"])
         images.append(data["im4"])
         images = np.array(images)

         mask = np.ones(data["im1"].shape)

         albedo, normals, depth, horn, gx, gy = photometric_stereo(images, lights, mask, horn_r
```

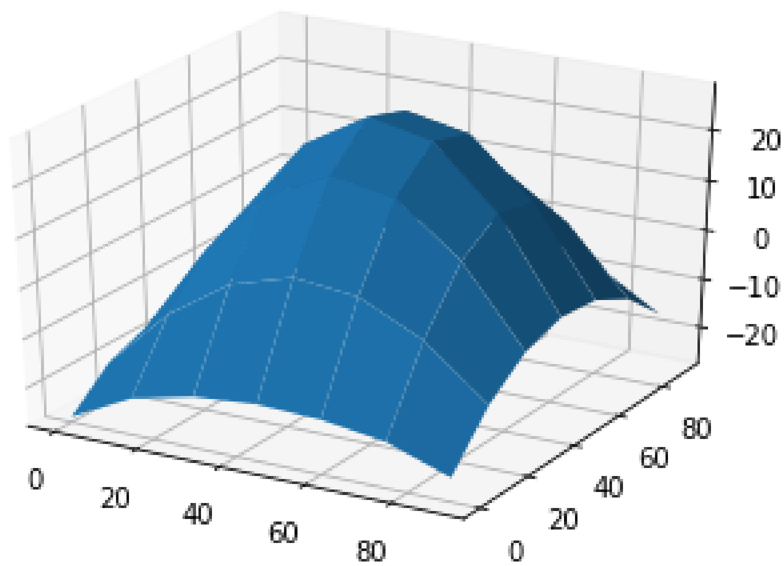
```
In [13]: plot_all(albedo, normals, depth, horn)
```


0
20
40
60
80
0
20
40
60
80


0
25
50
75
100
0
50
100


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


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



1.2 Part 2: [4 pts]

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.

```
In [7]: def get_rot_mat(rot_v, unit=None):
            '''
            Takes a vector and returns the rotation matrix required to align the
            unit vector(2nd arg) to it.
            '''
            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
```

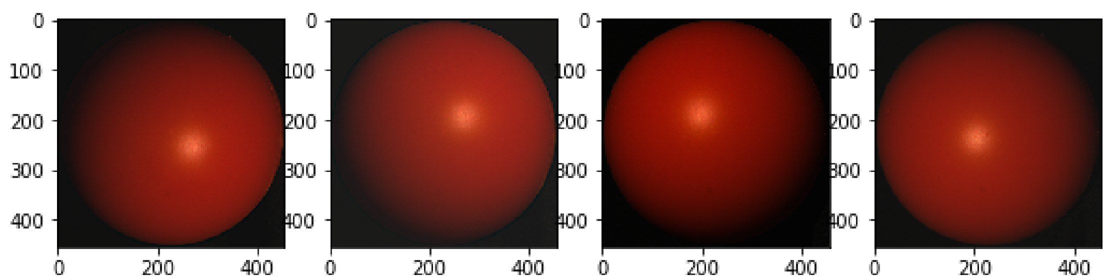
```
    u, v, w = uvw

    # 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 orientation of S channel wrt to RGB
    '''
    R = get_rot_mat(rot_vec)
    S = np.zeros(I_rgb.shape[:2])
    G = np.zeros(I_rgb.shape[:2])
    for x in range(I_rgb.shape[0]):
        for y in range(I_rgb.shape[1]):
            SUV = R.dot(I_rgb[x, y, :])
            S[x, y] = SUV[0]
            G[x, y] = (SUV[1] ** 2 + SUV[2] ** 2) ** 0.5
    return (S,G)
```

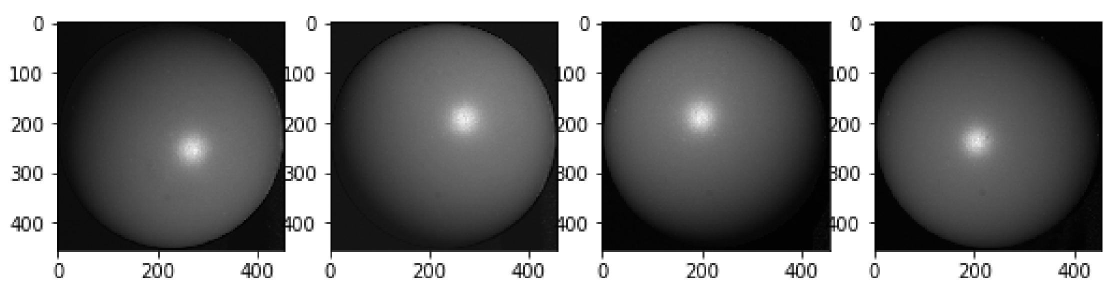
In [8]:

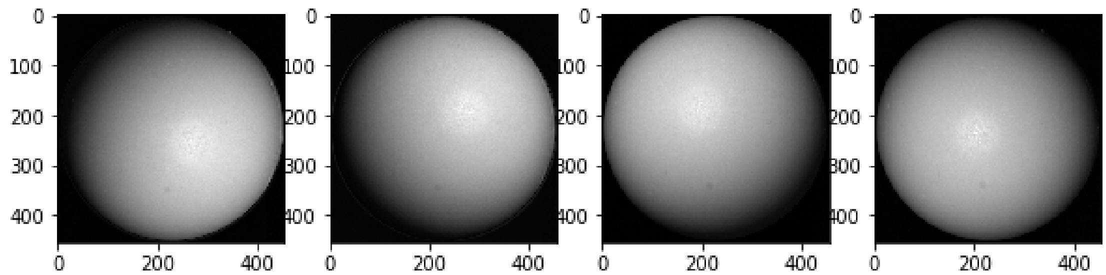
```
def normalize(image):
    res = image.copy()
    res -= res.min()
    res /= res.max()
    return res

def get_three_list(data):
    rot_vec = np.hstack((data['c'][0][0],data['c'][1][0],data['c'][2][0]))
    RGB_list = list()
    S_list = list()
    G_list = list()
    for name in ["im1", "im2", "im3", "im4"]:
        #RGB_list.append(data[name])
        RGB_list.append(normalize(data[name]))
        S, G = RGBToSUV(RGB_list[-1], rot_vec)
        S_list.append(normalize(S))
        G_list.append(G)
    return RGB_list, S_list, G_list
```

```
def display_list(S_list):
    figure = plt.figure(figsize=(10,10))
    for i in range(len(S_list)):
        ax = figure.add_subplot(1, len(S_list), i + 1)
        ax.imshow(S_list[i], cmap="gray")
    plt.show()
```

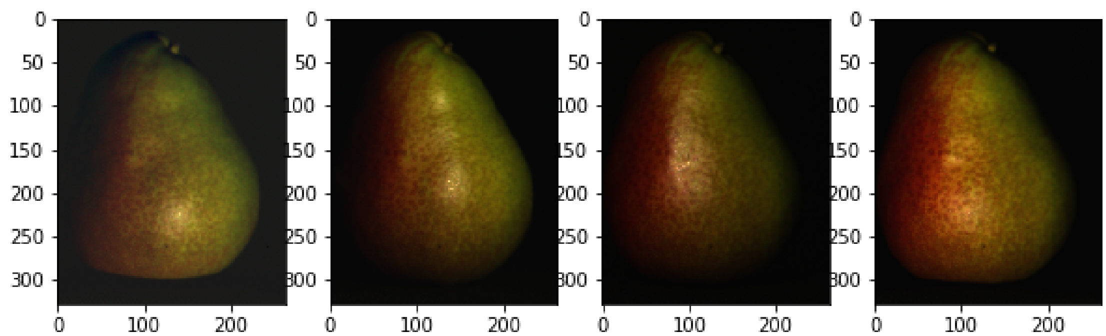
In [9]:
```
pickle_in_sphere = open("specular_sphere.pickle", "rb")
data_sphere = pickle.load(pickle_in_sphere)
# data = pickle.load(pickle_in, encoding="latin1")

# sample input
RGB_sphere_list, S_sphere_list, G_sphere_list = get_three_list(data_sphere)

display_list(RGB_sphere_list)
display_list(S_sphere_list)
display_list(G_sphere_list)
```

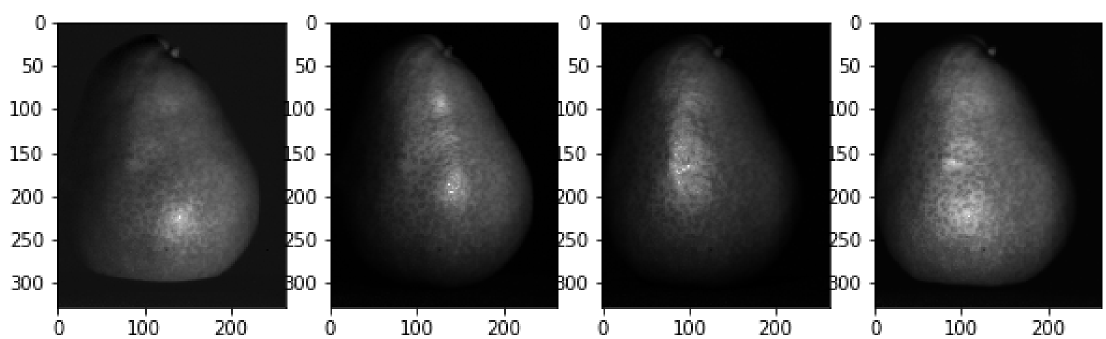


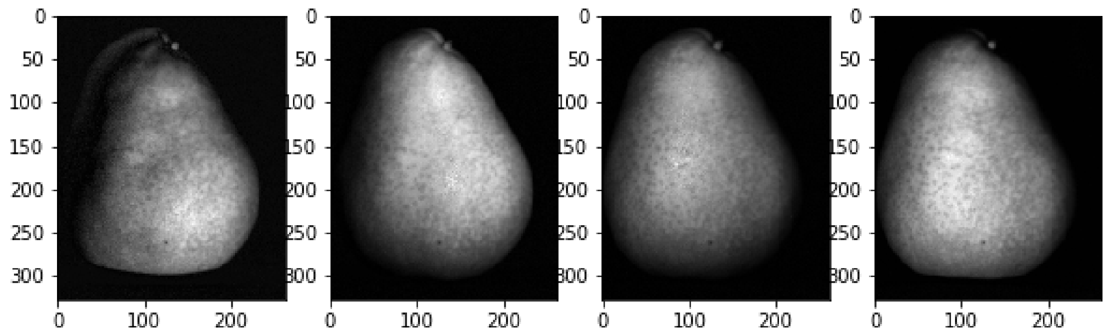




In [10]:
```python
pickle_in_pear = open("specular_pear.pickle", "rb")
data_pear = pickle.load(pickle_in_pear)
# data = pickle.load(pickle_in, encoding="latin1")

# sample input
RGB_pear_list, S_pear_list, G_pear_list = get_three_list(data_pear)

display_list(RGB_pear_list)
display_list(S_pear_list)
display_list(G_pear_list)
```







1.3 Part 3: [4 pts]

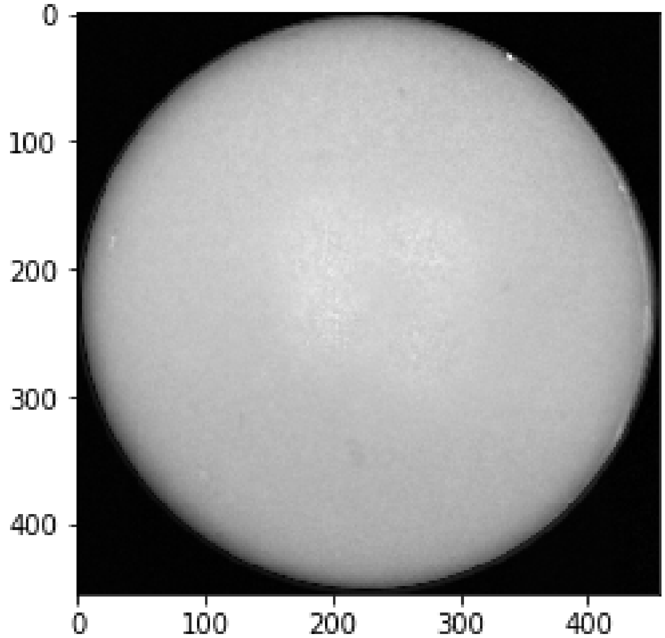
For each specular sphere and pear image sets, using all the four images, include:

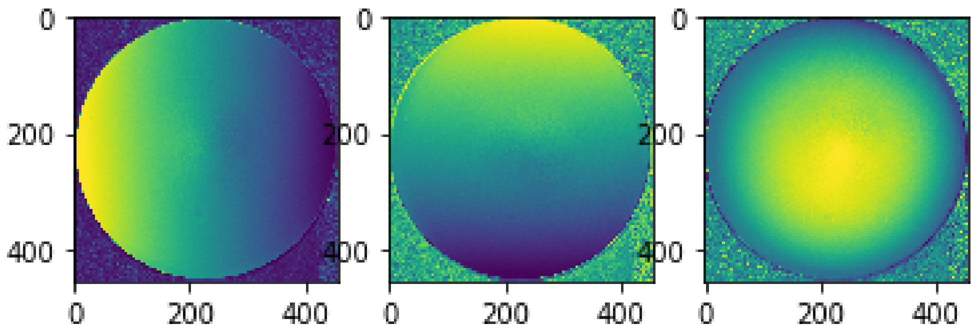
1. The estimated albedo map (original and diffuse)
2. The estimated surface normals (original and diffuse) by showing both
    1. Needle map, and
    2. Three images showing components of surface normal
3. A wireframe of depth map (original and diffuse)

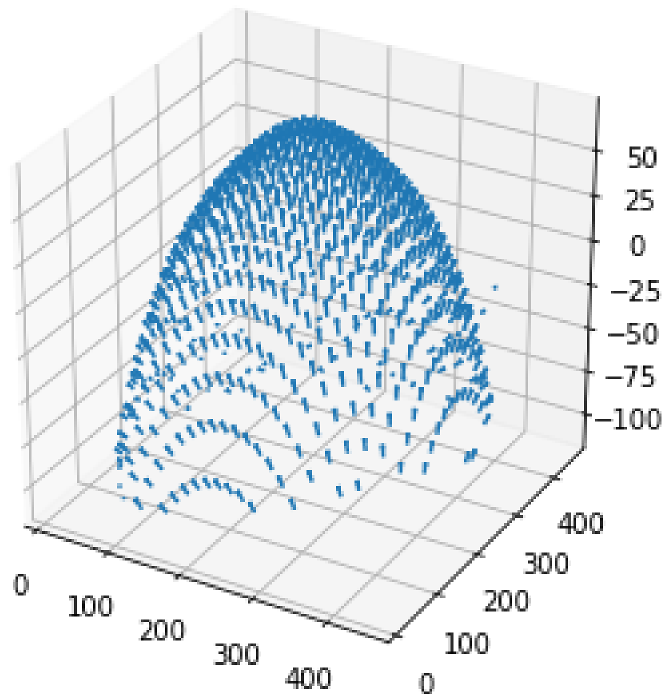
In [11]:
```
def creat_mask(G_list, threshold):
    mask = np.zeros(G_list[0].shape)
    for G in G_list:
        mask = np.logical_or(mask, G > threshold)
    return mask

def rgb2gray(rgb):
    return np.dot(rgb[..., :3], [0.299, 0.587, 0.114])
```

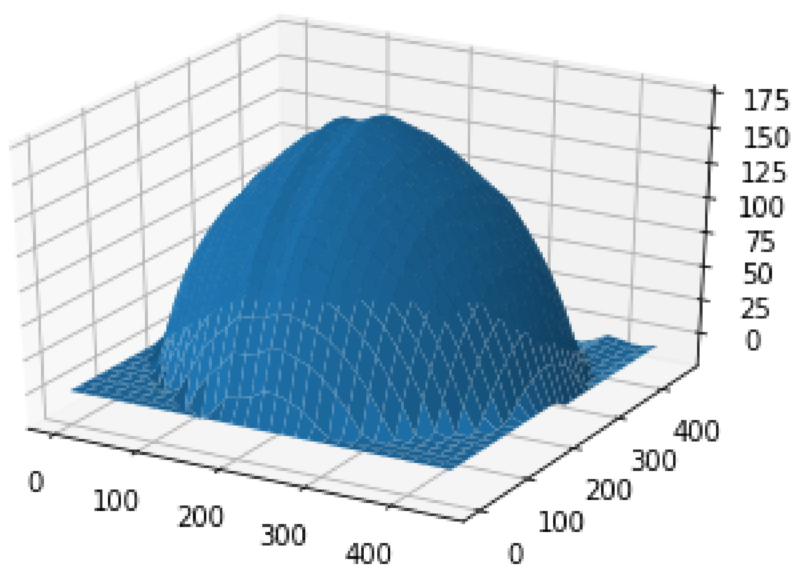
In [41]:
```
# --------------------------------------------------------------------------
# You may reuse the code for photometric_stereo here.
# Write your code below to process the data and send it to photometric_stereo
# and display the albedo, normals and depth maps.
# --------------------------------------------------------------------------

lights_sphere = np.vstack((data_sphere['l1'],data_sphere['l2'],data_sphere['l3'],data_

mask_sphere = creat_mask(G_sphere_list, 0.05)

plt.imshow(mask_sphere,cmap="gray")
plt.show()
```

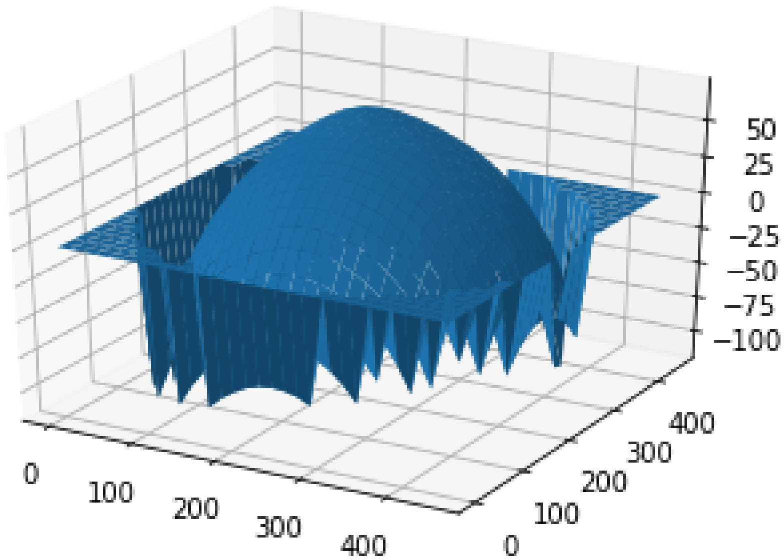
```
albedo_sphere, normals_sphere, depth_sphere, horn_sphere,gx_sphere,gy_sphere = photome

plot_all(albedo_sphere, normals_sphere, depth_sphere, horn_sphere, mask=mask_sphere)
```



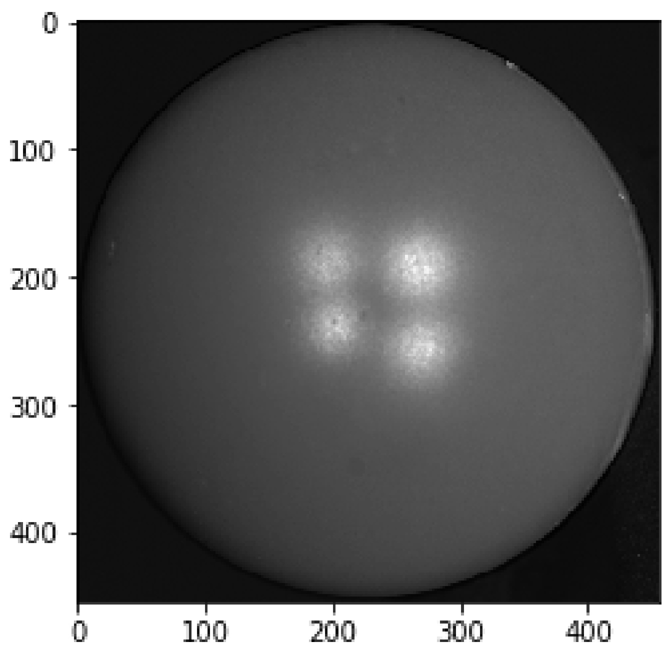


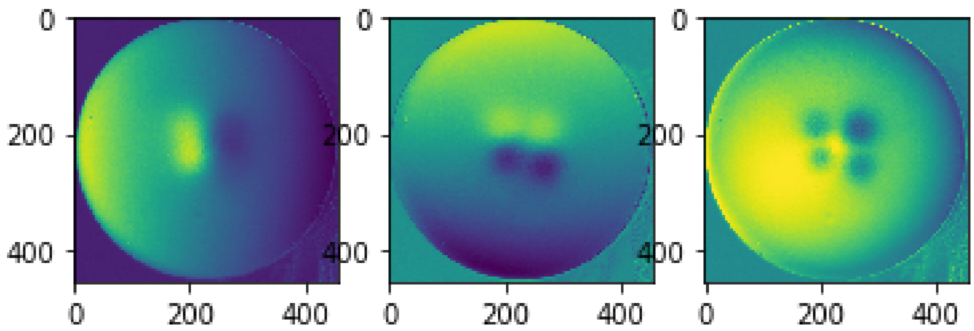

50
25
0
−25
−50
−75
−100
0
100
200
300
400
0
100
200
300
400

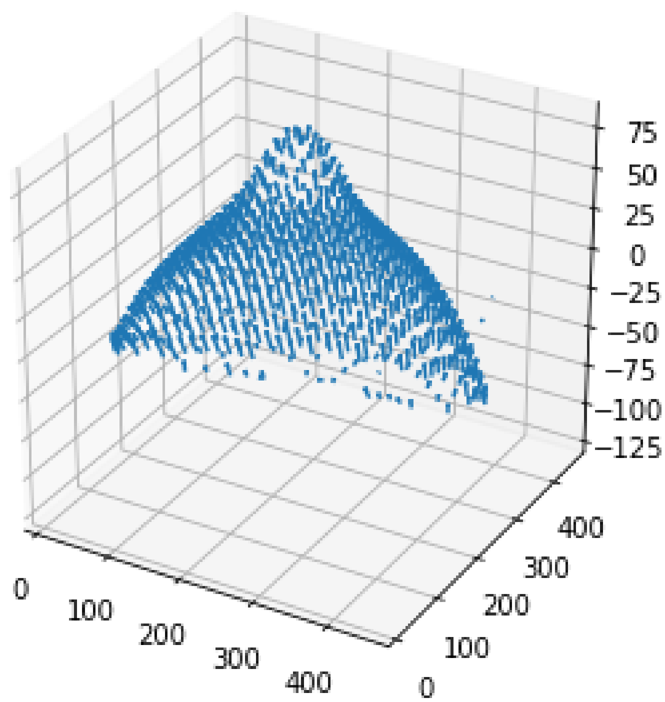

175
150
125
100
75
50
25
0
0
100
200
300
400
0
100
200
300
400

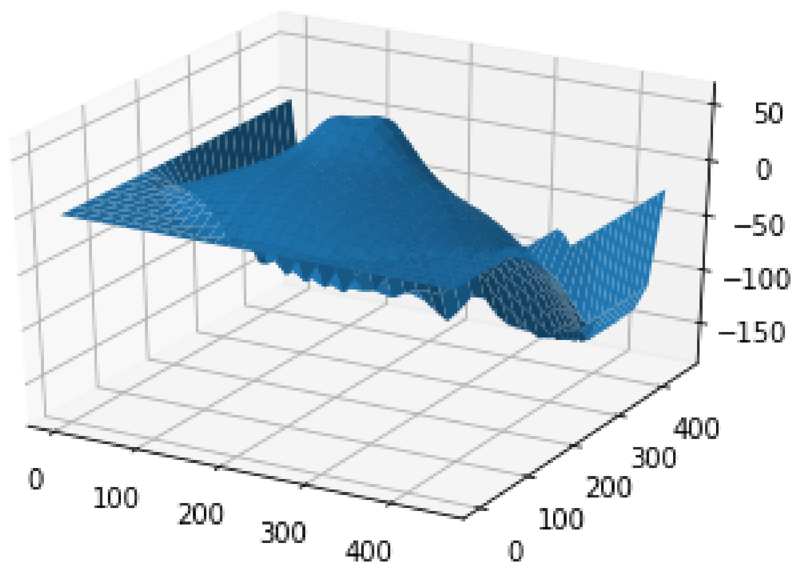


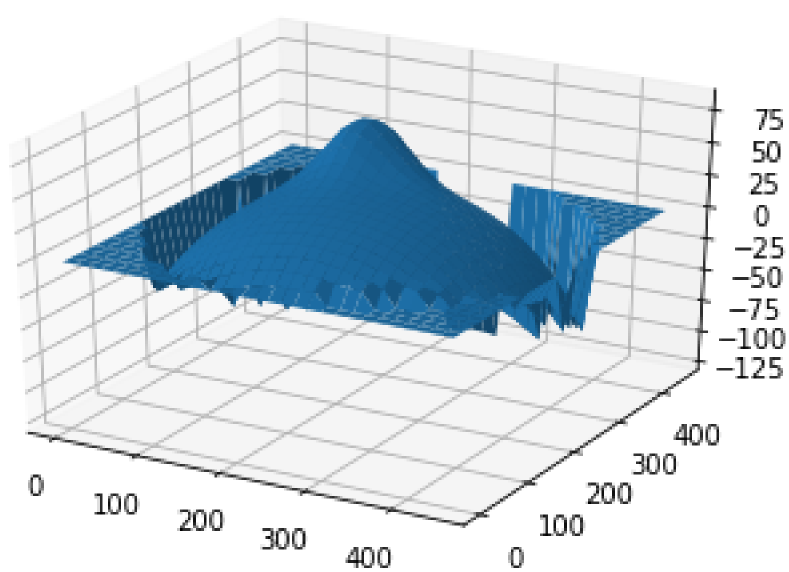

```
In [43]: gray_sphere_list = map(rgb2gray, RGB_sphere_list)

         albedo_sphere_g, normals_sphere_g, depth_sphere_g, horn_sphere_g,gx_sphere_g,gy_sphere

         plot_all(albedo_sphere_g, normals_sphere_g, depth_sphere_g, horn_sphere_g, mask=mask_s
```




0
200
400
0
200
400

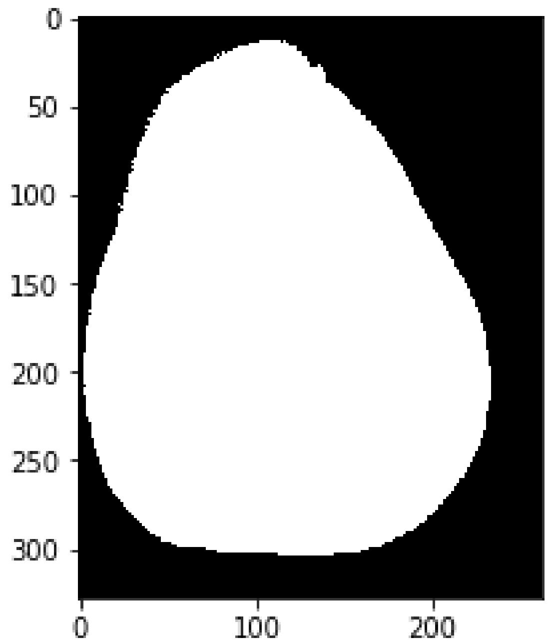

75
50
25
0
−25
−50
−75
−100
−125
0
100
200
300
400

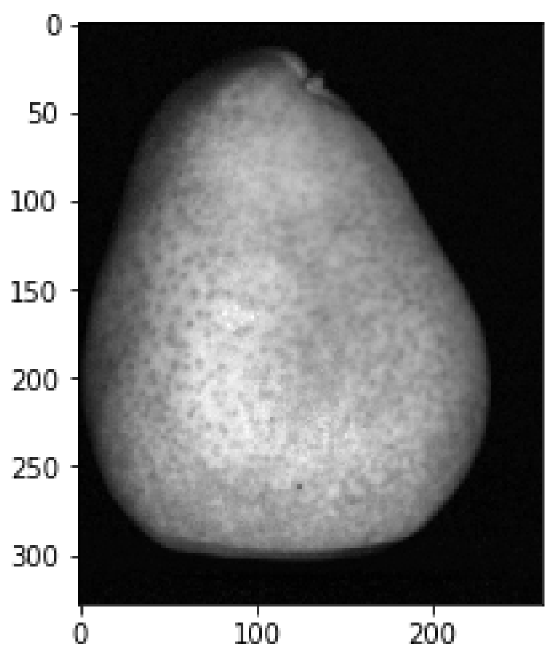


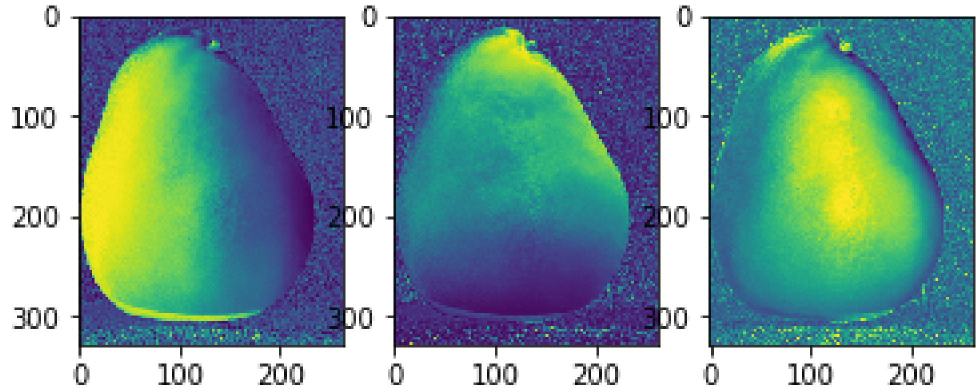


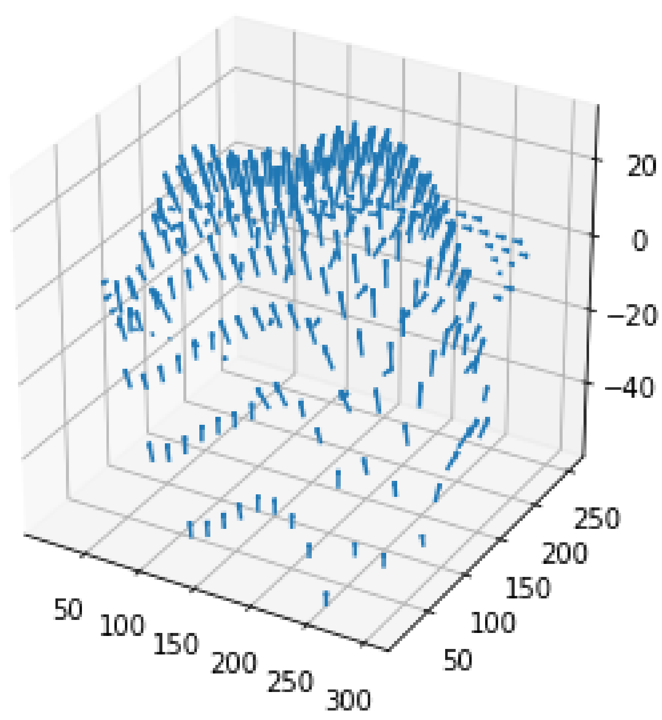
In [49]:
```
lights_pear = np.vstack((data_pear['l1'],data_pear['l2'],data_pear['l3'],data_pear['l4

mask_pear = creat_mask(G_pear_list, 0.03)

plt.imshow(mask_pear,cmap="gray")
```

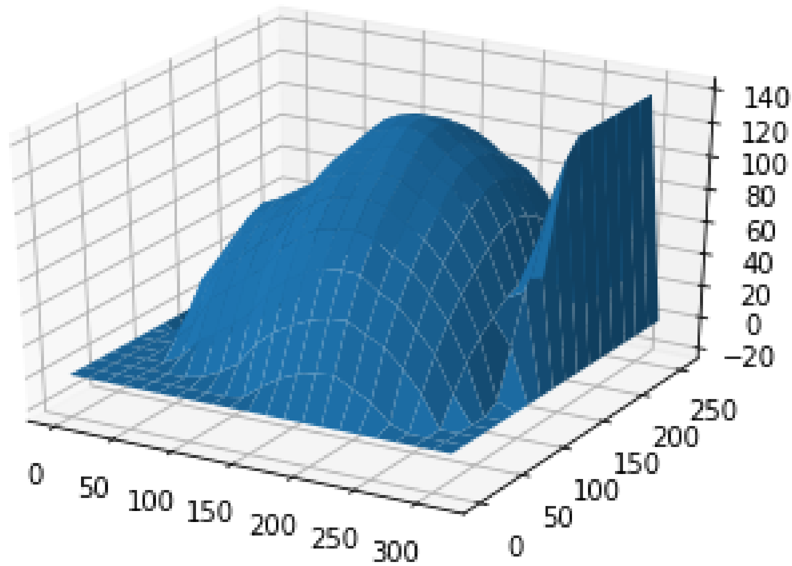
```
plt.show()

albedo_pear, normals_pear, depth_pear, horn_pear,gx_pear,gy_pear = photometric_stereo

plot_all(albedo_pear, normals_pear, depth_pear, horn_pear, mask_pear)
```

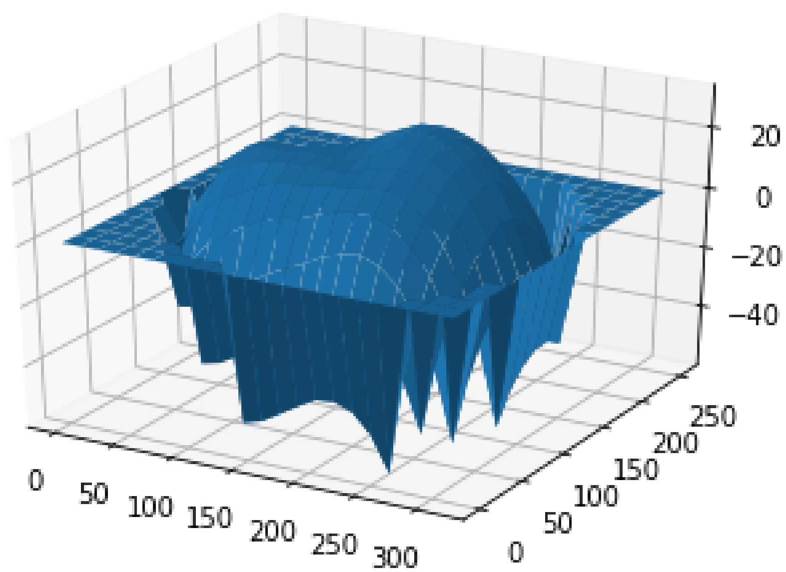




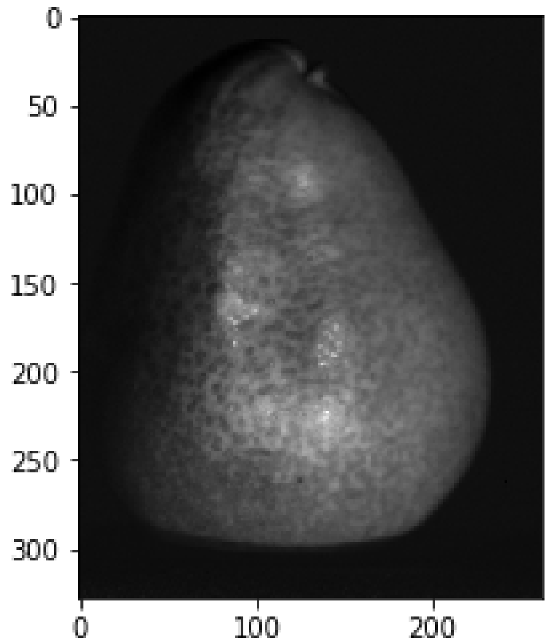

0
100
200
300
0
100
200

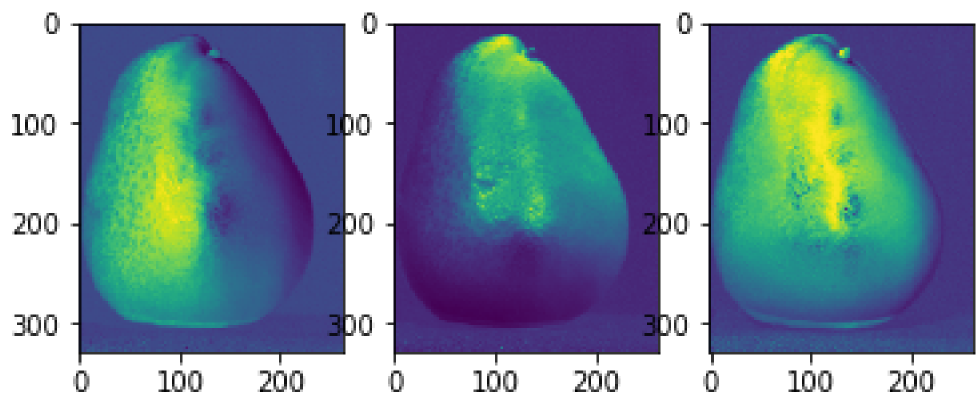

20
0
−20
−40
50
100
150
200
250
300
50
100
150
200
250

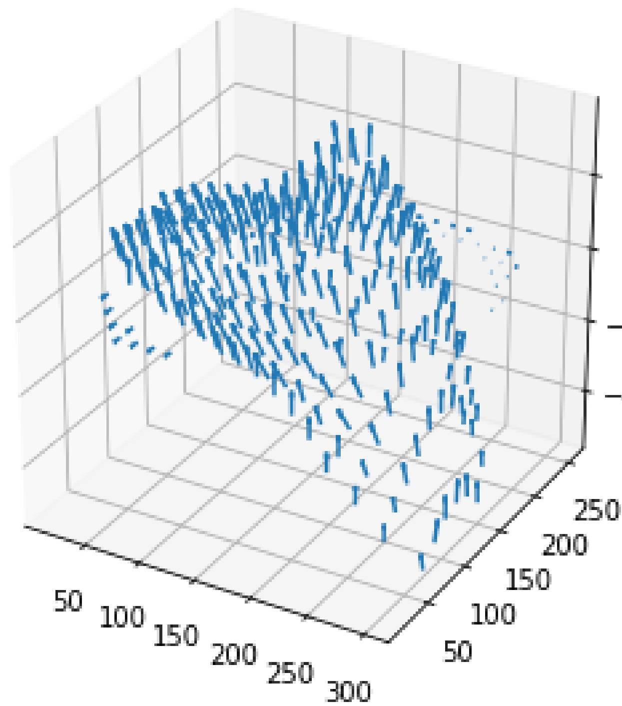


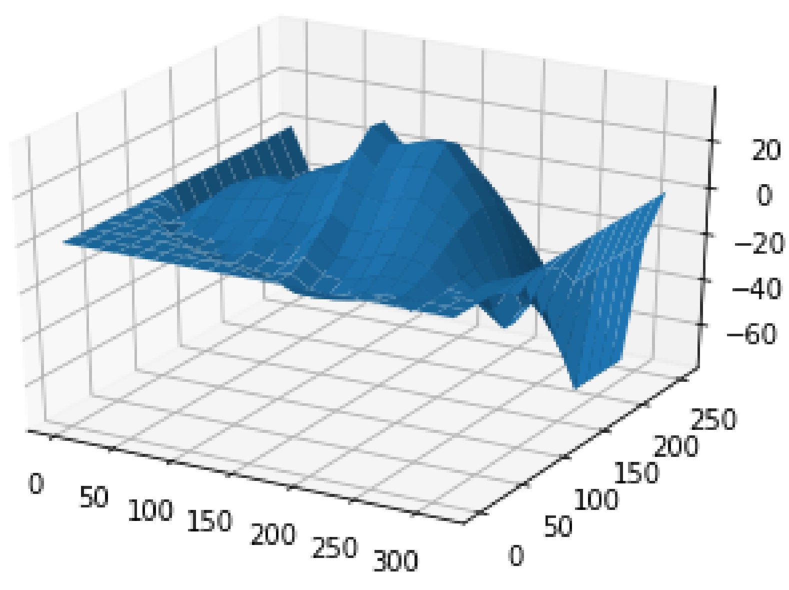


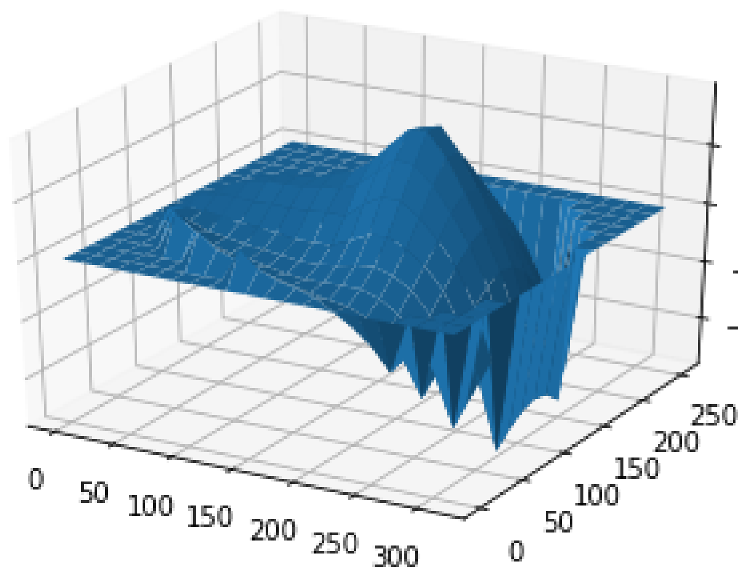
In [50]:
```
gray_pear_list = map(rgb2gray, RGB_pear_list)

albedo_pear_g, normals_pear_g, depth_pear_g, horn_pear_g,gx_pear_g,gy_pear_g = photome

plot_all(albedo_pear_g, normals_pear_g, depth_pear_g, horn_pear_g, mask_pear)
```


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


20
0
−20
−40
50
100
150
200
250
300
50
100
150
200
250


20
0
−20
−40
−60
0
50
100
150
200
250
300
0
50
100
150
200
250



2 Problem 6: Surface Rendering [10 pts]

Please complete the following:

1. Write the function `lambertian()` that calculates the Lambertian light intensity given the light direction **L** with color and intensity **C** and $I_l = 1$, and normal vector **N**. Then use this function in a program that calculates and displays the specular sphere and the pear using each of the two lighting sources found in Table 1. *Note: You do not need to worry about material coefficients in this model.*

2. Write the function `phong()` that calculates the Phong light intensity given the material constants (k_a, k_d, k_s, α), $\mathbf{V} = (0, 0, 1)^\top$, **N** and some number of M light sources. Then use this function in a program that calculates and displays the specular sphere and the pear using each of the sets of coefficients found in Table 2 with each light source individually, and both light sources combined.

Table 1: Light Sources

m	Location	Color (RGB)
1	$(-\frac{1}{3}, \frac{1}{3}, \frac{1}{3})^\top$	$(1, 1, 1)$
2	$(1, 0, 0)^\top$	$(1, .5, .5)$

Table 2: Material Coefficients

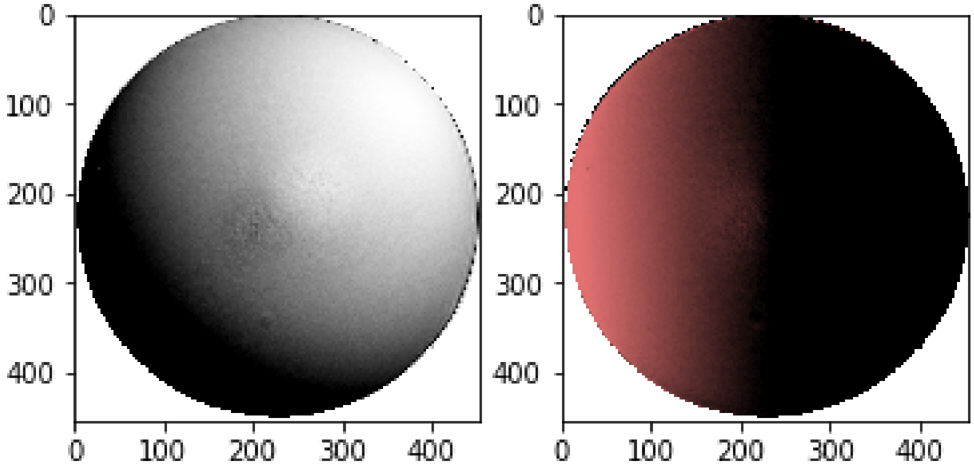
Mat.	k_a	k_d	k_s	α
1	0	0.1	0.75	5
2	0	0.5	0.1	5
3	0	0.5	0.5	10

2.1 Part 1. Lambertian model [4 pts]

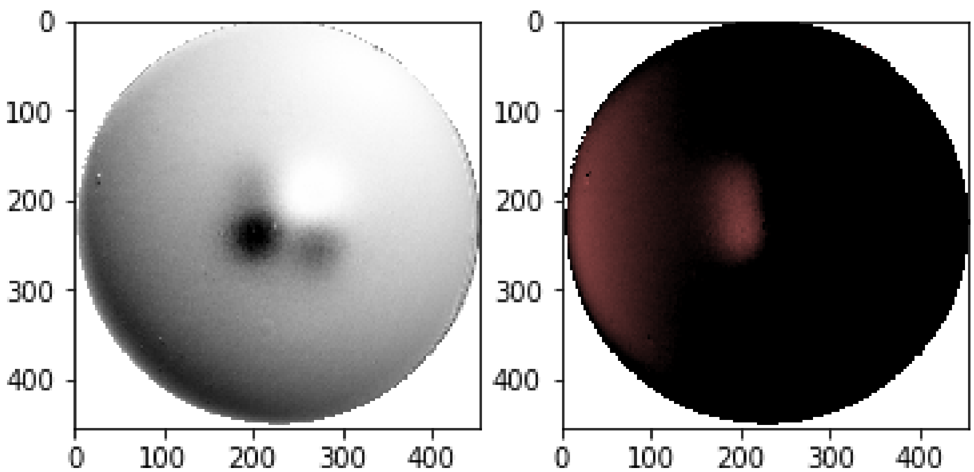
```
In [56]: def lambertian(normals, lights, color, intensity, mask):
             '''Your implementation'''
             i = intensity * normals.dot(lights).dot(color)
             i[i < 0] = 0
             i[mask==0] = 1
             return i
```

```
In [57]: # Output the rendering results
         for normals, mask in zip(
             [normals_sphere, normals_sphere_g, normals_pear, normals_pear_g],
             [mask_sphere, mask_sphere, mask_pear, mask_pear]):
             figure = plt.figure()
             idx = 1
             for lights, color in zip(
                 [np.array([[-1.732/3, 1.732/3, 1.732/3]]).T,np.array([[1, 0, 0]]).T],
                 [np.array([[1, 1, 1]]), np.array([[1, 0.5, 0.5]])]):
                 intensity = 1
                 i = lambertian(normals, lights, color, intensity, mask)
                 print(i.max()),
                 print(i.min())
                 ax = figure.add_subplot(1, 2, idx)
                 ax.imshow(i)
                 idx += 1
             plt.show()
```

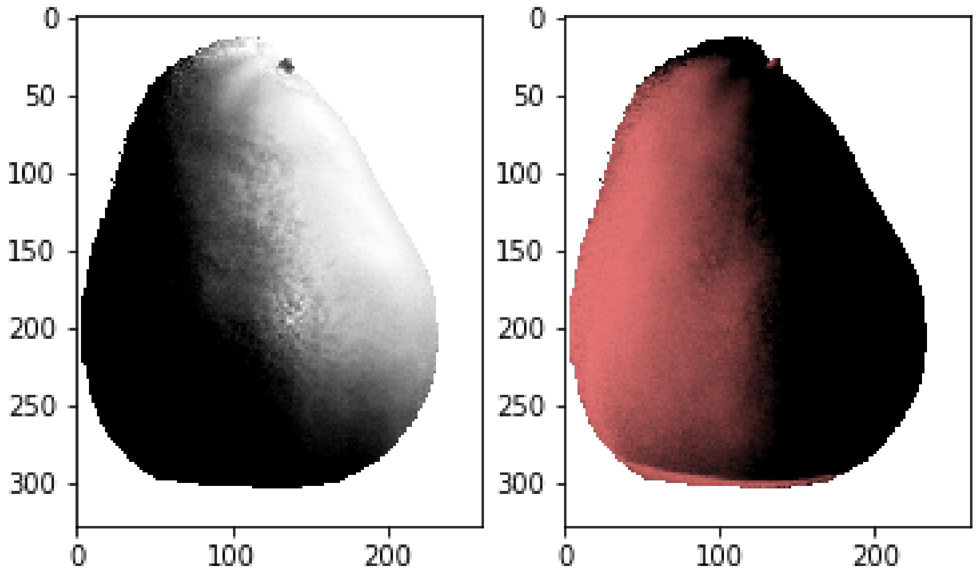
```
1.0 0.0
1.0 0.0
```



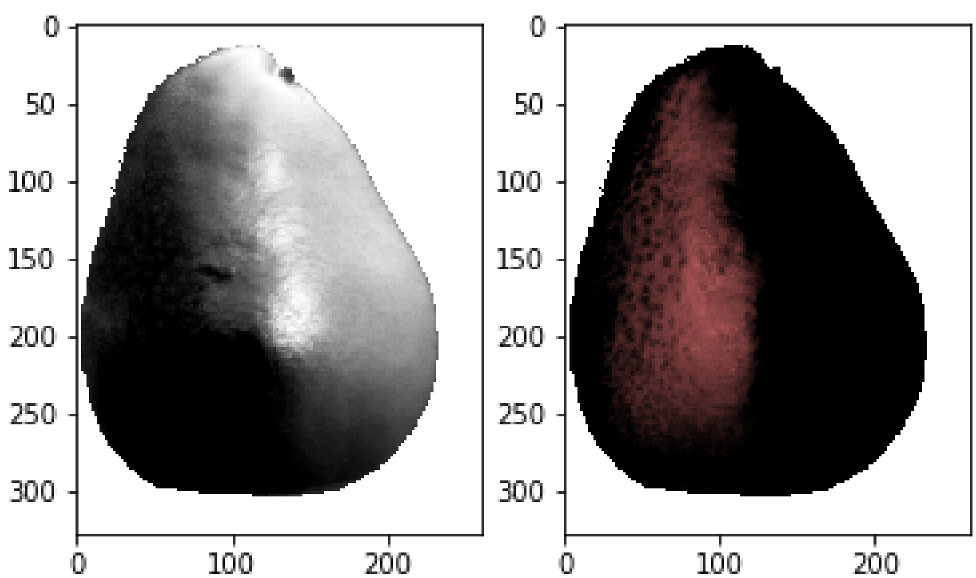
```
1.0 0.0
1.0 0.0
```



```
1.0 0.0
1.0 0.0
```




```
1.0 0.0
1.0 0.0
```



2.2 Part 2. Phong model [6 pts]

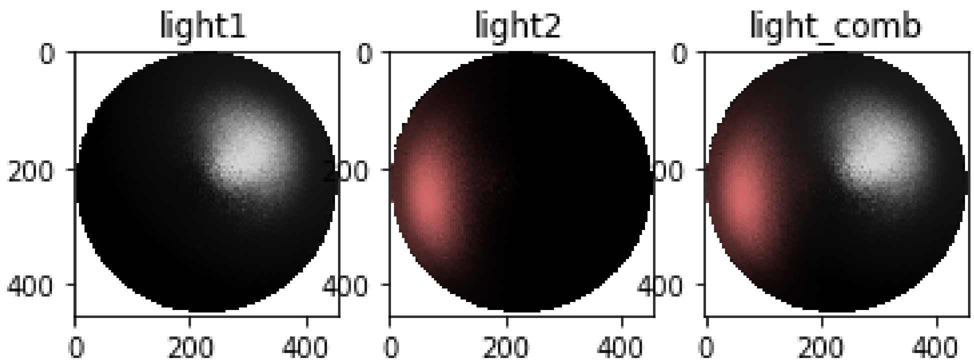
```
In [58]: def phong(normals, lights, color, material, view, mask):
             '''Your implementation'''
```

```
    ka, kd, ks, alpha = material
    M = lights.shape[1]
    i = np.zeros((normals.shape[0], normals.shape[1], 3))
    for m in range(M):
        diffuse = kd * normals.dot(lights[:, m:m+1]).dot(color[m:m+1,:])
        diffuse[diffuse < 0] = 0
        R = (2 * normals * normals.dot(lights[:, m:m+1])) - lights[:,m]
        specular = ks * (R.dot(view) ** alpha).dot(color[m:m+1,:])
        specular[specular < 0] = 0
        i += diffuse + specular
    i[i < 0] = 0
    i[mask==0] = 1
    return i
```

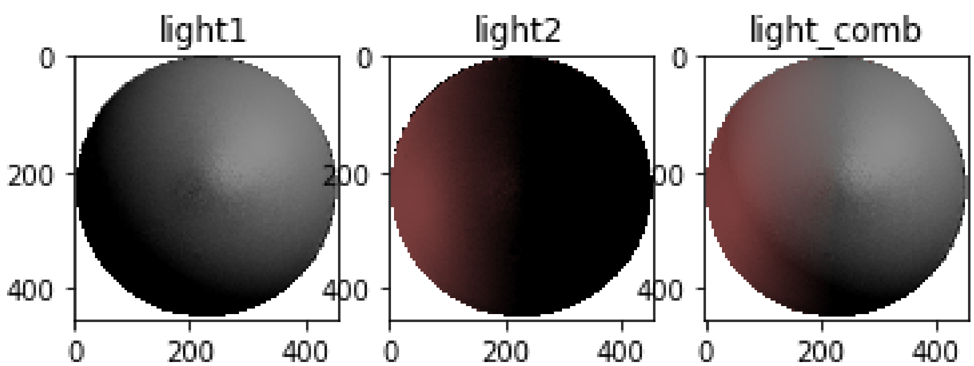
```
In [60]: # Output the rendering results
         # Output the rendering results
         def plot_three(normals, lights, color, material, view, mask):
             figure = plt.figure()

             i1 = phong(normals, lights[:, 0:1], color[0:1, :], material, view, mask)
             print(i1.max()),
             print(i1.min())
             ax1 = figure.add_subplot(131)
             ax1.set_title("light1")
             ax1.imshow(i1)

             i2 = phong(normals, lights[:, 1:2], color[1:2, :], material, view, mask)
             print(i2.max()),
             print(i2.min())
             ax2 = figure.add_subplot(132)
             ax2.set_title("light2")
             ax2.imshow(i2)

             i3 = phong(normals, lights, color, material, view, mask)
             print(i3.max()),
             print(i3.min())
             ax3 = figure.add_subplot(133)
             ax3.set_title("light_comb")
             ax3.imshow(i3)

             plt.show()

         lights_comb=np.array([[-1.732/3, 1.732/3, 1.732/3], [1, 0, 0]]).T
         color_comb=np.array([[1,1,1],[1, 0.5, 0.5]])
         view=np.array([[0,0,1]]).T

         for name, normals, mask in zip(
             ["sphere G", "sphere RGB", "pear G", "pear RGB"],
```

```
[normals_sphere, normals_sphere_g, normals_pear, normals_pear_g],
[mask_sphere, mask_sphere, mask_pear, mask_pear]):
idx = 1
for material in [(0,0.1,0.75,5), (0,0.5,0.1,5), (0,0.5,0.5,10)]:
    print(name + ", Mat.:" + str(material))
    plot_three(normals, lights_comb, color_comb, material, view, mask)
```

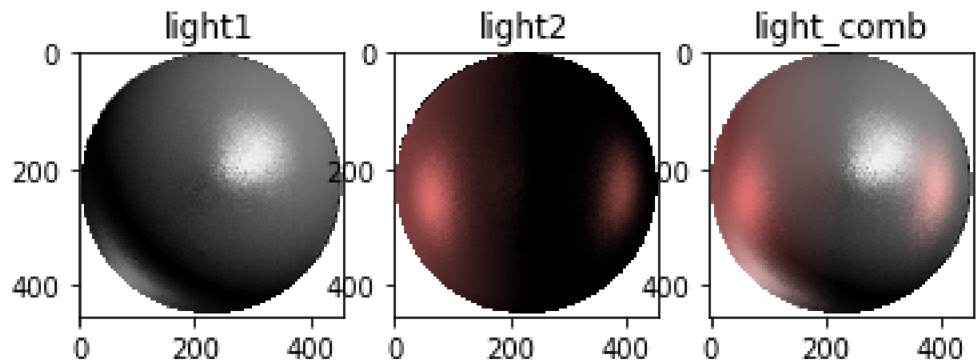
```
sphere G, Mat.:(0, 0.1, 0.75, 5)
1.0 0.0
1.0 0.0
1.0 0.0
```



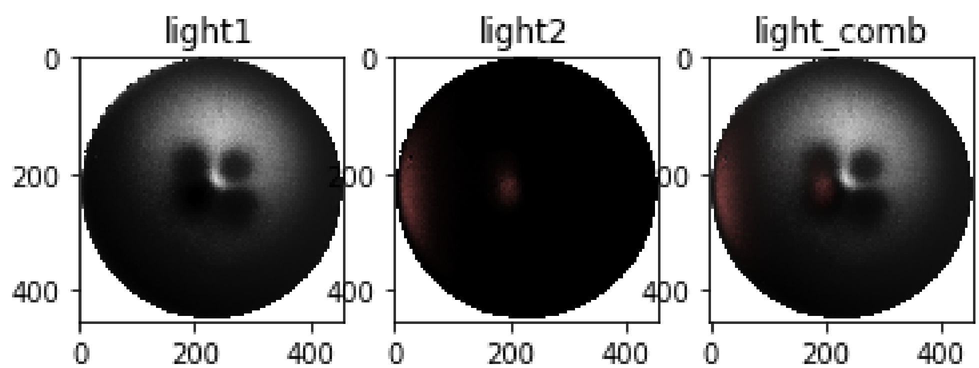

```
sphere G, Mat.:(0, 0.5, 0.1, 5)
1.0 0.0
1.0 0.0
1.0 0.0
```



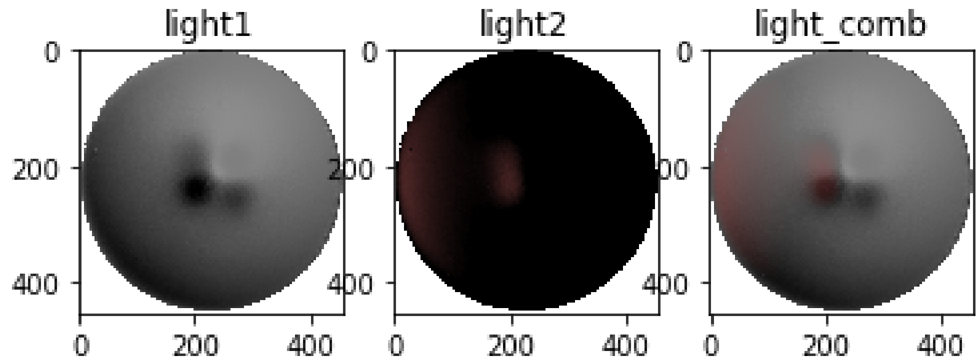
```
sphere G, Mat.:(0, 0.5, 0.5, 10)
1.0 0.0020579164975333777
1.0 2.3813987573071224e-50
1.0 0.0020594043611896736
```



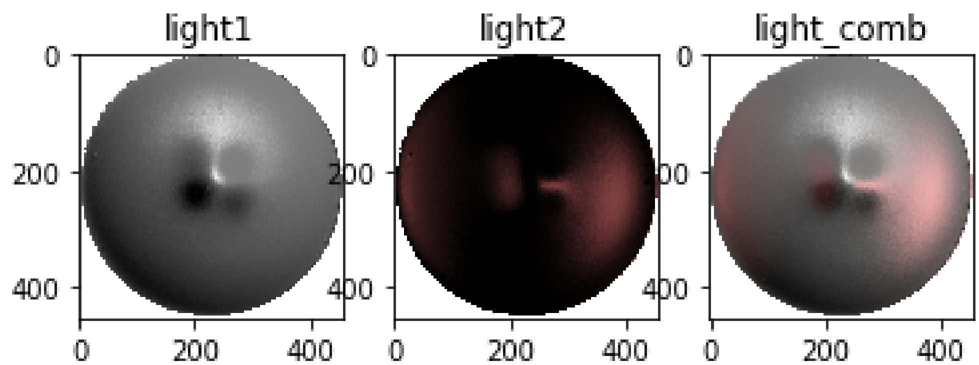

```
sphere RGB, Mat.:(0, 0.1, 0.75, 5)
1.0 0.0
1.0 0.0
1.0 0.01068454712519547
```



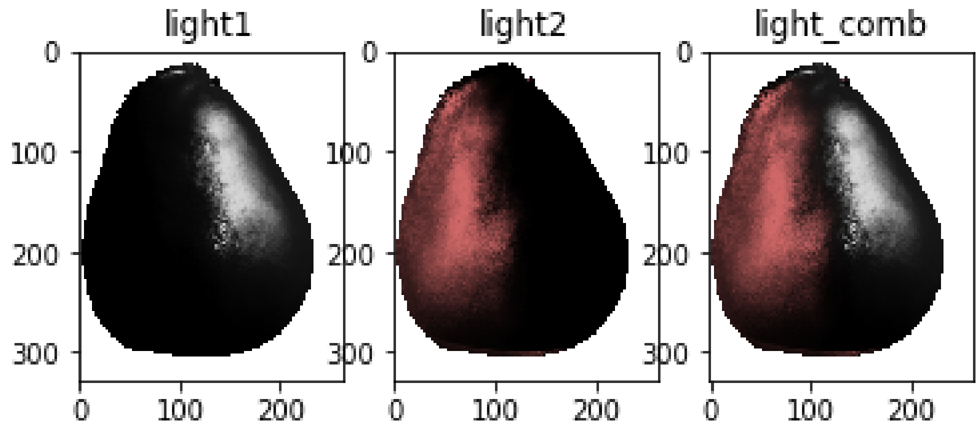

```
sphere RGB, Mat.:(0, 0.5, 0.1, 5)
1.0 0.0
1.0 0.0
1.0 0.05337165477892052
```



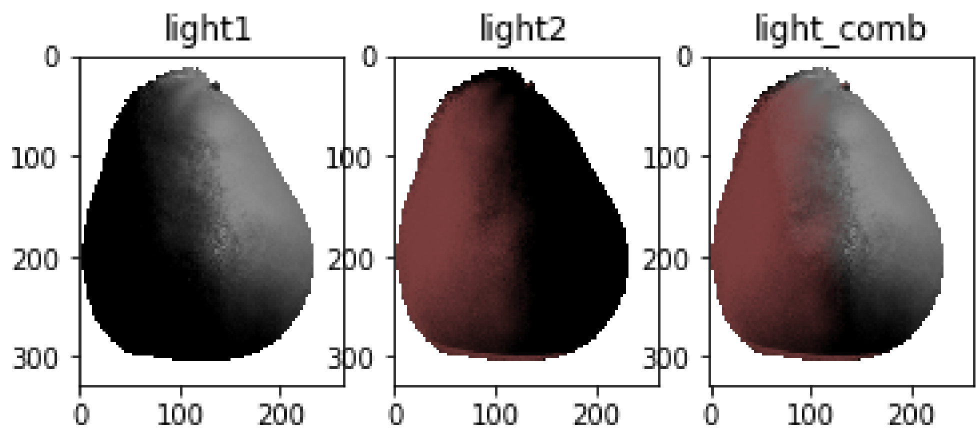

```
sphere RGB, Mat.:(0, 0.5, 0.5, 10)
1.0 0.0020748385903057724
1.0 1.8600061649690463e-56
1.0 0.053626992830774406
```



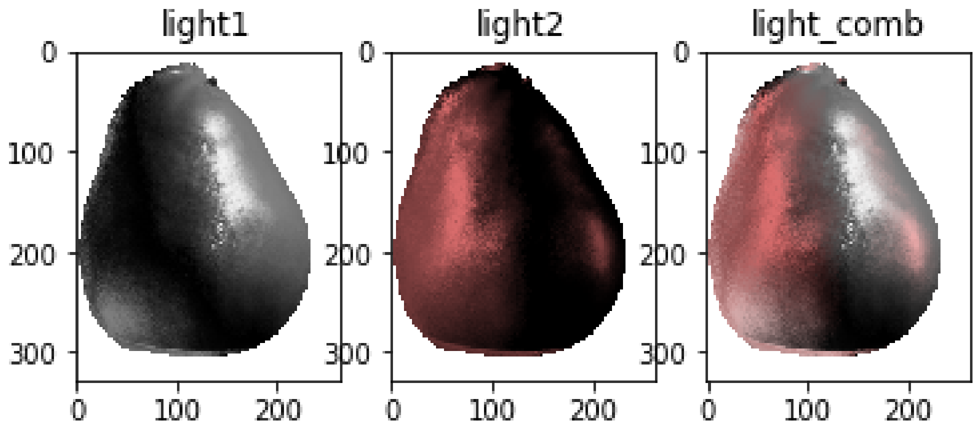

```
pear G, Mat.:(0, 0.1, 0.75, 5)
1.0 0.0
1.0 0.0
1.0 0.0
```



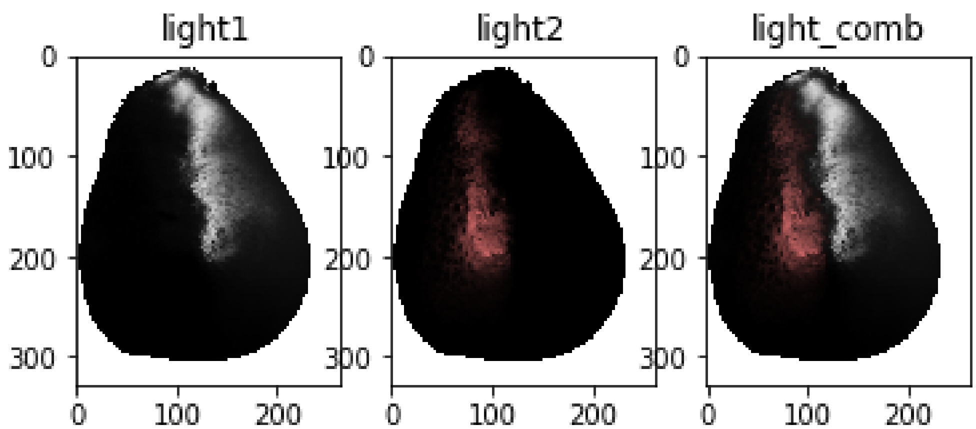

```
pear G, Mat.:(0, 0.5, 0.1, 5)
1.0 0.0
1.0 0.0
1.0 0.0
```



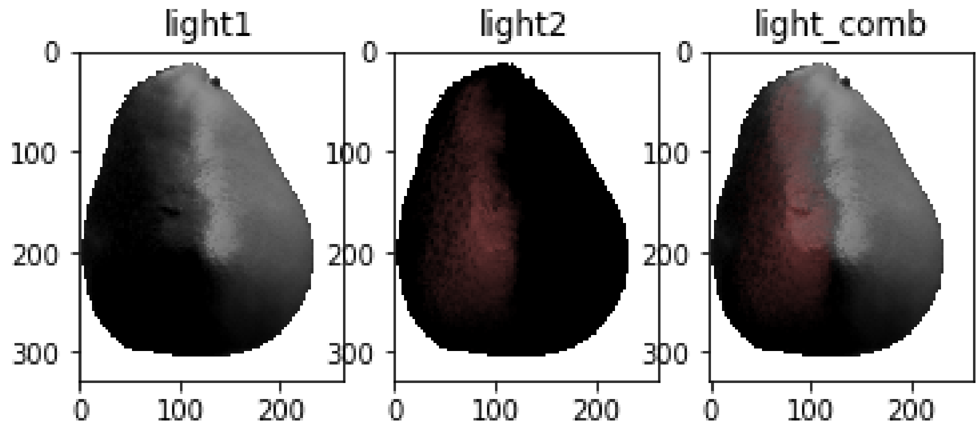

```
pear G, Mat.:(0, 0.5, 0.5, 10)
1.0 0.002060541113712729
1.0 1.0853629893922963e-42
1.0 0.002060947223383022
```




```
pear RGB, Mat.:(0, 0.1, 0.75, 5)
1.0 0.0
1.0 0.0
1.0 0.0
```




```
pear RGB, Mat.:(0, 0.5, 0.1, 5)
1.0 0.0
1.0 0.0
1.0 0.0
```




```
pear RGB, Mat.:(0, 0.5, 0.5, 10)
1.0 0.0020570273467083585
1.0 1.5581369565310348e-51
1.0 0.002057028393946988
```