

# HW2

November 8, 2018

## 1 CSE 252A Computer Vision I Fall 2018 - Assignment 2

1.0.1 Instructor: David Kriegman

1.0.2 Assignment Published On: Wednesday, October 24, 2018

1.0.3 Due On: Wednesday, November 7, 2018 11:59 pm

### 1.1 Instructions

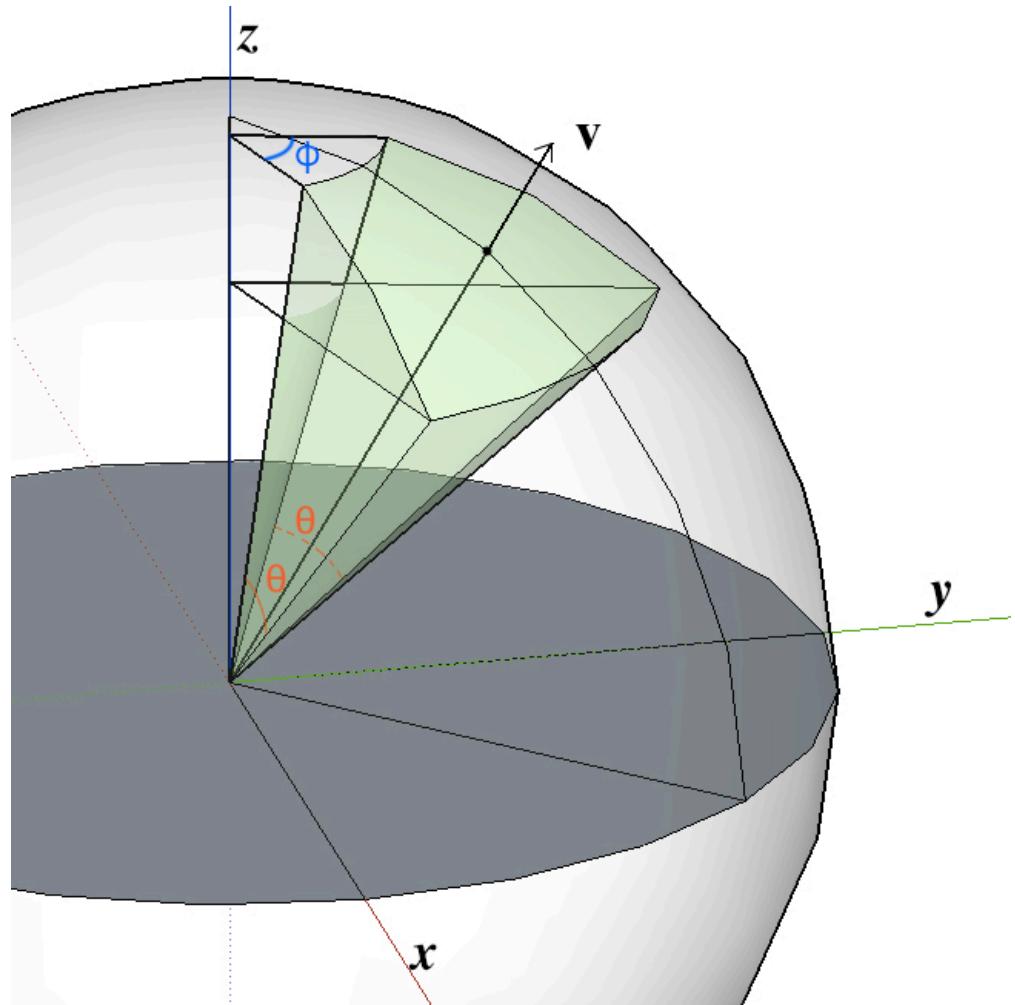
- Review the academic integrity and collaboration policies on the course website.
- This assignment must be completed individually.
- This assignment contains theoretical and programming exercises. If you plan to submit hand written answers for theoretical exercises, please be sure your writing is readable and merge those in order with the final pdf you create out of this notebook. You could fill the answers within the notebook itself by creating a markdown cell.
- Programming aspects of this assignment must be completed using Python in this notebook.
- If you want to modify the skeleton code, you can do so. This has been provided just to provide you with a framework for the solution.
- You may use python packages for basic linear algebra (you can use numpy or scipy for basic operations), but you may not use packages that directly solve the problem.
- If you are unsure about using a specific package or function, then ask the instructor and teaching assistants for clarification.
- You must submit this notebook exported as a pdf. You must also submit this notebook as .ipynb file.
- You must submit both files (.pdf and .ipynb) on Gradescope. You must mark each problem on Gradescope in the pdf.
- **Late policy** - 10% per day late penalty after due date up to 3 days.

### 1.2 Problem 1: Steradians [2 pts]

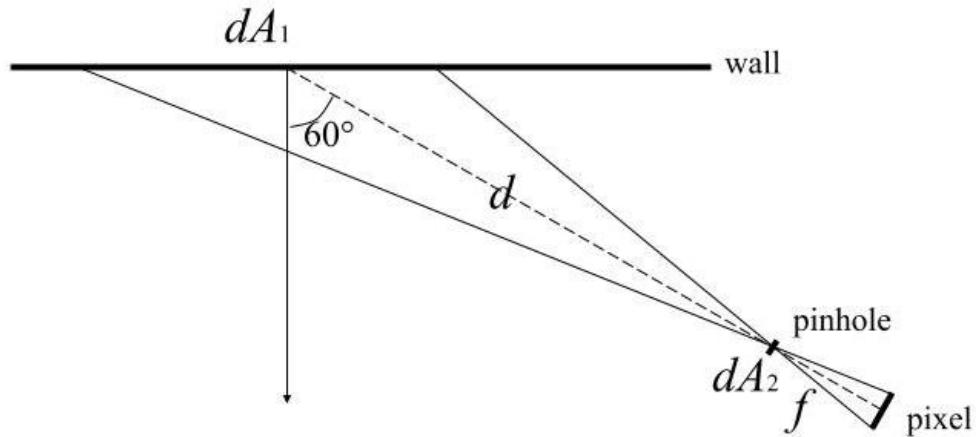
Calculate the number of steradians contained in a spherical wedge with radius  $r = 1$ , defined by  $\theta = \frac{\pi}{6}$ ,  $\phi = \frac{\pi}{6}$  centered around vector  $(\frac{\sqrt{2}}{4}, \frac{\sqrt{2}}{4}, \frac{\sqrt{3}}{2})$ .

### 1.3 Problem 2: Irradiance [6 pts]

Consider a camera looking at Lambertian wall with constant albedo, illuminated by a light source at infinity such that the radiance emitted by the wall is  $L$  in all directions. The angle between the



Problem1 spherical wedge



### Problem2 irradiance

optical axis and the wall's surface normal is 60 degrees. The focal length of the camera is 50mm, and the pixels are 1mm by 1mm.

We will solve this for a pixel centered at the optical axis and also consider three distances along the line of sight between the pinhole and wall (1000mm, 2000mm, 4000mm). We will also walk through four steps and take advantage of small area approximations.

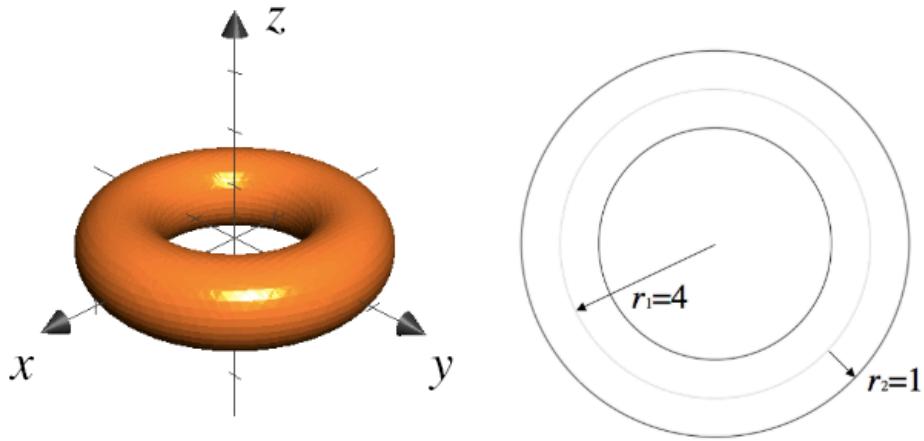
1. Start with  $d = 1000$  mm,
  1. If the wall is parallel to the image plane at distance  $d$ , what is the area of the wall that projects to the central pixel?
  2. Now, if the wall is tilted by 60 degrees, what is the area of the wall that projects to the pixel (you can treat this as a small area and can neglect perspective distortion). Call this  $dA_1$ .
  3. If we treat the pinhole as having a small area  $dA_2$  that is parallel to the image plane, what is the power received at  $dA_2$ ?
  4. If all the power received at  $dA_2$  passes through the pinhole and is received by the central pixel, what is the irradiance at the central pixel?
2. What is the irradiance at the central pixel for  $d = 2000$  mm?
3. What is the irradiance at the central pixel for  $d = 4000$  mm?
4. What can you learn about image irradiance as a function of distance from this example?

### 1.4 Problem 3: Diffused Objects and Brightness [4 pts]

We see a diffuse torus centered at the origin in an orthographic camera, looking down the z-axis. The parameters of the torus are shown in the Figure "Problem3 torus" and the albedo is  $\rho$ .

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

What is the brightness at a point  $(x, y)$  on the surface?



Problem3 torus

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

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

The edge lengths of the source and occluder are 2 and 4, respectively, and they are vertically above one another with their centers aligned. The distances from the occluder to the source and plane are both 3.

1. What is the area of the umbra on the plane?
2. What is the area of the penumbra on the plane?

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

The goal of this problem 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 specularities 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.

#### 1.6.1 Data

Synthetic Images, Specular Sphere Images, Pear Images for Part 1, 2, 3: Available in \*.pickle files (graciously provided by Satya Mallick) which contain

- im1, im2, im3, im4... images.
- l1, l2, l3, l4... light source directions.
- c (when required) color of light source.

### 1.6.2 Part 1: [6 pts]

Implement the photometric stereo technique described in Forsyth and Ponce 2.2.4 (*Photometric Stereo: Shape from Multiple Shaded Images*) and the lecture notes.

Your program should have two parts:

1. Read in the images and corresponding light source directions, and estimate the surface normals and albedo map.
2. Reconstruct the depth map from the surface 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 showing both
  1. Needle map, and
  2. Three images showing components of surface normal.
3. A wireframe of depth map.

An example of outputs is shown in the Figure "Problem5 example".

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

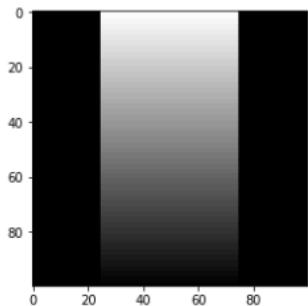
```
In [26]: ## Example: How to read and access data from a pickle
import pickle
import matplotlib.pyplot as plt
%matplotlib inline
pickle_in = open("synthetic_data.pickle", "rb")
# data = pickle.load(pickle_in)
data = pickle.load(pickle_in, encoding="latin1")

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

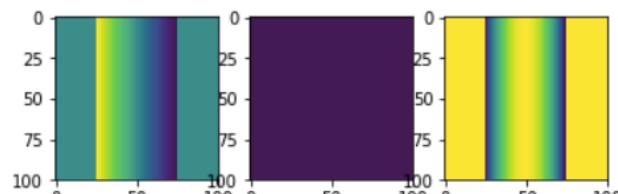
# To access the value of an entity, refer it by its key.
print("Image:")
plt.imshow(data["im1"], cmap = "gray")
plt.show()

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

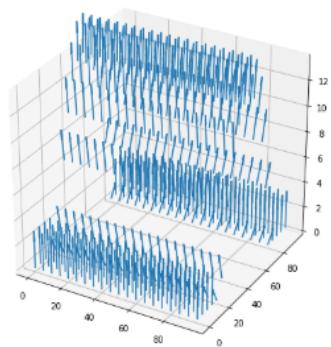
```
Keys: dict_keys(['__version__', 'l4', '__header__', 'im1', 'im3', 'im2', 'l2', 'im4', 'l1', '__g
Image:
```



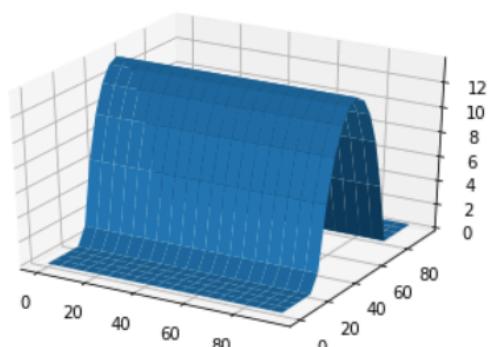
Albedo map



Normals as three separate channels

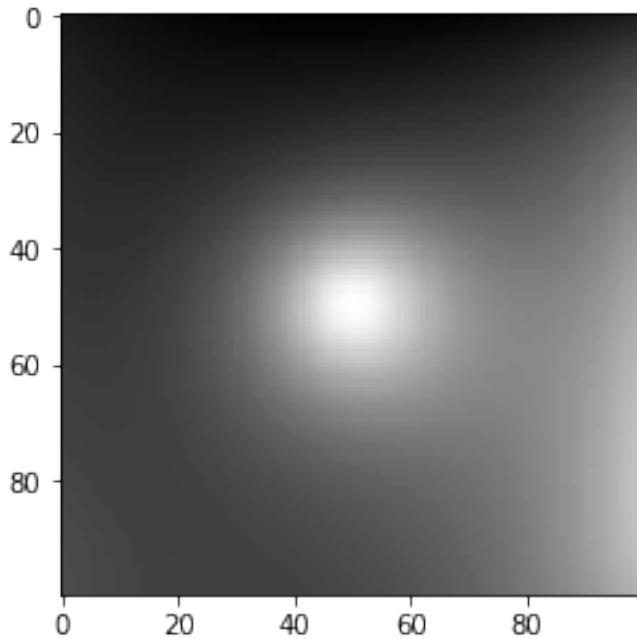


Needle map



Wireframe of depth map

Problem5 example



Light source direction: [[0 0 1]]

```
In [2]: import numpy as np
        from scipy.signal import convolve
        from numpy import linalg
        import math

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 [3]: def photometric_stereo(images, lights, mask, niter=1000):
    # We will use linear least square formula to compute
    # b, which is surface normal scaled by albedo, given by formula
    # b = (S_transpose * S)_inverse * S_transpose * E(pixel-values/intensities)
    h, w = images[0].shape

    albedo_scaled_normal = np.ones((h, w, 3))

    # light sources matrices
    S = lights
    S_tran_S = np.matmul(np.transpose(S), S)
    S_tran_S_inv = np.linalg.inv(S_tran_S)
    _b = np.matmul(S_tran_S_inv, np.transpose(S))

    # Now iterate over each pixel and computer
    # its 'b'(which is albedo scaled by normal)
    for i in range(0, h):
        for j in range(0, w):
            E = [images[k, i, j] for k in range(0, images.shape[0])]
            E = np.asarray(E)
            b = np.matmul(_b, np.transpose(E))
            albedo_scaled_normal[i, j, :] = np.transpose(b)

```

```

# create containers to save albedo and normals
albedo = np.zeros(images[0].shape)
normals = np.dstack((np.zeros(images[0].shape),
                     np.zeros(images[0].shape),
                     np.ones(images[0].shape)))

# Now use calculated 'b' to computer albedo and normal
for i in range(0, h):
    for j in range(0, w):
        x, y, z = albedo_scaled_normal[i, j, :]
        magnitude = float(math.sqrt(x**2 + y**2 + z**2))
        albedo[i, j] = magnitude
        if mask[i, j] != 0:
            normals[i, j, :] = x/magnitude, y/magnitude, z/magnitude

# creat conatiners to save scaline-integral
H = np.zeros(images[0].shape)
height = 0
# calculate height values for leftmost column
for i in range(0, h):
    if mask[i, 0] != 0:
        x, y, z = normals[i, 0, :]
        q = y/z
        height += q
        H[i, 0] = height
# calculate height values for rest rows and cols
for i in range(0, h):
    height = H[i, 0]
    for j in range(1, w):
        if mask[i, j] != 0:
            x, y, z = normals[i, j, :]
            p = x/z
            height += p
            H[i, j] = height

# Horns integration method
gx = np.zeros(images[0].shape)
gy = np.zeros(images[0].shape)
for i in range(0, h):
    for j in range(1, w):
        if mask[i, j] != 0:
            x, y, z = normals[i, j, :]
            p, q = x/z, y/z
            gx[i, j], gy[i, j] = p, q

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

return albedo, normals, H, H_horn

```

```
In [27]: 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 = photometric_stereo(images, lights, mask, niter=100000)

# -----
# 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.
# -----


# Stride in the plot, you may want to adjust it to different images
stride = 5
print("Using Synthetic data: im1, im2, im4")
# showing albedo map
print("")
print("Albedo Map")
fig = plt.figure(figsize=(3, 3))
albedo_max = albedo.max()
albedo = albedo / albedo_max
plt.imshow(albedo, cmap="gray")
plt.show()

print("Normal as three separate channels")
# 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()
```

```

print("Showing Normal as quiver")
# showing normals as quiver
X, Y, _ = np.meshgrid(np.arange(0,np.shape(normals)[0], 5),
                      np.arange(0,np.shape(normals)[1], 5),
                      np.arange(1))
X = X[..., 0]
Y = Y[..., 0]
Z = depth[::-stride,::stride].T
NX = normals[..., 0][::stride,::-stride].T
NY = normals[..., 1][::-stride,::stride].T
NZ = normals[..., 2][::stride,::stride].T
fig = plt.figure(figsize=(7, 7))
ax = fig.gca(projection='3d')
plt.quiver(X,Y,Z,NX,NY,NZ, length=5.0)
plt.show()

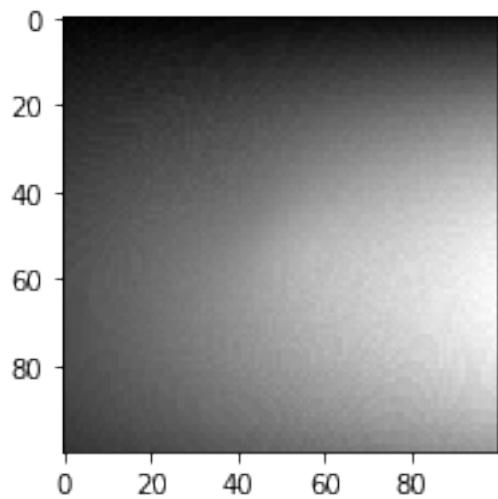
print("Wireframe depth map using scaline approach")
# plotting wireframe depth map
H = depth[::-stride,::stride]
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(X,Y, H.T)
plt.show()

print("Wireframe depth map using Horn")
H = horn[::-stride,::stride]
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(X,Y, H.T)
plt.show()

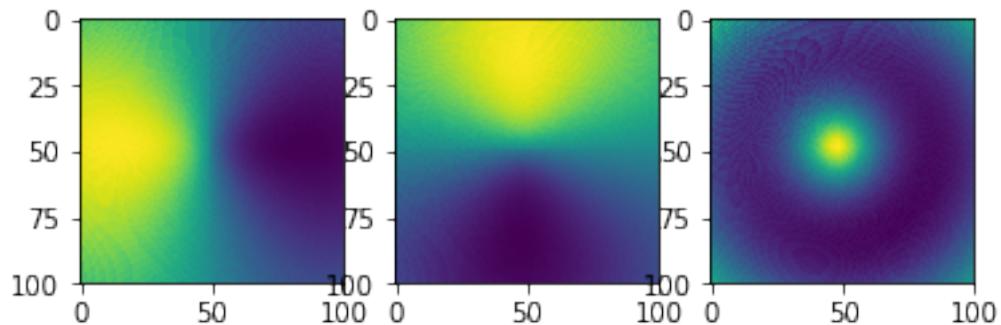
```

Using Synthetic data: im1, im2, im4

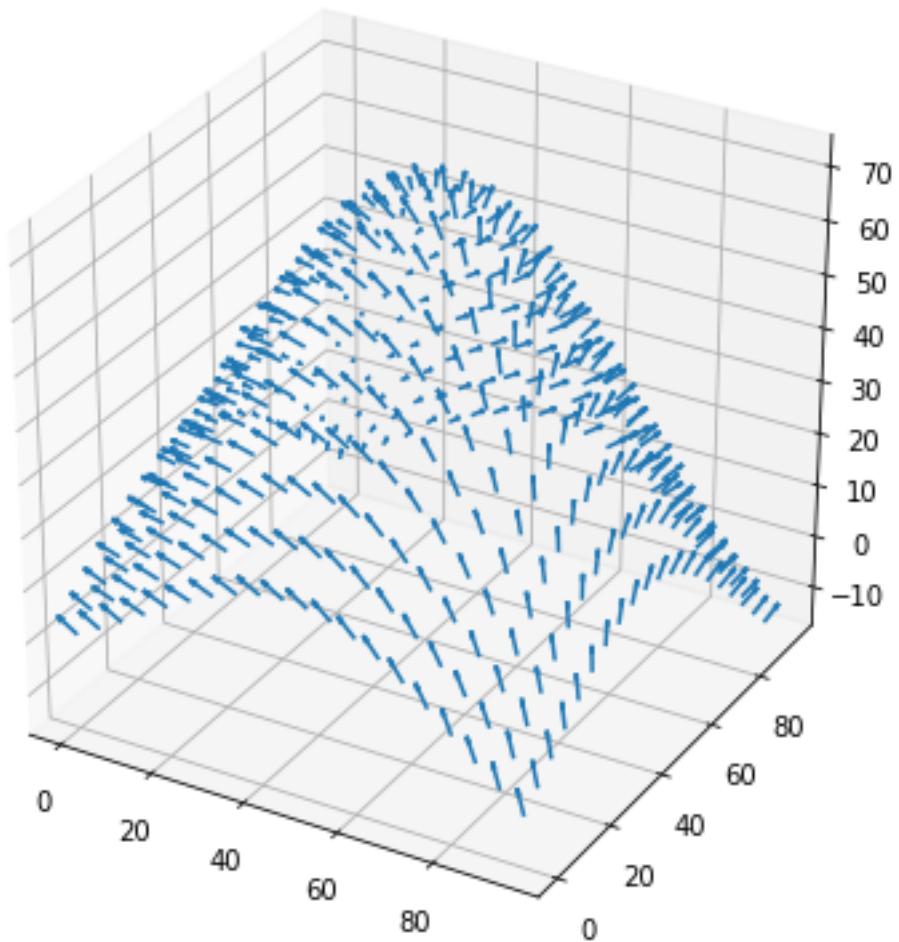
Albedo Map



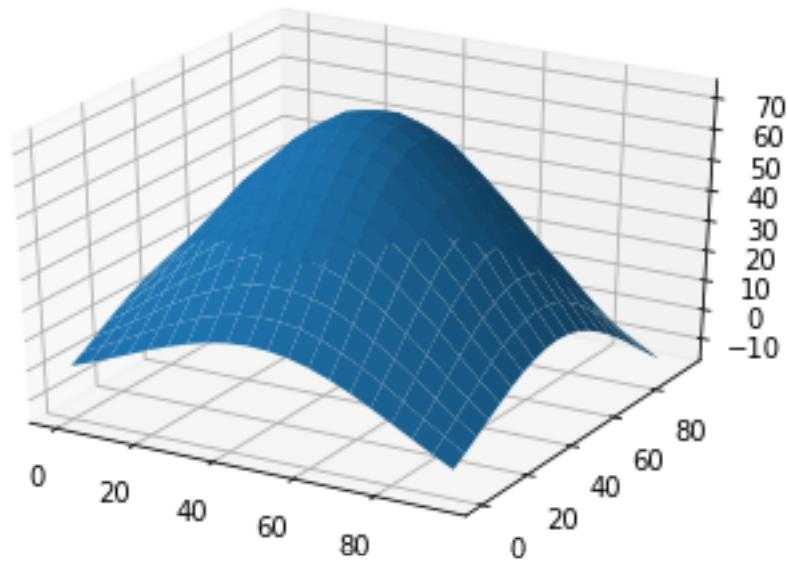
Normal as three separate channels



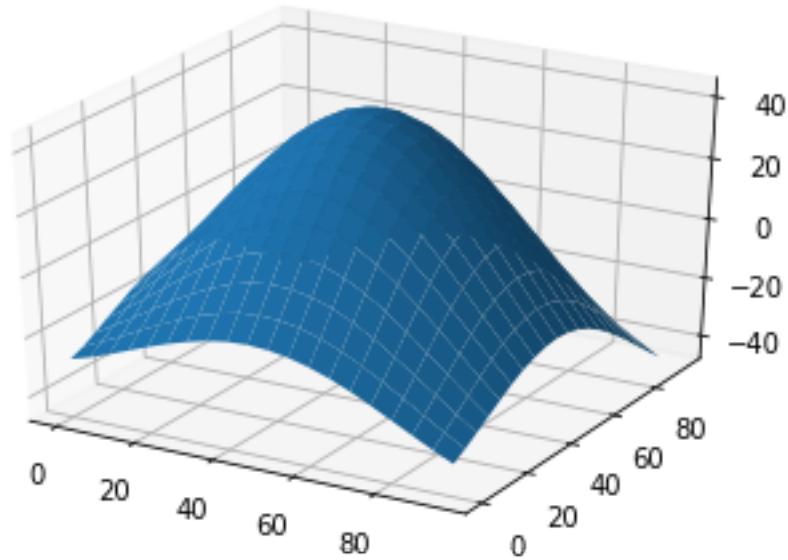
Showing Normal as quiver



Wireframe depth map using scaline approach



Wireframe depth map using Horn



```
In [28]: 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 = photometric_stereo(images, lights, mask, niter=100000)

# -----
# 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.
# -----


# Stride in the plot, you may want to adjust it to different images
stride = 5

print("Using Synthetic data: im1, im2, im3, im4")
print("")
# showing albedo map
print("Showing Albedo map")
fig = plt.figure(figsize=(3, 3))
albedo_max = albedo.max()
albedo = albedo / albedo_max
plt.imshow(albedo, cmap="gray")
plt.show()

print("Showing normal as three separate channels")
# 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()

```

```

print("Showing normal as quiver")
# showing normals as quiver
X, Y, _ = np.meshgrid(np.arange(0,np.shape(normals)[0], 5),
                      np.arange(0,np.shape(normals)[1], 5),
                      np.arange(1))

X = X[..., 0]
Y = Y[..., 0]
Z = depth[::-stride,::stride].T
NX = normals[..., 0][::-stride,::stride].T
NY = normals[..., 1][::-stride,::stride].T
NZ = normals[..., 2][::-stride,::stride].T
fig = plt.figure(figsize=(7, 7))
ax = fig.gca(projection='3d')
plt.quiver(X,Y,Z,NX,NY,NZ, length=5.0)
plt.show()

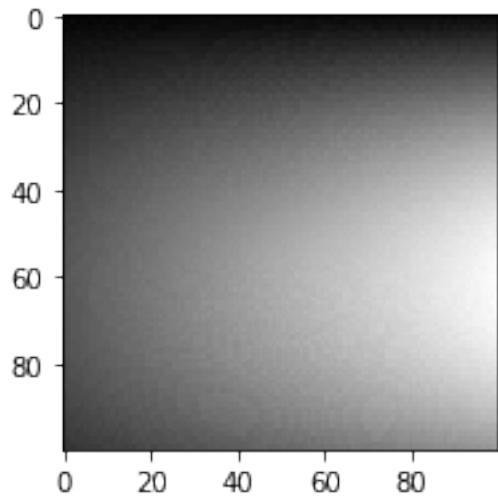
print("Wireframe depth map using scaline approach")
# plotting wireframe depth map
H = depth[::-stride,::stride]
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(X,Y, H.T)
plt.show()

print("Wireframe depth map using Horn")
H = horn[::-stride,::stride]
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(X,Y, H.T)
plt.show()

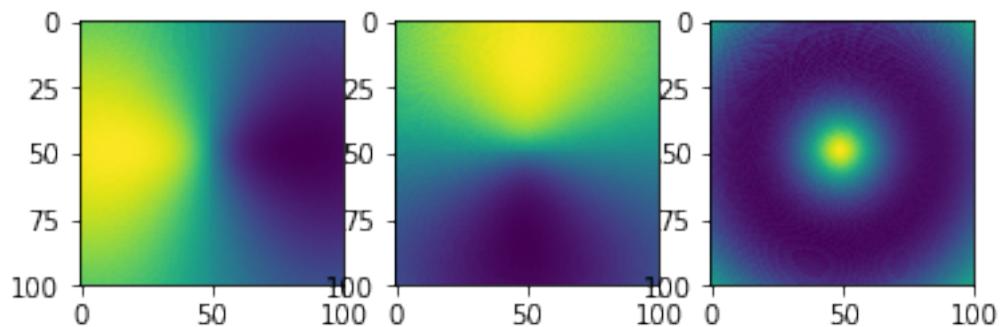
```

Using Synthetic data: im1, im2, im3, im4

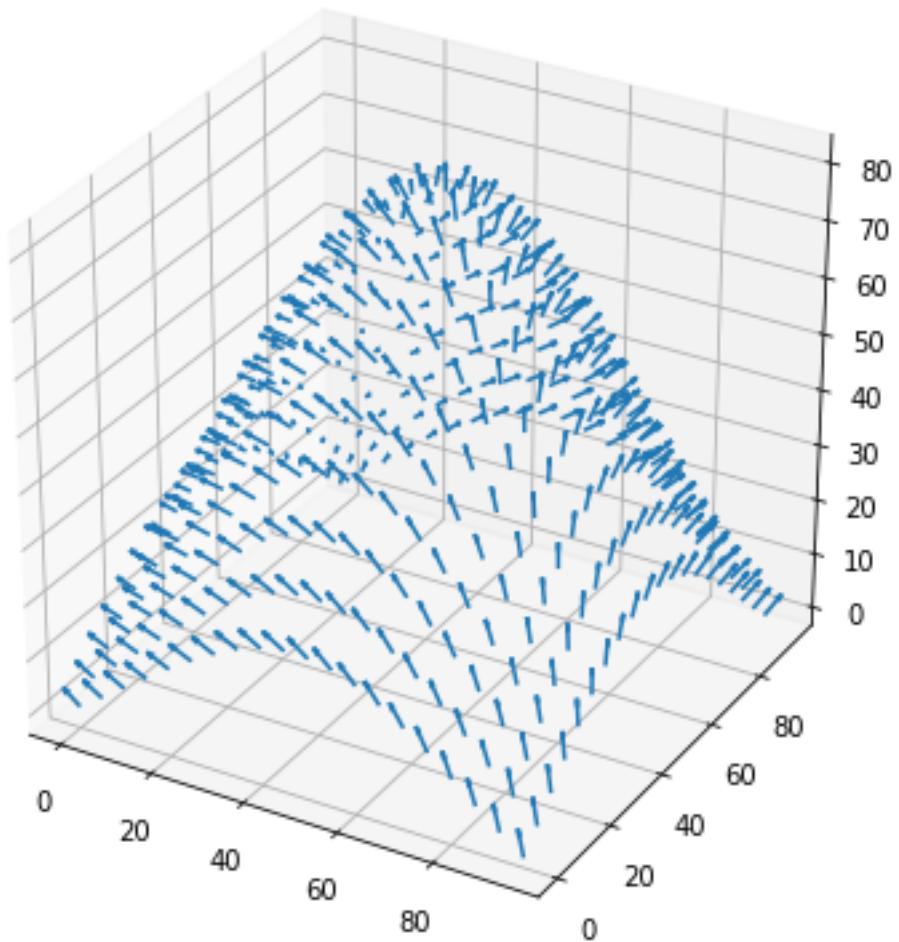
Showing Albedo map



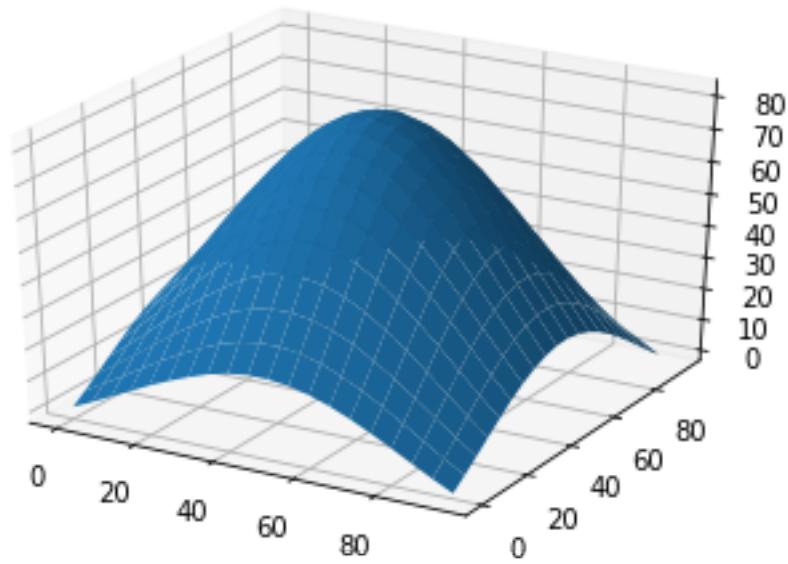
Showing normal as three separate channels



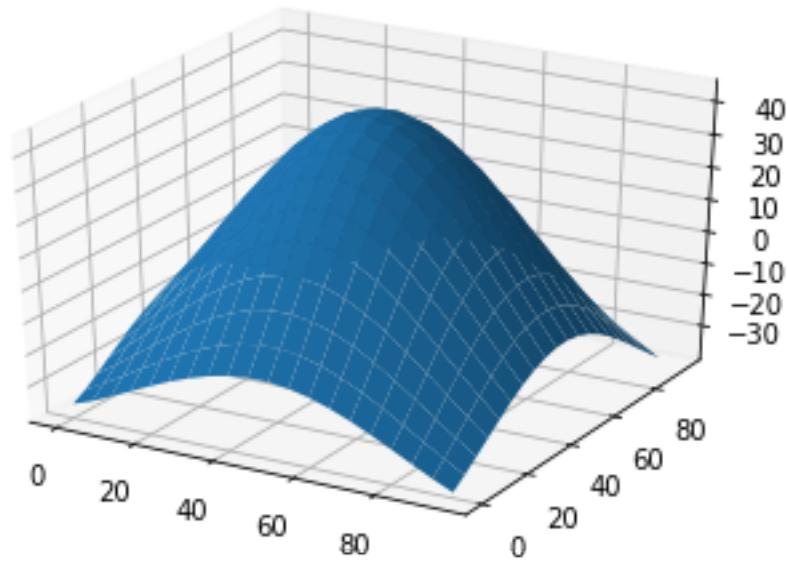
Showing normal as quiver



Wireframe depth map using scaline approach



Wireframe depth map using Horn



### 1.6.3 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 [6]: def normalize_rgb_image(image):  
    minimum = image.flatten().min()  
    maximum = image.flatten().max()  
    delta = maximum - minimum  
    image = (image - minimum)/delta  
  
    return image
```

```
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):
    """
    We will computer S, U, V
    """

    # Specular component
    S = np.ones(I_rgb.shape[:2])
    # Diffuse component
    G = np.ones(I_rgb.shape[:2])
    R = get_rot_mat(rot_vec)
    h, w, _ = I_rgb.shape
    for i in range(0, h):
        for j in range(0, w):
            r, g, b = I_rgb[i, j, :]
            rgb_matrix = np.expand_dims(np.array([r, g, b]).T, axis=1)
            SUV = np.matmul(R, rgb_matrix)
            S[i, j] = SUV[0][0]
            G[i, j] = math.sqrt(SUV[1][0]**2 + SUV[2][0]**2)

    return S, G

```

```

In [29]: # Specular Sphere
pickle_in = open("specular_sphere.pickle", "rb")
# data = pickle.load(pickle_in)
data = pickle.load(pickle_in, encoding="latin1")

# image1 S, G
S, G = RGBToSUV(data["im1"], np.hstack((data["c"])[0][0],
                                           data["c"][1][0],
                                           data["c"][2][0])))

print("Sphere:im1-> Original, Specular, Diffused")
# showing S, G
figure = plt.figure()
ax1 = figure.add_subplot(131)
ax1.imshow(normalize_rgb_image(data["im1"]))
ax2 = figure.add_subplot(132)
ax2.imshow(S, cmap="gray")
ax3 = figure.add_subplot(133)
ax3.imshow(G, cmap="gray")
plt.show()

# image2 S, G
S, G = RGBToSUV(data["im2"], np.hstack((data["c"])[0][0],
                                           data["c"][1][0],
                                           data["c"][2][0])))

```

```

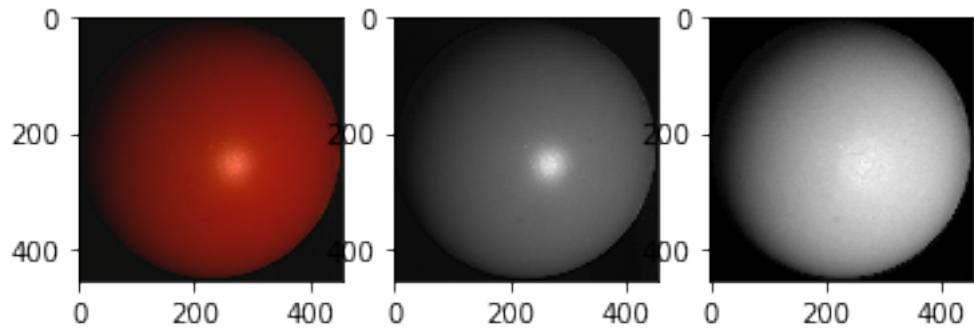
        data[\"c\"] [1] [0] ,
        data[\"c\"] [2] [0])))
print("Sphere:im2-> Original, Specular, Diffused")
# showing S, G
figure = plt.figure()
ax1 = figure.add_subplot(131)
ax1.imshow(normalize_rgb_image(data["im2"]))
ax2 = figure.add_subplot(132)
ax2.imshow(S, cmap="gray")
ax3 = figure.add_subplot(133)
ax3.imshow(G, cmap="gray")
plt.show()

# image1 S, G
S, G = RGBToSUV(data["im3"], np.hstack((data[\"c\"] [0] [0] ,
                                            data[\"c\"] [1] [0] ,
                                            data[\"c\"] [2] [0])))
print("Sphere:im3-> Original, Specular, Diffused")
# showing S, G
figure = plt.figure()
ax1 = figure.add_subplot(131)
ax1.imshow(normalize_rgb_image(data["im3"]))
ax2 = figure.add_subplot(132)
ax2.imshow(S, cmap="gray")
ax3 = figure.add_subplot(133)
ax3.imshow(G, cmap="gray")
plt.show()

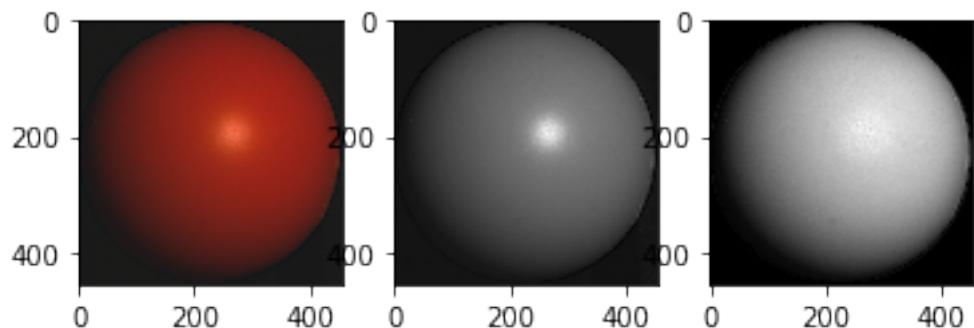
# image1 S, G
S, G = RGBToSUV(data["im4"], np.hstack((data[\"c\"] [0] [0] ,
                                            data[\"c\"] [1] [0] ,
                                            data[\"c\"] [2] [0])))
print("Sphere:im4-> Original, Specular, Diffused")
# showing S, G
figure = plt.figure()
ax1 = figure.add_subplot(131)
ax1.imshow(normalize_rgb_image(data["im4"]))
ax2 = figure.add_subplot(132)
ax2.imshow(S, cmap="gray")
ax3 = figure.add_subplot(133)
ax3.imshow(G, cmap="gray")
plt.show()

```

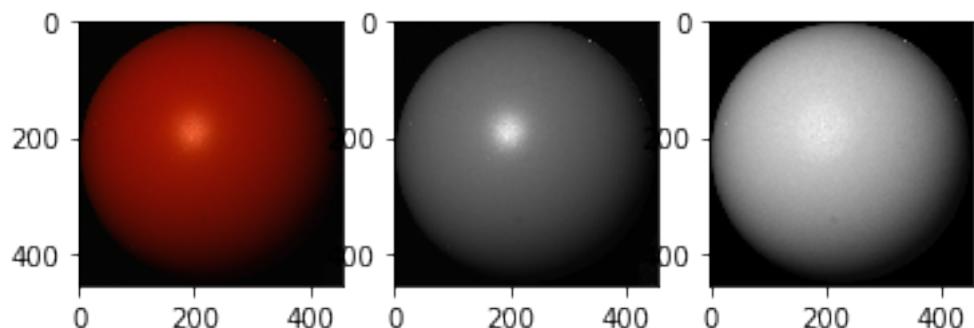
Sphere:im1-> Original, Specular, Diffused



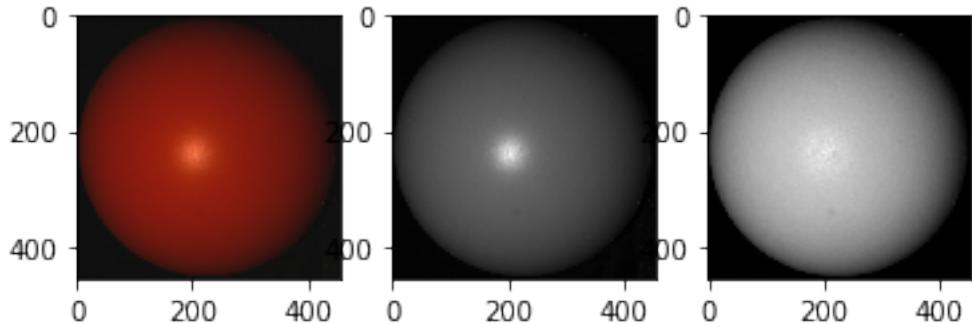
Sphere:im2-> Original, Specular, Diffused



Sphere:im3-> Original, Specular, Diffused



Sphere:im4-> Original, Specular, Diffused



```
In [30]: # Specular Pear
pickle_in = open("specular_pear.pickle", "rb")
# data = pickle.load(pickle_in)
data = pickle.load(pickle_in, encoding="latin1")

# image1 S, G
S, G = RGBToSUV(data["im1"], np.hstack((data["c"] [0] [0],
                                            data["c"] [1] [0],
                                            data["c"] [2] [0])))

print("Pear:im1-> Original, Specular, Diffused")
# showing S, G
figure = plt.figure()
ax1 = figure.add_subplot(131)
ax1.imshow(normalize_rgb_image(data["im1"]))
ax2 = figure.add_subplot(132)
ax2.imshow(S, cmap="gray")
ax3 = figure.add_subplot(133)
ax3.imshow(G, cmap="gray")
plt.show()

# image2 S, G
S, G = RGBToSUV(data["im2"], np.hstack((data["c"] [0] [0],
                                            data["c"] [1] [0],
                                            data["c"] [2] [0])))

print("Pear:im2-> Original, Specular, Diffused")
# showing S, G
figure = plt.figure()
ax1 = figure.add_subplot(131)
ax1.imshow(normalize_rgb_image(data["im2"]))
ax2 = figure.add_subplot(132)
ax2.imshow(S, cmap="gray")
ax3 = figure.add_subplot(133)
```

```

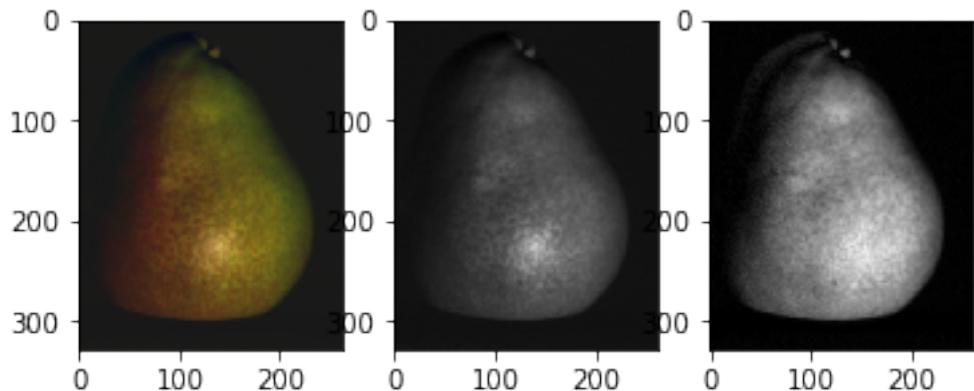
ax3.imshow(G, cmap="gray")
plt.show()

# image1 S, G
S, G = RGBToSUV(data["im3"], np.hstack((data["c"] [0] [0],
                                         data["c"] [1] [0],
                                         data["c"] [2] [0])))
print("Pear:im3-> Original, Specular, Diffused")
# showing S, G
figure = plt.figure()
ax1 = figure.add_subplot(131)
ax1.imshow(normalize_rgb_image(data["im3"]))
ax2 = figure.add_subplot(132)
ax2.imshow(S, cmap="gray")
ax3 = figure.add_subplot(133)
ax3.imshow(G, cmap="gray")
plt.show()

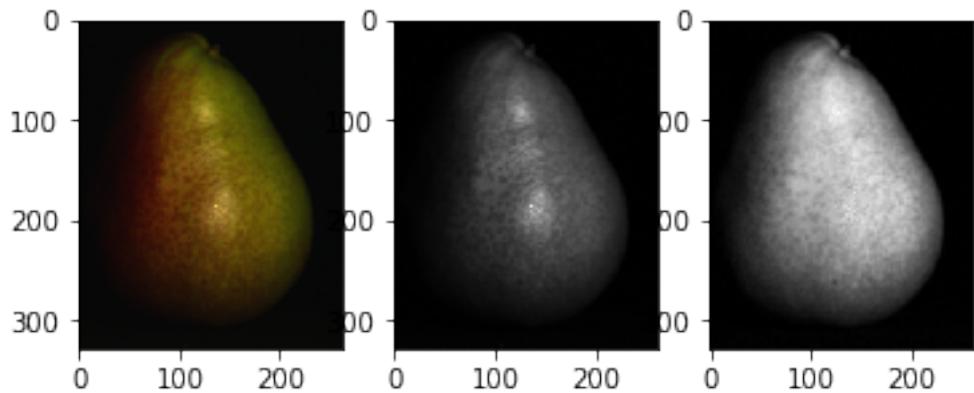
# image1 S, G
S, G = RGBToSUV(data["im4"], np.hstack((data["c"] [0] [0],
                                         data["c"] [1] [0],
                                         data["c"] [2] [0])))
print("Pear:im4-> Original, Specular, Diffused")
# showing S, G
figure = plt.figure()
ax1 = figure.add_subplot(131)
ax1.imshow(normalize_rgb_image(data["im4"]))
ax2 = figure.add_subplot(132)
ax2.imshow(S, cmap="gray")
ax3 = figure.add_subplot(133)
ax3.imshow(G, cmap="gray")
plt.show()

```

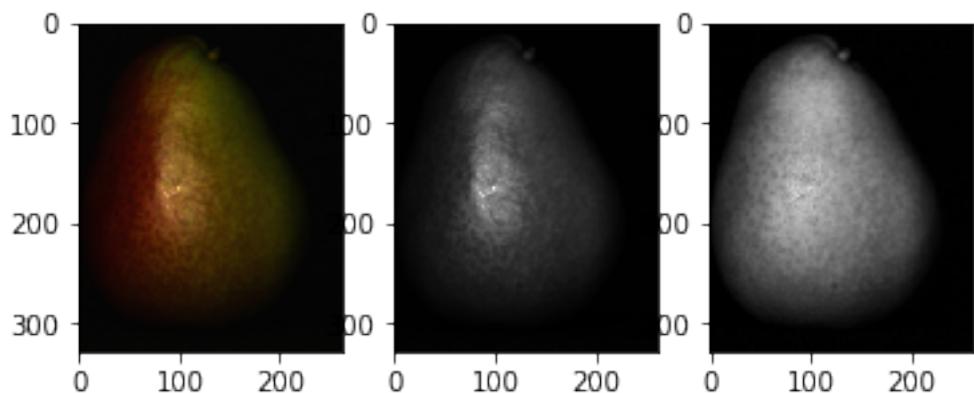
Pear:im1-> Original, Specular, Diffused



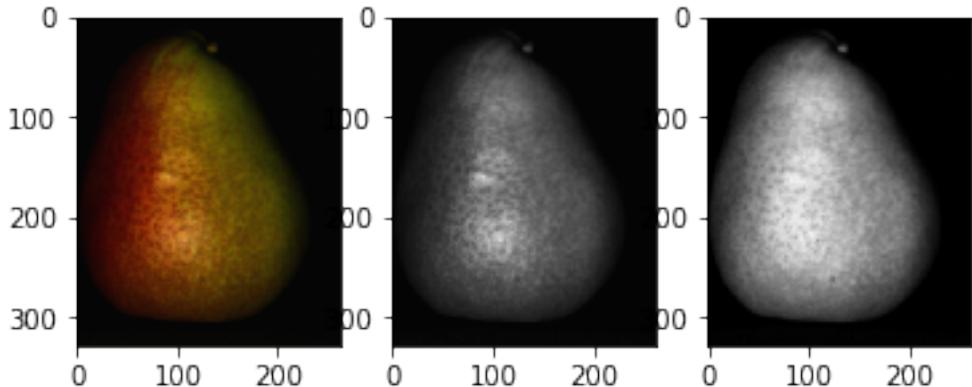
Pear:im2-> Original, Specular, Diffused



Pear:im3-> Original, Specular, Diffused



Pear:im4-> Original, Specular, Diffused



#### 1.6.4 Part 3: [4 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 showing both
  1. Needle map, and
  2. Three images showing components of surface normal
3. A wireframe of depth map (original and diffuse)

#### 1.6.5 Specular Sphere Results

```
In [10]: #####
##### SPHERE
#####
from mpl_toolkits.mplot3d import Axes3D

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

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

# Calculating Albedo, Normal and Depth on original images
images = []
```

```

# convert all RGB to grayscale before feeding into function
images.append(np.dot(data["im1"][:, :, :3], [0.299, 0.587, 0.114]))
images.append(np.dot(data["im2"][:, :, :3], [0.299, 0.587, 0.114]))
images.append(np.dot(data["im3"][:, :, :3], [0.299, 0.587, 0.114]))
images.append(np.dot(data["im4"][:, :, :3], [0.299, 0.587, 0.114]))
images = np.array(images)

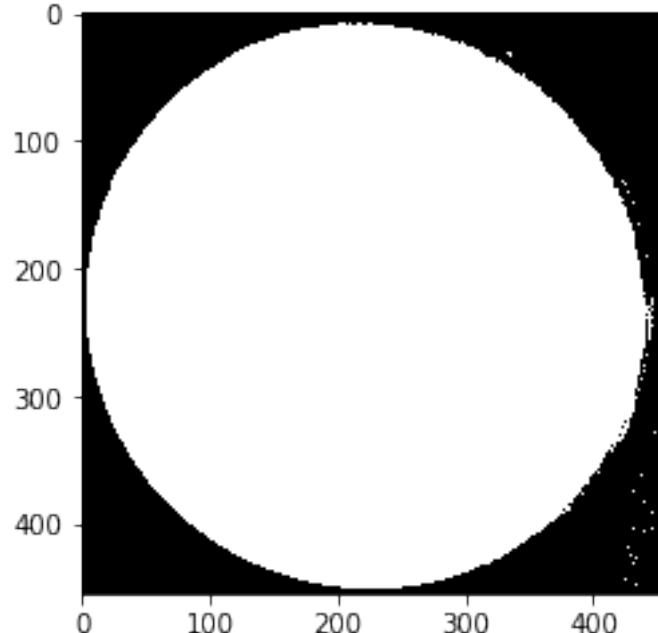
sphere_mask = np.dot(data["im4"][:, :, :3], [0.299, 0.587, 0.114])
mask = np.ones((sphere_mask.shape))
sphere_mask /= sphere_mask.max()

for i in range(sphere_mask.shape[0]):
    for j in range(sphere_mask.shape[1]):
        if sphere_mask[i, j] <= 0.05:
            mask[i, j] = 0.

sphere_mask = mask
orig_albedo_sp, orig_normals_sp, orig_depth_sp, orig_horn_sp = photometric_stereo(
    images, lights, sphere_mask, niter=100000
)
print("Sphere Binary mask")
plt.imshow(sphere_mask, cmap='gray')

```

Out[10]: <matplotlib.image.AxesImage at 0x7f9bd666f128>



In [11]: #####  
##### SPHERE

```

#####
# Calculating Albedo, Normal and Depth on diffused images for Spheres
# Find diffused component for each RGB image
S_1, G_1 = RGBToSUV(data["im1"], np.hstack((data["c"] [0] [0],
                                              data["c"] [1] [0],
                                              data["c"] [2] [0])))
S_2, G_2 = RGBToSUV(data["im2"], np.hstack((data["c"] [0] [0],
                                              data["c"] [1] [0],
                                              data["c"] [2] [0])))
S_3, G_3 = RGBToSUV(data["im3"], np.hstack((data["c"] [0] [0],
                                              data["c"] [1] [0],
                                              data["c"] [2] [0])))
S_4, G_4 = RGBToSUV(data["im4"], np.hstack((data["c"] [0] [0],
                                              data["c"] [1] [0],
                                              data["c"] [2] [0])))

images = []
images.append(G_1)
images.append(G_2)
images.append(G_3)
images.append(G_4)
images = np.array(images)

albedo_sp, normals_sp, depth_sp, horn_sp = photometric_stereo(
    images, lights, sphere_mask, niter=100000
)

```

In [12]: #####

```

#####
# SPHERE
#####
# Stride in the plot, you may want to adjust it to different images
stride = 5

# showing original albedo
fig = plt.figure(figsize=(3, 3))
orig_albedo_sp = orig_albedo_sp / orig_albedo_sp.max()
plt.imshow(orig_albedo_sp, cmap="gray")
plt.title("Original Albedo")
plt.show()

# showing diffused albedo
fig = plt.figure(figsize=(3, 3))
albedo_sp = albedo_sp / albedo_sp.max()
plt.imshow(albedo_sp, cmap="gray")
plt.title("Diffused Albedo")
plt.show()

# showing normals as three separate channels

```

```

figure = plt.figure()
ax1 = figure.add_subplot(131)
ax1.imshow(orig_normals_sp[..., 0])
ax2 = figure.add_subplot(132)
ax2.imshow(orig_normals_sp[..., 1])
ax3 = figure.add_subplot(133)
ax3.imshow(orig_normals_sp[..., 2])
plt.title("Original Normal Channels")
plt.show()

figure = plt.figure()
ax1 = figure.add_subplot(131)
ax1.imshow(normals_sp[..., 0])
ax2 = figure.add_subplot(132)
ax2.imshow(normals_sp[..., 1])
ax3 = figure.add_subplot(133)
ax3.imshow(normals_sp[..., 2])
plt.title("Diffused Normal Channels")
plt.show()

# showing normals as quiver
# original
orig_X, orig_Y, _ = np.meshgrid(np.arange(0,np.shape(orig_normals_sp)[0], 5),
                                np.arange(0,np.shape(orig_normals_sp)[1], 5),
                                np.arange(1))
orig_X = orig_X[..., 0]
orig_Y = orig_Y[..., 0]
orig_Z = orig_depth_sp[::-stride, ::stride].T
orig_NX = orig_normals_sp[..., 0][::stride, ::-stride].T
orig_NY = orig_normals_sp[..., 1][::-stride, ::stride].T
orig_NZ = orig_normals_sp[..., 2][::stride, ::stride].T
fig = plt.figure(figsize=(7, 7))
ax = fig.gca(projection='3d')
plt.quiver(orig_X,orig_Y,orig_Z,orig_NX,orig_NY,orig_NZ, length=3.0)
plt.title("original Normals quiver")
plt.show()

# diffused
X, Y, _ = np.meshgrid(np.arange(0,np.shape(normals_sp)[0], 5),
                      np.arange(0,np.shape(normals_sp)[1], 5),
                      np.arange(1))
X = X[..., 0]
Y = Y[..., 0]
Z = depth_sp[::-stride, ::stride].T
NX = normals_sp[..., 0][::stride, ::-stride].T
NY = normals_sp[..., 1][::-stride, ::stride].T
NZ = normals_sp[..., 2][::stride, ::stride].T
fig = plt.figure(figsize=(7, 7))

```

```

ax = fig.gca(projection='3d')
plt.quiver(X,Y,Z,NX,NY,NZ, length=3.0)
plt.title("Diffused Normals quiver")
plt.show()

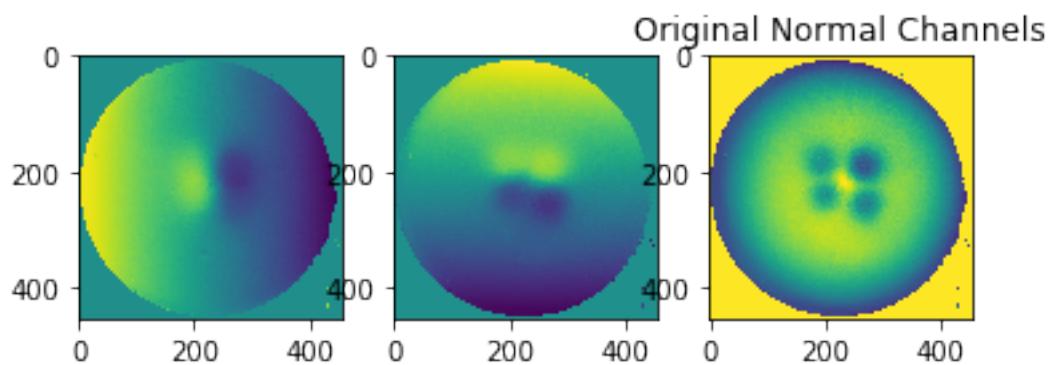
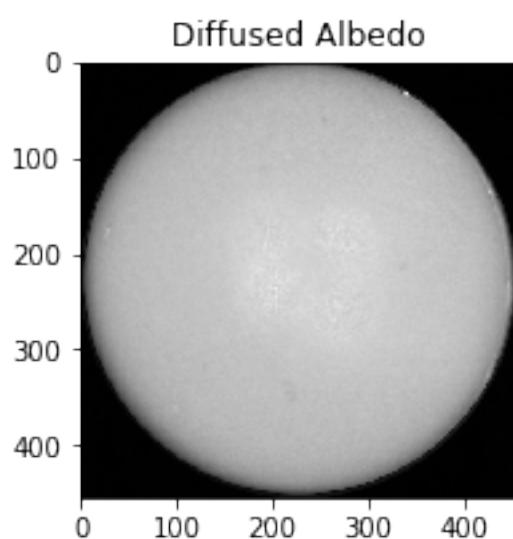
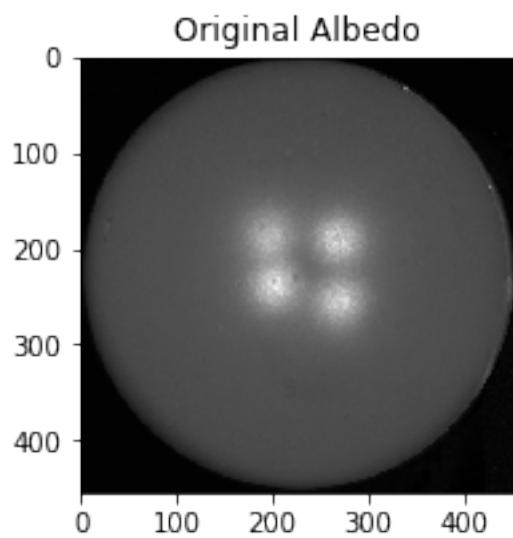
# plotting wireframe depth map
# original
orig_H = orig_depth_sp[::-stride, ::stride]
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(orig_X, orig_Y, orig_H.T)
plt.title("Original wireframe depth map using Scaline approach")
plt.show()

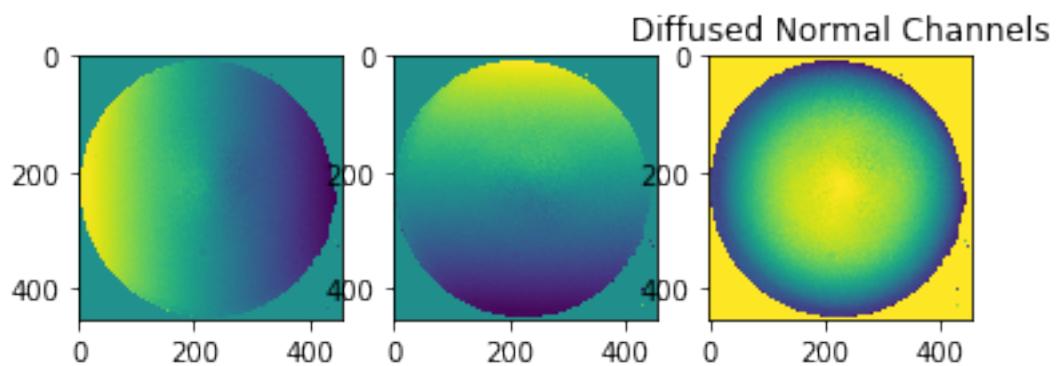
# diffused
H = depth_sp[::-stride, ::stride]
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(X,Y, H.T)
plt.title("Diffused wireframe depth map using Scaline approach")
plt.show()

# plotting wireframe depth map using horns method
# original
orig_H = orig_horn_sp[::-stride, ::stride]
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(orig_X, orig_Y, orig_H.T)
plt.title("Original wireframe depth map using Horns")
plt.show()

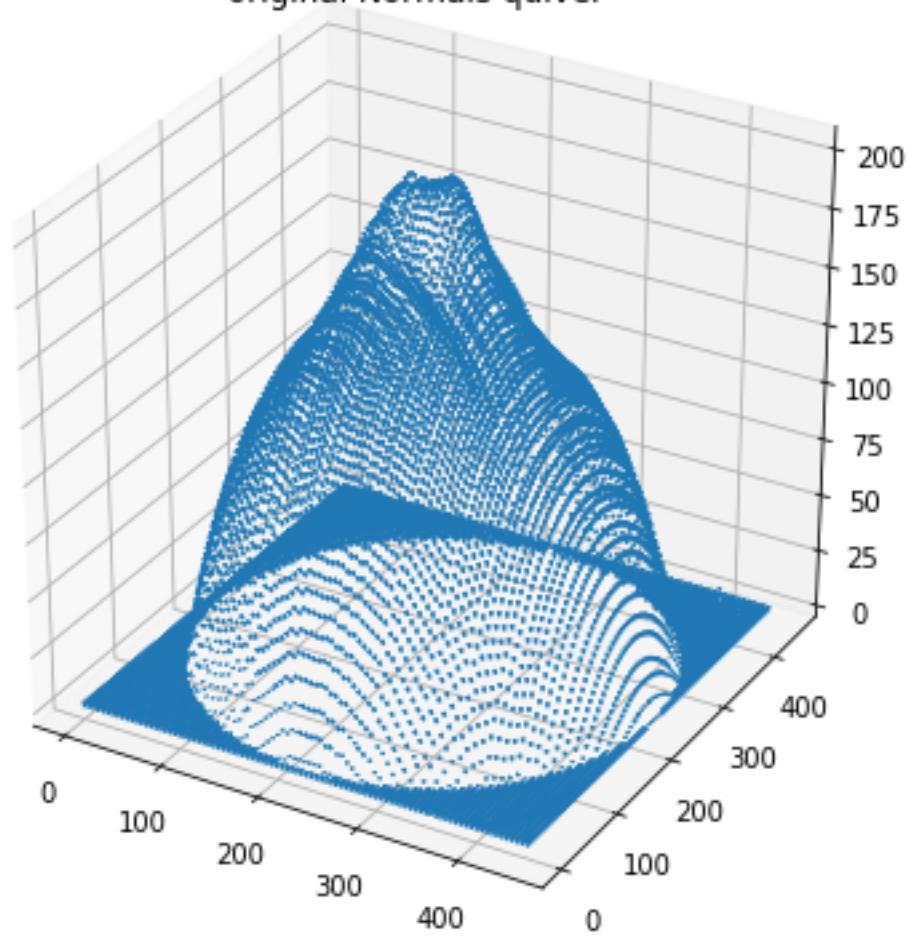
# diffused
H = horn_sp[::-stride, ::stride]
fig = plt.figure()
ax = fig.gca(projection='3d')
plt.title("Diffused wireframe depth map using Horns")
ax.plot_surface(X,Y, H.T)

```

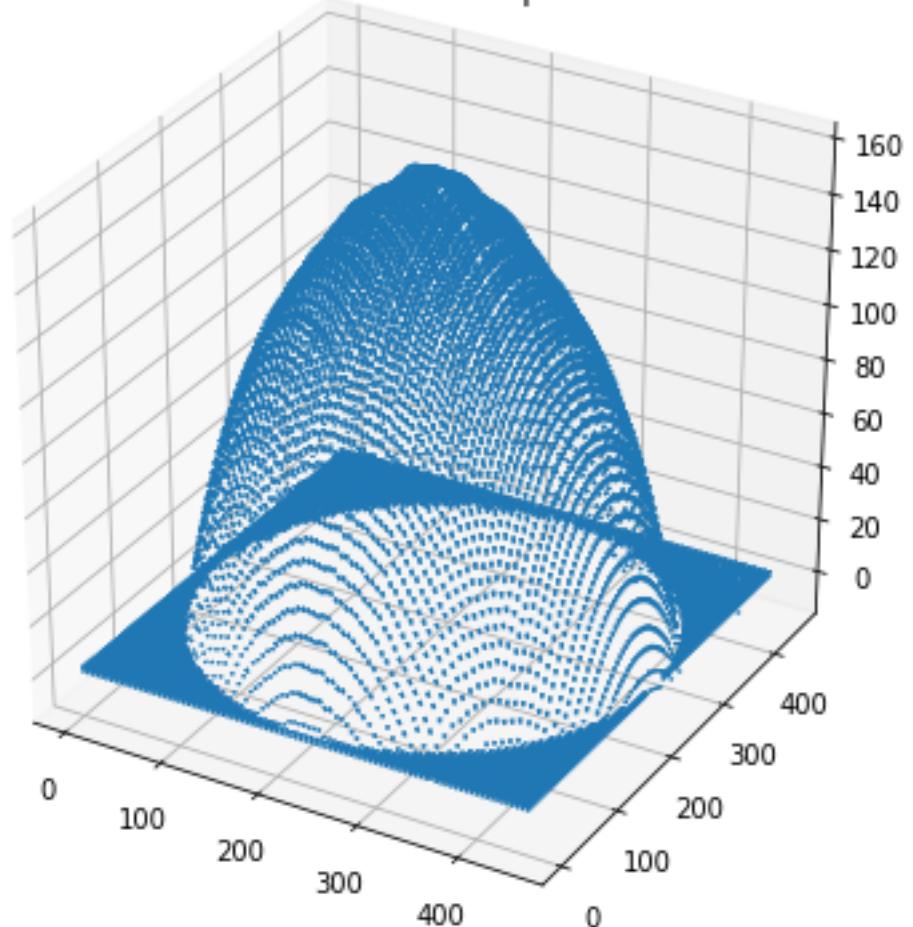




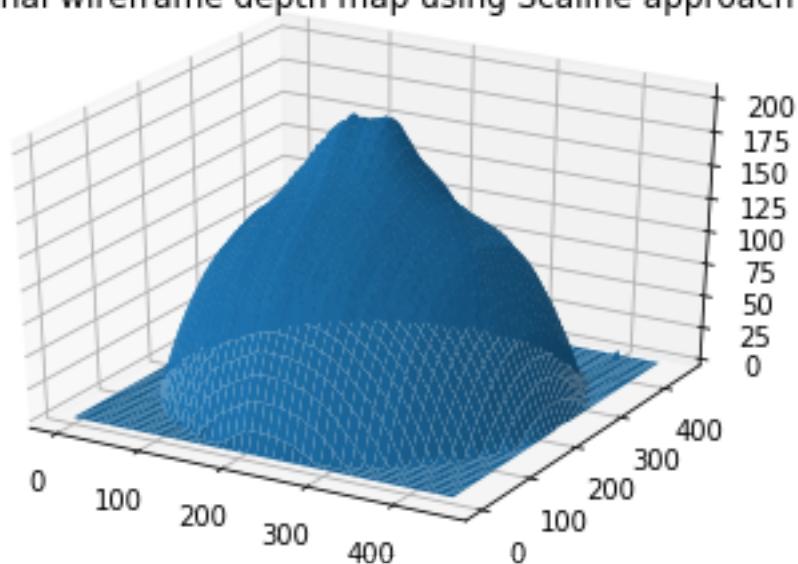
original Normals quiver



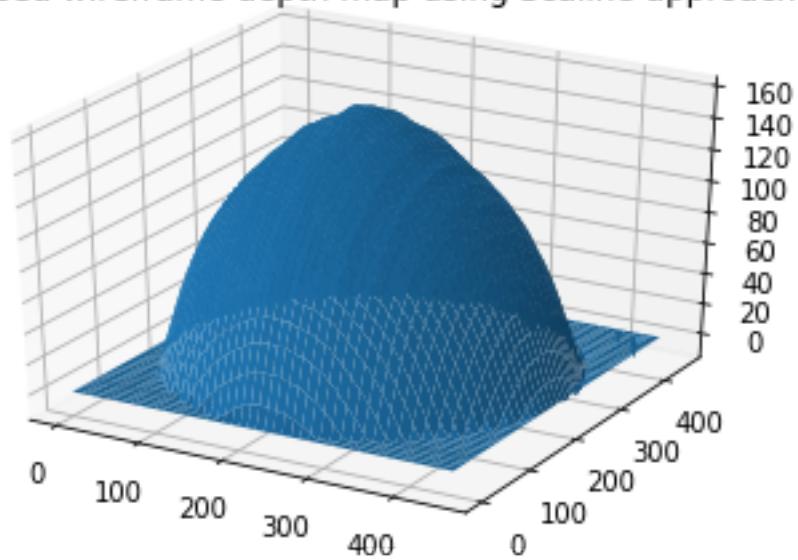
Diffused Normals quiver



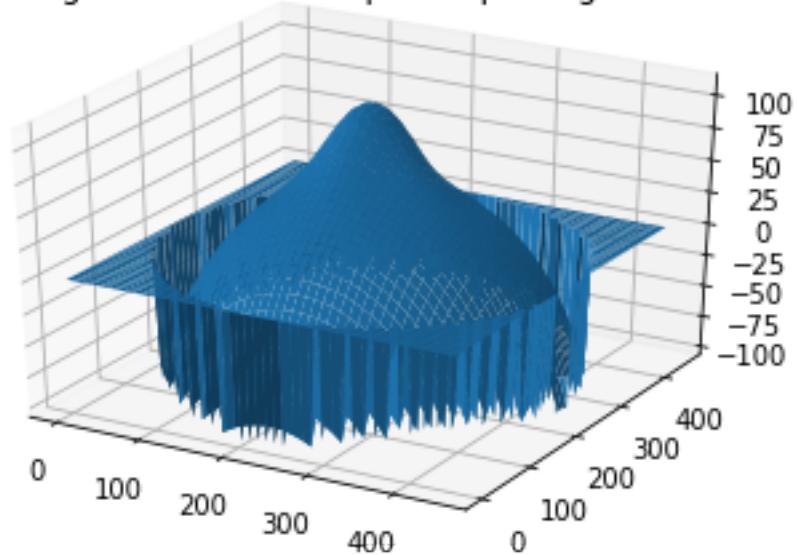
Original wireframe depth map using Scaline approach



Diffused wireframe depth map using Scaline approach

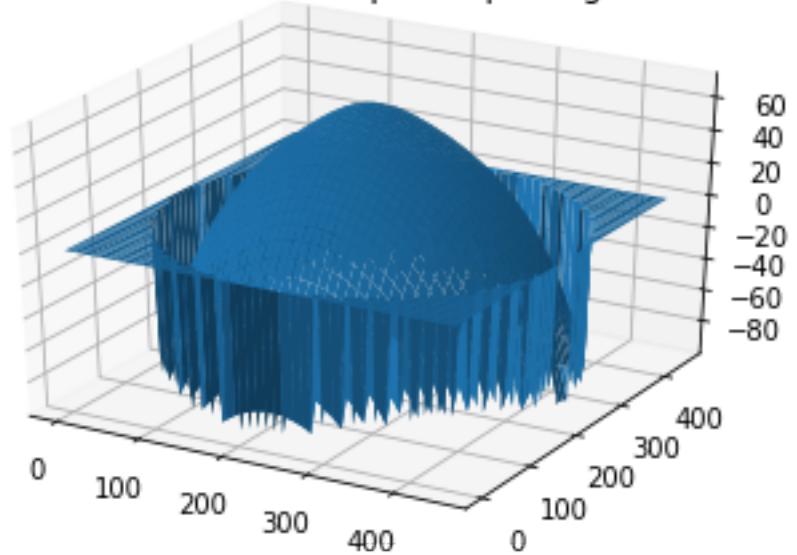


Original wireframe depth map using Horns



Out [12]: <mpl\_toolkits.mplot3d.art3d.Poly3DCollection at 0x7f9bd674f2e8>

Diffused wireframe depth map using Horns



## 1.6.6 Specular Pear Result

In [13]: #####  
##### PEAR

```

#####
from mpl_toolkits.mplot3d import Axes3D

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

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

# Calculating Albedo, Normal and Depth on original images
images = []
# convert all RGB to grayscale before feeding into function
images.append(np.dot(data["im1"][:, :, :3], [0.299, 0.587, 0.114]))
images.append(np.dot(data["im2"][:, :, :3], [0.299, 0.587, 0.114]))
images.append(np.dot(data["im3"][:, :, :3], [0.299, 0.587, 0.114]))
images.append(np.dot(data["im4"][:, :, :3], [0.299, 0.587, 0.114]))
images = np.array(images)

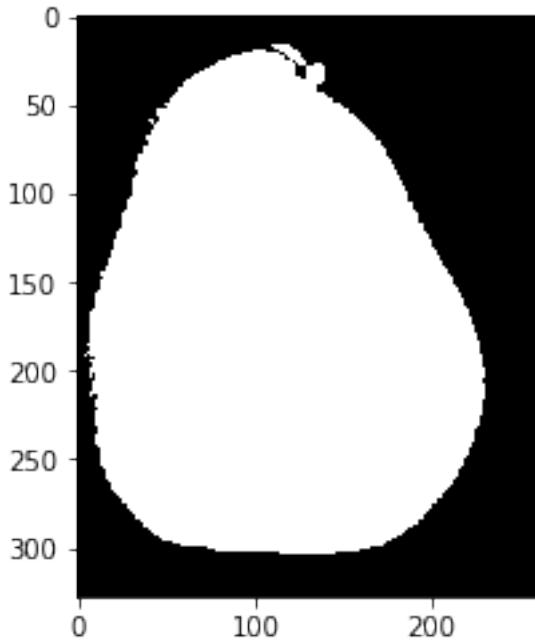
pear_mask = np.dot(data["im4"][:, :, :3], [0.299, 0.587, 0.114])
mask = np.ones((pear_mask.shape))
pear_mask /= pear_mask.max()

for i in range(pear_mask.shape[0]):
    for j in range(pear_mask.shape[1]):
        if pear_mask[i, j] <= 0.035:
            mask[i, j] = 0.

pear_mask = mask
#mask = np.ones((data["im4"].shape[:2]))
orig_albedo_p, orig_normals_p, orig_depth_p, orig_horn_p = photometric_stereo(
    images, lights, pear_mask, niter=150000
)
print("Pear Binary mask")
plt.imshow(pear_mask, cmap='gray')

```

Out[13]: <matplotlib.image.AxesImage at 0x7f9bd67fca90>



```
In [14]: # Calculating Albedo, Normal and Depth on diffused images
# Find diffused component for each RGB image
S_1, G_1 = RGBToSUV(data["im1"], np.hstack((data["c"] [0] [0],
                                              data["c"] [1] [0],
                                              data["c"] [2] [0])))
S_2, G_2 = RGBToSUV(data["im2"], np.hstack((data["c"] [0] [0],
                                              data["c"] [1] [0],
                                              data["c"] [2] [0])))
S_3, G_3 = RGBToSUV(data["im3"], np.hstack((data["c"] [0] [0],
                                              data["c"] [1] [0],
                                              data["c"] [2] [0])))
S_4, G_4 = RGBToSUV(data["im4"], np.hstack((data["c"] [0] [0],
                                              data["c"] [1] [0],
                                              data["c"] [2] [0])))

images = []
images.append(G_1)
images.append(G_2)
images.append(G_3)
images.append(G_4)
images = np.array(images)

albedo_p, normals_p, depth_p, horn_p = photometric_stereo(
    images, lights, pear_mask, niter=150000
)
```

```
In [15]: # Stride in the plot, you may want to adjust it to different images
stride = 5

# showing original albedo
fig = plt.figure(figsize=(3, 3))
orig_albedo_p = orig_albedo_p / orig_albedo_p.max()
plt.imshow(orig_albedo_p, cmap="gray")
plt.title("Original Albedo")
plt.show()

# showing diffused albedo
fig = plt.figure(figsize=(3, 3))
albedo_p = albedo_p / albedo_p.max()
plt.imshow(albedo_p, cmap="gray")
plt.title("Diffused Albedo")
plt.show()

# showing normals as three separate channels
figure = plt.figure()
ax1 = figure.add_subplot(131)
ax1.imshow(orig_normals_p[..., 0])
ax2 = figure.add_subplot(132)
ax2.imshow(orig_normals_p[..., 1])
ax3 = figure.add_subplot(133)
ax3.imshow(orig_normals_p[..., 2])
plt.title("Original Normal Channels")
plt.show()

figure = plt.figure()
ax1 = figure.add_subplot(131)
ax1.imshow(normals_p[..., 0])
ax2 = figure.add_subplot(132)
ax2.imshow(normals_p[..., 1])
ax3 = figure.add_subplot(133)
ax3.imshow(normals_p[..., 2])
plt.title("Diffused Normal Channels")
plt.show()

# showing normals as quiver
# original
orig_X, orig_Y, _ = np.meshgrid(np.arange(0,np.shape(orig_normals_p)[0], 5),
                                 np.arange(0,np.shape(orig_normals_p)[1], 5),
                                 np.arange(1))
orig_X = orig_X[..., 0]
orig_Y = orig_Y[..., 0]
orig_Z = orig_depth_p[::-stride, ::stride].T
orig_NX = orig_normals_p[..., 0][::stride, ::-stride].T
orig_NY = orig_normals_p[..., 1][:::-stride, ::stride].T
```

```

orig_NZ = orig_normals_p[..., 2][::stride,::stride].T
fig = plt.figure(figsize=(7, 7))
ax = fig.gca(projection='3d')
plt.quiver(orig_X,orig_Y,orig_Z,orig_NX,orig_NY,orig_NZ, length=10.0)
plt.title("original Normals quiver")
plt.show()

# diffused
X, Y, _ = np.meshgrid(np.arange(0,np.shape(normals_p)[0], 5),
                      np.arange(0,np.shape(normals_p)[1], 5),
                      np.arange(1))
X = X[..., 0]
Y = Y[..., 0]
Z = depth_p[::stride,::stride].T
NX = normals_p[..., 0][::stride,:::-stride].T
NY = normals_p[..., 1][::-stride,::stride].T
NZ = normals_p[..., 2][::stride,::stride].T
fig = plt.figure(figsize=(7, 7))
ax = fig.gca(projection='3d')
plt.quiver(X,Y,Z,NX,NY,NZ, length=10.0)
plt.title("Diffused Normals quiver")
plt.show()

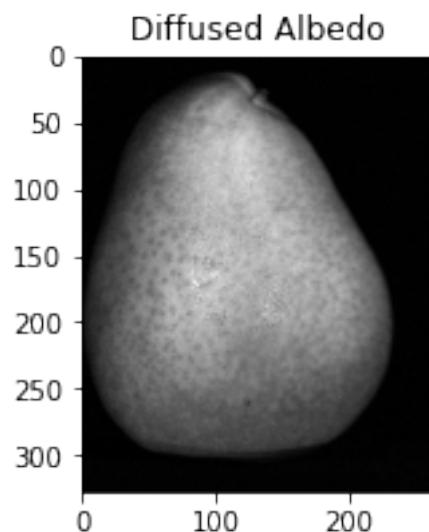
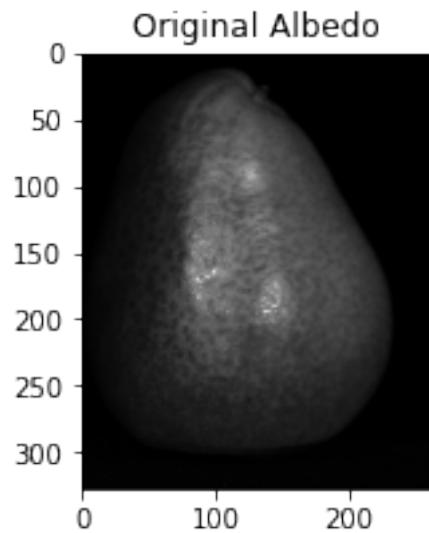
# plotting wireframe depth map
# original
orig_H = orig_depth_p[::stride,::stride]
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(orig_X, orig_Y, orig_H.T)
plt.title("Original wireframe depth map using Scaline approach")
plt.show()

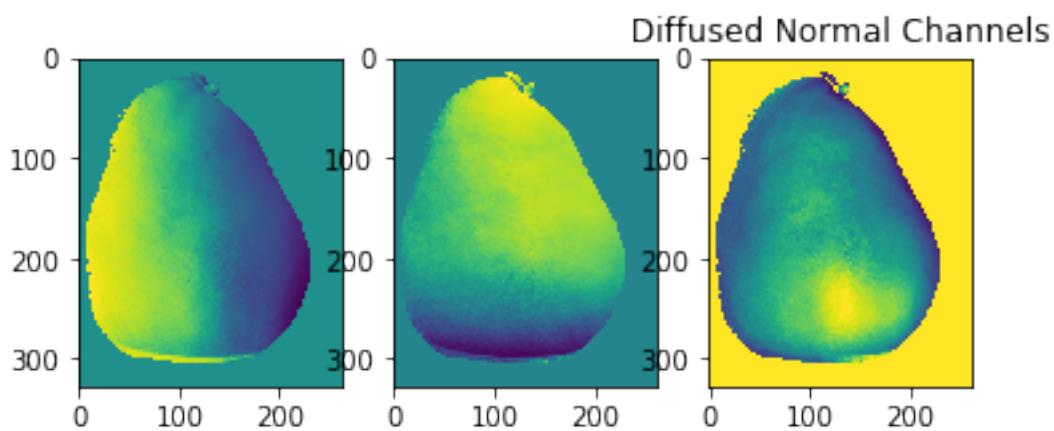
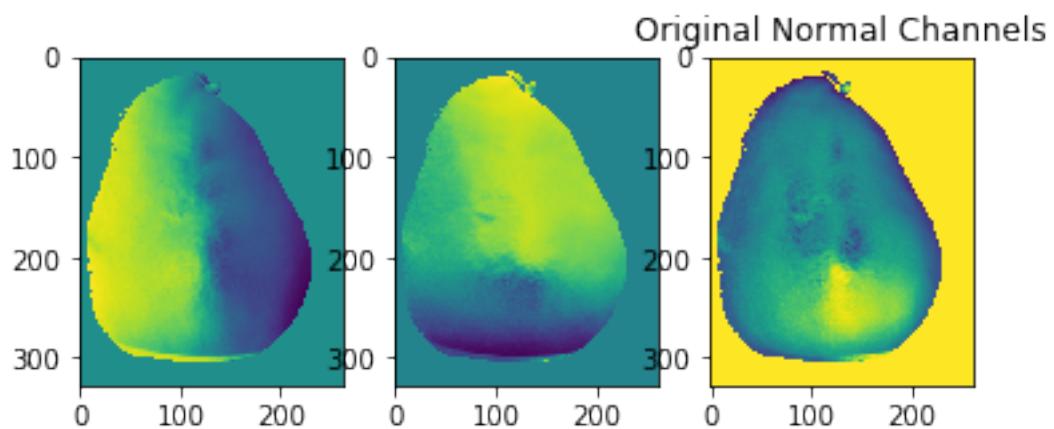
# diffused
H = depth_p[::stride,::stride]
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(X,Y, H.T)
plt.title("Diffused wireframe depth map using Scaline approach")
plt.show()

# plotting wireframe depth map using horns method
# original
orig_H = orig_horn_p[::stride,::stride]
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(orig_X, orig_Y, orig_H.T)
plt.title("Original wireframe depth map using Horns")
plt.show()

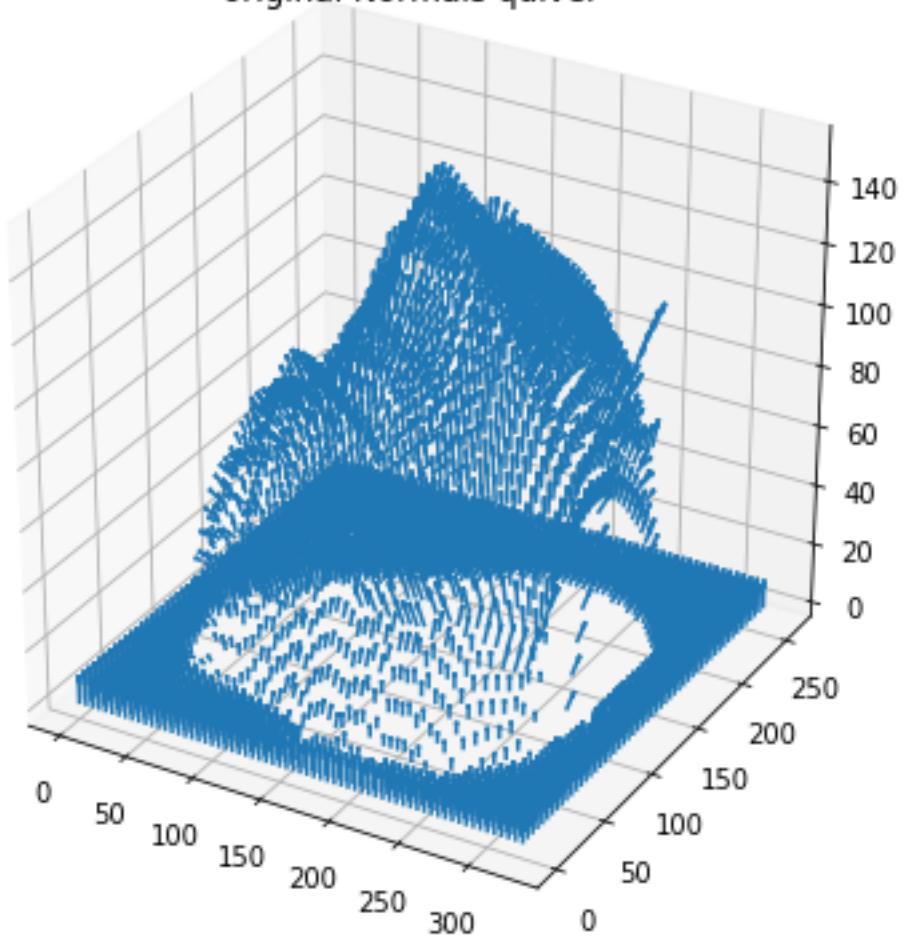
```

```
# diffused
H = horn_p[::-stride, ::stride]
fig = plt.figure()
ax = fig.gca(projection='3d')
plt.title("Diffused wireframe depth map using Horns")
ax.plot_surface(X, Y, H.T)
```

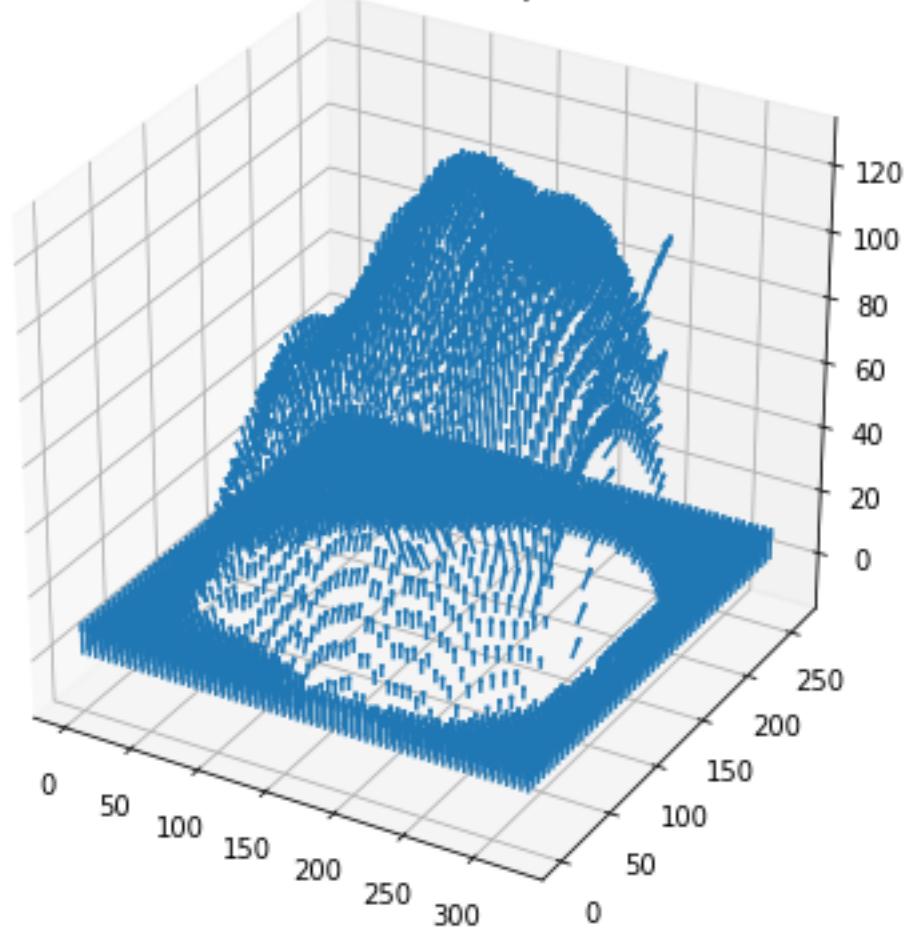




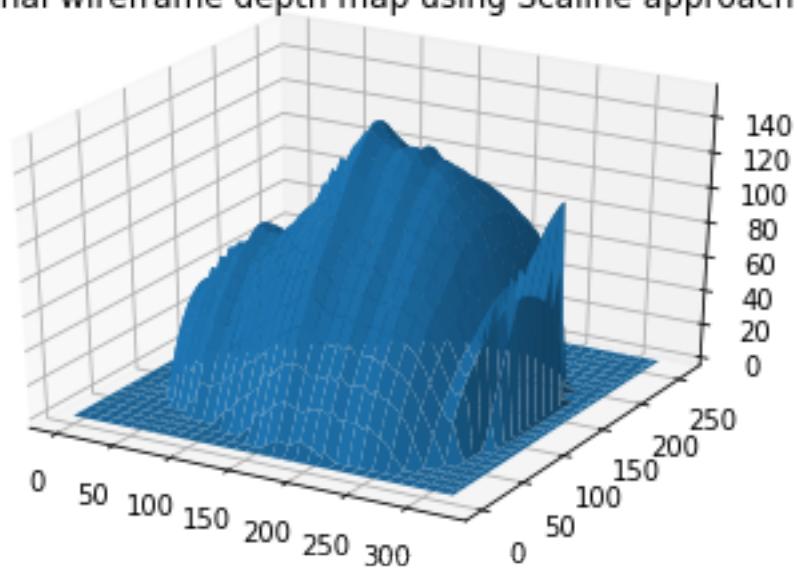
original Normals quiver



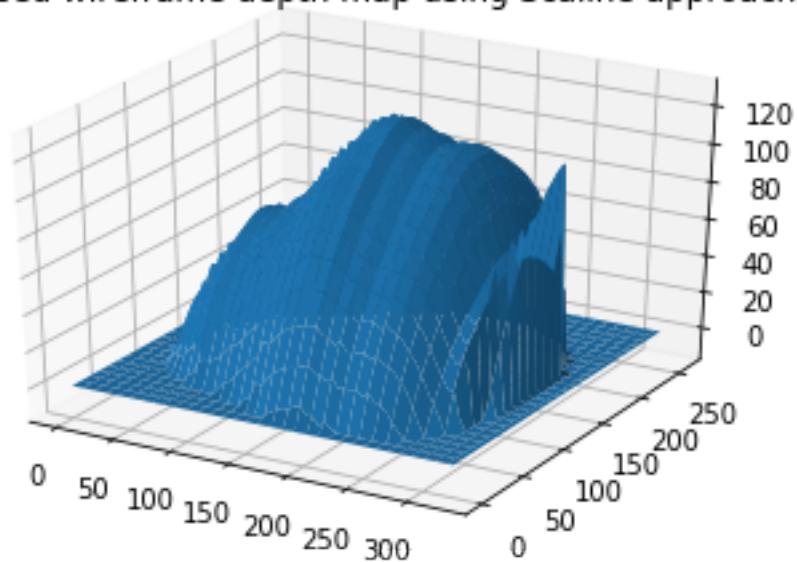
Diffused Normals quiver



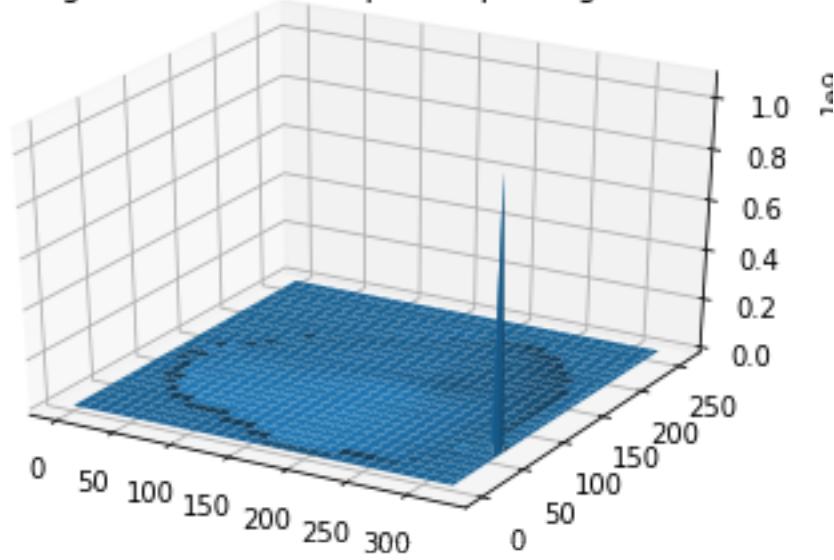
Original wireframe depth map using Scaline approach



Diffused wireframe depth map using Scaline approach

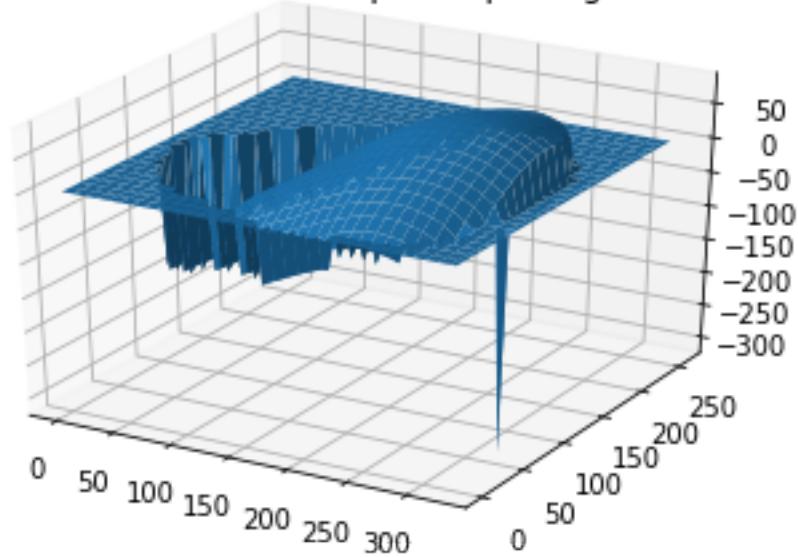


Original wireframe depth map using Horns



```
Out[15]: <mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x7f9bd81a9f60>
```

Diffused wireframe depth map using Horns



## 1.7 Problem 6: Surface Rendering [10 pts]

In this portion of the assignment we will be exploring different methods of approximating local illumination of objects in a scene. As discovered in the photometric stereo portion of this home-

work, we know that different light models work better with different view, illumination sources and materials. This last section of the homework will be an exercise in rendering surfaces. Here, you need use the surface normals from Part 3 of Problem 5 to calculate the image intensity of the specular sphere and pear, with various light sources, different materials, and using a number of illumination models. For the sake of simplicity, multiple reflections of light rays, and occlusion of light rays due to object/scene can be ignored.

### 1.7.1 Data

The surface normals of the specular sphere and the pear from Part 3 of Problem 5. For comparison, You should display the rendering results for both normals calculated from the original image and the diffuse components.

Assume that the albedo map is uniform.

### 1.7.2 Lambertian Illumination

One of the simplest models available to render 3D objections with illumination is the Lambertian model. This model finds the apparent brightness to an observer using the direction of the light source  $\mathbf{L}$  and the normal vector on the surface of the object  $\mathbf{N}$ . The brightness intensity at a given point on an object's surface,  $\mathbf{I}_d$ , with a single light source is found using the following relationship:

$$\mathbf{I}_d = \mathbf{L} \cdot \mathbf{N}(I_l \mathbf{C})$$

where,  $\mathbf{C}$  and  $I_l$  are the the color and intensity of the light source respectively.

### 1.7.3 Phong Illumination

One major drawback of Lambertian illumination is that it only considers the diffuse light in its calculation of brightness intensity. One other major component to illumination rendering is the specular component. The specular reflectance is the component of light that is reflected in a single direction, as opposed to all directions, which is the case in diffuse reflectance. One of the most used models to compute surface brightness with specular components is the Phong illumination model. This model combines ambient lighting, diffused reflectance as well as specular reflectance to find the brightness on a surface. Phong shading also considers the material in the scene which is characterized by four values: the ambient reflection constant ( $k_a$ ), the diffuse reflection constant ( $k_d$ ), the specular reflection constant ( $k_s$ ) and  $\alpha$  the Phong constant, which is the 'shininess' of an object. Furthermore, since the specular component produces 'rays', only some of which would be observed by a single observer, the observer's viewing direction ( $\mathbf{V}$ ) must also be known. For some scene with known material parameters with  $M$  light sources the light intensity  $\mathbf{I}_{phong}$  on a surface with normal vector  $\mathbf{N}$  seen from viewing direction  $\mathbf{V}$  can be computed by:

$$\mathbf{I}_{phong} = k_a \mathbf{I}_a + \sum_{m \in M} \{k_d(\mathbf{L}_m \cdot \mathbf{N}) \mathbf{I}_{m,d} + k_s(\mathbf{R}_m \cdot \mathbf{V})^\alpha \mathbf{I}_{m,s}\},$$

$$\mathbf{R}_m = 2\mathbf{N}(\mathbf{L}_m \cdot \mathbf{N}) - \mathbf{L}_m,$$

where  $\mathbf{I}_a$  is the color and intensity of the ambient lighting,  $\mathbf{I}_{m,d}$  and  $\mathbf{I}_{m,s}$  are the color values for the diffuse and specular light of the  $m$ th light source.

### 1.7.4 Rendering

Please complete the following:

1. Write the function `lambertian()` that calculates the Lambertian light intensity given the light direction  $\mathbf{L}$  with color and intensity  $\mathbf{C}$  and  $I_l = 1$ , and normal vector  $\mathbf{N}$ . 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  $(k_a, k_d, k_s, \alpha)$ ,  $\mathbf{V} = (0, 0, 1)^\top$ ,  $\mathbf{N}$  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.

*Hint: To avoid artifacts due to shadows, ensure that any negative intensities found are set to zero.*

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.	$k_a$	$k_d$	$k_s$	$\alpha$
1	0	0.1	0.75	5
2	0	0.5	0.1	5
3	0	0.5	0.5	10

### 1.7.5 Part 1. Lambertian model [4 pts]

```
In [16]: # lights information
light_1 = np.array([[-1./3, 1./3, 1./3]])
light_2 = np.array([[1., 0, 0]])
color_1 = np.array([[1., 1., 1.]])
color_2 = np.array([[1., .5, .5]])
intensity = 1
# materials information
material_1 = [0, 0.1, 0.75, 5]
material_2 = [0, 0.5, 0.1, 5]
material_3 = [0, 0.5, 0.5, 10]
# view vector for Phongs model
view = np.array([[0,0,1]]).T
```

```
In [17]: def lambertian(normals, lights, color, intensity, mask=None):
    '''We will use the generated normals'''
```

```

image = np.ones((normals.shape[0], normals.shape[1], 3))
h, w, _ = normals.shape
for i in range(0, h):
    for j in range(0, w):
        norm = np.expand_dims(normals[i, j, :], axis=1)
        # apply Lambertian model here
        image[i, j, :] = np.dot(lights, norm) * intensity * color
return np.maximum(image, 0)

In [18]: #####
##### SPHERE
#####

##### Normals from original spheres
#light1
orig_sp_lmb_model_lighth1 = lambertian(orig_normals_sp, light_1, color_1, intensity)
#light2
orig_sp_lmb_model_lighth2 = lambertian(orig_normals_sp, light_2, color_2, intensity)

##### Normals from diffused spheres
#light1
sp_lmb_model_lighth1 = lambertian(normals_sp, light_1, color_1, intensity)
#light2
sp_lmb_model_lighth2 = lambertian(normals_sp, light_2, color_2, intensity)

print("Sphere-Lambartian Model:")
print("")
print("Using original normal")
plt.subplot(1,2,1)
plt.imshow(normalize_rgb_image(orig_sp_lmb_model_lighth1))
plt.title("Lighth1")
plt.subplot(1,2,2)
plt.imshow(normalize_rgb_image(orig_sp_lmb_model_lighth2))
plt.title("Lighth2")
plt.show()
print("Using diffused normal")

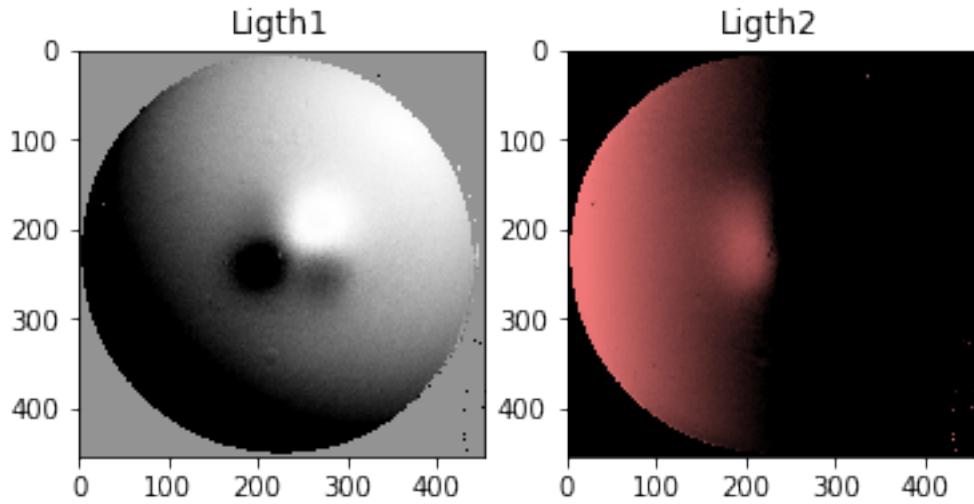
plt.subplot(1,2,1)
plt.imshow(normalize_rgb_image(sp_lmb_model_lighth1))
plt.title("Lighth1")
plt.subplot(1,2,2)
plt.imshow(normalize_rgb_image(sp_lmb_model_lighth2))
plt.title("Lighth2")

plt.show()

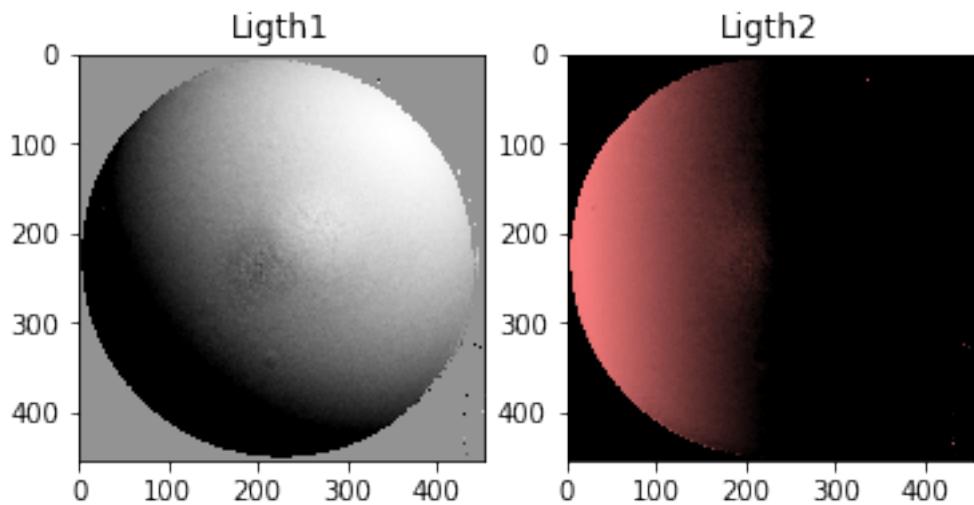
```

Sphere-Lambartian Model:

Using original normal



Using diffused normal



```
In [19]: #####
##### PEAR
#####
#####Normals from original spheres
```

```

#light1
orig_p_lmb_model_lighth1 = lambertian(orig_normals_p, light_1, color_1, intensity)
#light2
orig_p_lmb_model_lighth2 = lambertian(orig_normals_p, light_2, color_2, intensity)

##### Normals from diffused spheres
#light1
p_lmb_model_lighth1 = lambertian(normals_p, light_1, color_1, intensity)
#light2
p_lmb_model_lighth2 = lambertian(normals_p, light_2, color_2, intensity)

print("Pear-Lambartian Model:")
print("")
print("Using original normal")
plt.subplot(1,2,1)
plt.imshow(normalize_rgb_image(orig_p_lmb_model_lighth1))
plt.title("Ligth1")
plt.subplot(1,2,2)
plt.imshow(normalize_rgb_image(orig_p_lmb_model_lighth2))
plt.title("Ligth2")
plt.show()
print("Using diffused normal")

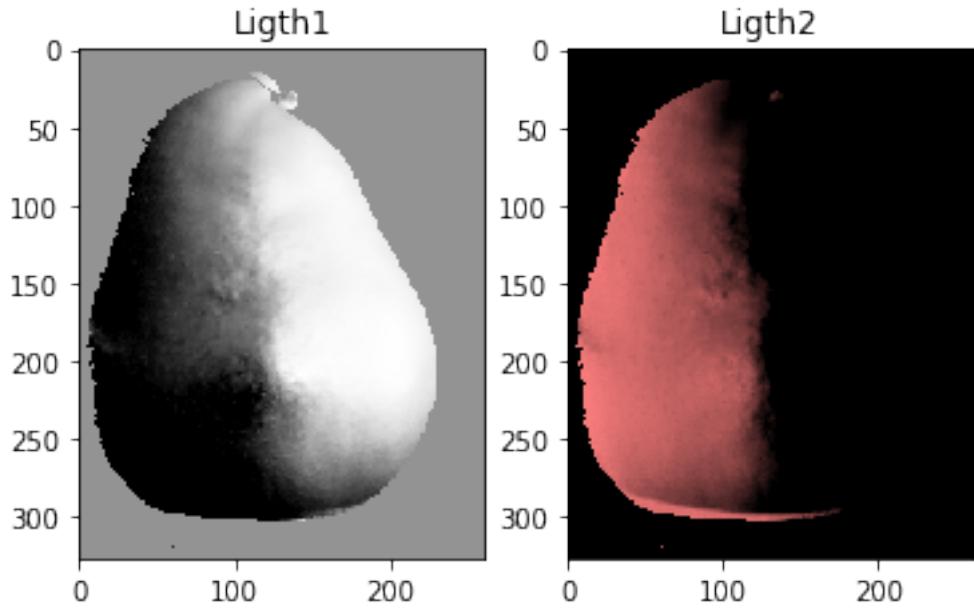
plt.subplot(1,2,1)
plt.imshow(normalize_rgb_image(p_lmb_model_lighth1))
plt.title("Ligth1")
plt.subplot(1,2,2)
plt.imshow(normalize_rgb_image(p_lmb_model_lighth2))
plt.title("Ligth2")

plt.show()

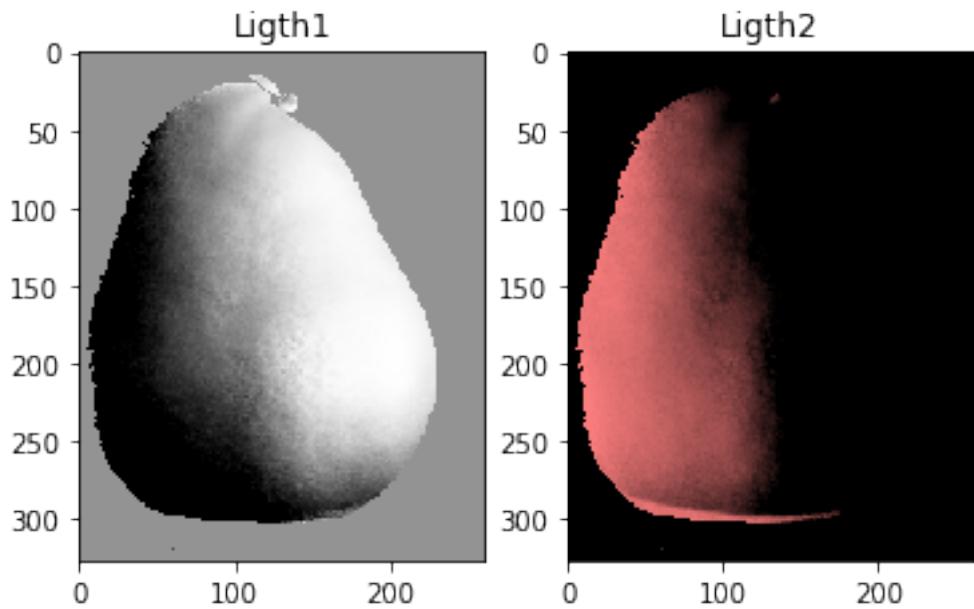
```

Pear-Lambartian Model:

Using original normal



Using diffused normal



### 1.7.6 Part 2. Phong model [6 pts]

```
In [20]: # Phong model
def phong(normals, lights, colors, material, view, mask=None):
```

```

'''Your implementation'''
ka, kd, ks, alpha = material
# first term in the Phong model is ka*Ia(color and intensity)
# of the ambient light source. Since for the given materials
# ka=0 so we can ignore the computation of first term
image = np.ones((normals.shape[0], normals.shape[1], 3))
h, w, _ = normals.shape
for i in range(0, h):
    for j in range(0, w):
        acc = np.array([[0., 0., 0.]])
        # compute for all light sources
        for light, color in zip(lights, colors):
            # normals 3x1
            N = np.expand_dims(normals[i, j, :], axis=1)
            Rm = N.T * 2 * np.dot(light, N) - light
            # KS_term
            KS_term = ks * np.power(np.dot(Rm, view), alpha) * color
            # KD_term
            KD_term = kd * np.dot(light, N) * color
            acc += KD_term + KS_term

        image[i, j, :] = acc

return np.maximum(image, 0)

```

```

In [21]: #####
##### SPHERE: Original Normal
#####
img1_1 = phong(orig_normals_sp, [light_1], color_1, material_1, view)
img1_2 = phong(orig_normals_sp, [light_2], color_2, material_1, view)
img1_3 = phong(orig_normals_sp, [light_1, light_2], [color_1, color_2], material_1, view)

img2_1 = phong(orig_normals_sp, [light_1], color_1, material_2, view)
img2_2 = phong(orig_normals_sp, [light_2], color_2, material_2, view)
img2_3 = phong(orig_normals_sp, [light_1, light_2], [color_1, color_2], material_2, view)

img3_1 = phong(orig_normals_sp, [light_1], color_1, material_3, view)
img3_2 = phong(orig_normals_sp, [light_2], color_2, material_3, view)
img3_3 = phong(orig_normals_sp, [light_1, light_2], [color_1, color_2], material_3, view)

print("Sphere-Phong Model:")
print("")
print("Using original normal and Material-1")

plt.subplot(1,3,1)
plt.imshow(normalize_rgb_image(img1_1))
plt.title("Light1")
plt.subplot(1,3,2)

```

```

plt.imshow(normalize_rgb_image(img1_2))
plt.title("Lighth2")
plt.subplot(1,3,3)
plt.imshow(normalize_rgb_image(img1_3))
plt.title("Lighth1 and Light2")
plt.show()

print("")
print("Using original normal and Material-2")

plt.subplot(1,3,1)
plt.imshow(normalize_rgb_image(img2_1))
plt.title("Lighth1")
plt.subplot(1,3,2)
plt.imshow(normalize_rgb_image(img2_2))
plt.title("Lighth2")
plt.subplot(1,3,3)
plt.imshow(normalize_rgb_image(img2_3))
plt.title("Lighth1 and Light2")
plt.show()

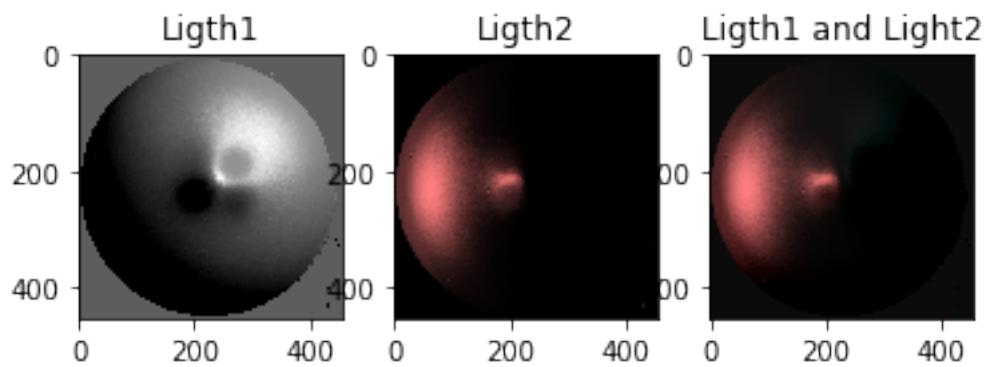
print("")
print("Using original normal and Material-3")

plt.subplot(1,3,1)
plt.imshow(normalize_rgb_image(img3_1))
plt.title("Lighth1")
plt.subplot(1,3,2)
plt.imshow(normalize_rgb_image(img3_2))
plt.title("Lighth2")
plt.subplot(1,3,3)
plt.imshow(normalize_rgb_image(img3_3))
plt.title("Lighth1 and Light2")
plt.show()

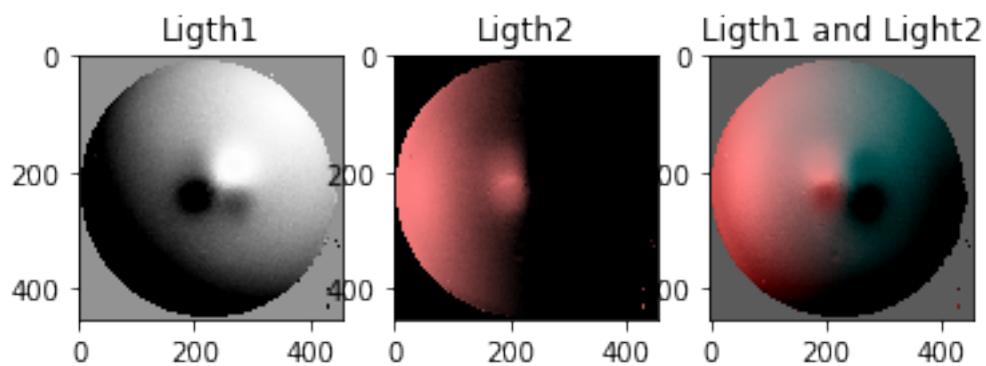
```

Sphere-Phong Model:

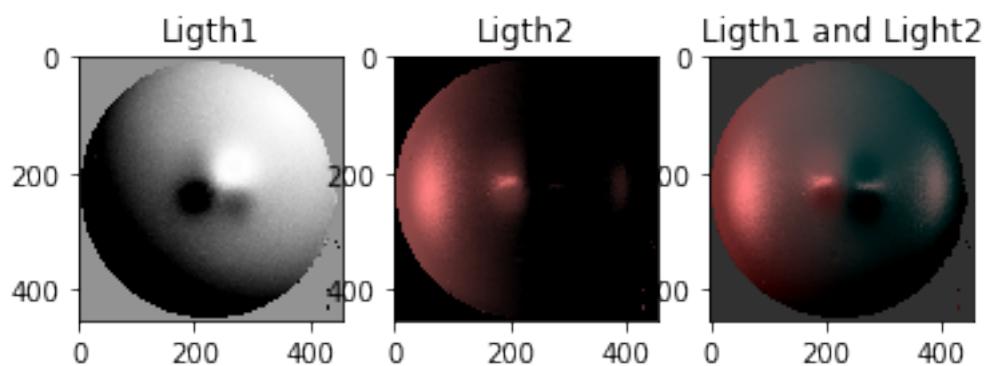
Using original normal and Material-1



Using original normal and Material-2



Using original normal and Material-3



```
In [22]: #####
##### SPHERE: Diffused Normal
#####

img1_1 = phong(normals_sp, [light_1], color_1, material_1, view)
img1_2 = phong(normals_sp, [light_2], color_2, material_1, view)
img1_3 = phong(normals_sp, [light_1, light_2], [color_1, color_2], material_1, view)

img2_1 = phong(normals_sp, [light_1], color_1, material_2, view)
img2_2 = phong(normals_sp, [light_2], color_2, material_2, view)
img2_3 = phong(normals_sp, [light_1, light_2], [color_1, color_2], material_2, view)

img3_1 = phong(normals_sp, [light_1], color_1, material_3, view)
img3_2 = phong(normals_sp, [light_2], color_2, material_3, view)
img3_3 = phong(normals_sp, [light_1, light_2], [color_1, color_2], material_3, view)

print("Sphere-Phong Model:")
print("")
print("Using diffused normal and Material-1")

plt.subplot(1,3,1)
plt.imshow(normalize_rgb_image(img1_1))
plt.title("Ligth1")
plt.subplot(1,3,2)
plt.imshow(normalize_rgb_image(img1_2))
plt.title("Ligth2")
plt.subplot(1,3,3)
plt.imshow(normalize_rgb_image(img1_3))
plt.title("Ligth1 and Light2")
plt.show()

print("")
print("Using diffused normal and Material-2")

plt.subplot(1,3,1)
plt.imshow(normalize_rgb_image(img2_1))
plt.title("Ligth1")
plt.subplot(1,3,2)
plt.imshow(normalize_rgb_image(img2_2))
plt.title("Ligth2")
plt.subplot(1,3,3)
plt.imshow(normalize_rgb_image(img2_3))
plt.title("Ligth1 and Light2")
plt.show()
```

```

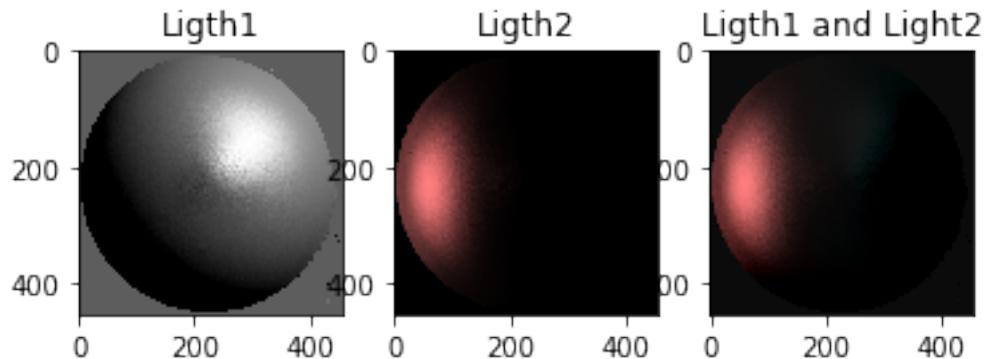
print("")
print("Using diffused normal and Material-3")

plt.subplot(1,3,1)
plt.imshow(normalize_rgb_image(img3_1))
plt.title("Lighth1")
plt.subplot(1,3,2)
plt.imshow(normalize_rgb_image(img3_2))
plt.title("Lighth2")
plt.subplot(1,3,3)
plt.imshow(normalize_rgb_image(img3_3))
plt.title("Lighth1 and Light2")
plt.show()

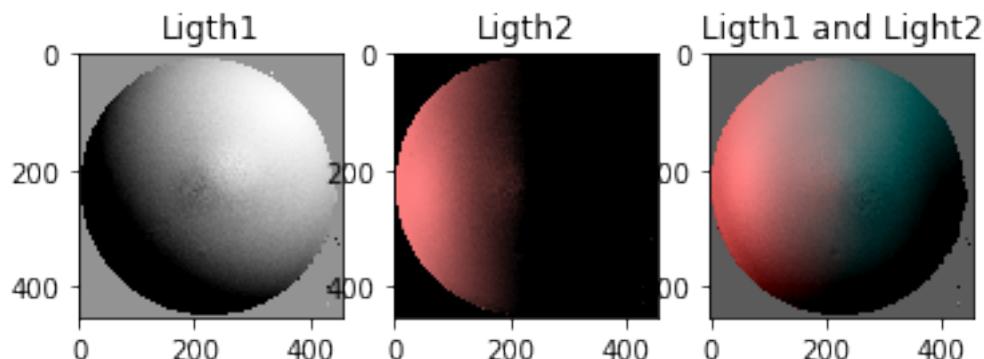
```

Sphere-Phong Model:

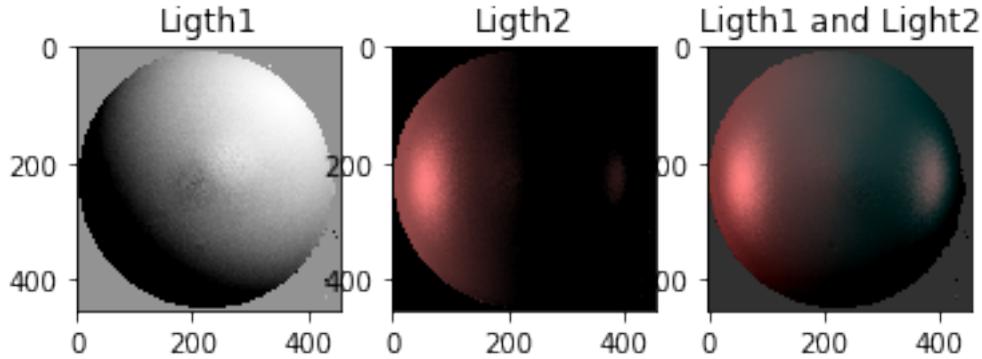
Using diffused normal and Material-1



Using diffused normal and Material-2



Using diffused normal and Material-3



```
In [23]: #####
##### PEAR: Original Normal
#####
img1_1 = phong(orig_normals_p, [light_1], color_1, material_1, view)
img1_2 = phong(orig_normals_p, [light_2], color_2, material_1, view)
img1_3 = phong(orig_normals_p, [light_1, light_2], [color_1, color_2], material_1, view)

img2_1 = phong(orig_normals_p, [light_1], color_1, material_2, view)
img2_2 = phong(orig_normals_p, [light_2], color_2, material_2, view)
img2_3 = phong(orig_normals_p, [light_1, light_2], [color_1, color_2], material_2, view)

img3_1 = phong(orig_normals_p, [light_1], color_1, material_3, view)
img3_2 = phong(orig_normals_p, [light_2], color_2, material_3, view)
img3_3 = phong(orig_normals_p, [light_1, light_2], [color_1, color_2], material_3, view)

print("Pear-Phong Model:")
print("")
print("Using original normal and Material-1")

plt.subplot(1,3,1)
plt.imshow(normalize_rgb_image(img1_1))
plt.title("Lighth1")
plt.subplot(1,3,2)
plt.imshow(normalize_rgb_image(img1_2))
plt.title("Lighth2")
plt.subplot(1,3,3)
plt.imshow(normalize_rgb_image(img1_3))
```

```

plt.title("Ligth1 and Light2")
plt.show()

print("")
print("Using original normal and Material-2")

plt.subplot(1,3,1)
plt.imshow(normalize_rgb_image(img2_1))
plt.title("Ligth1")
plt.subplot(1,3,2)
plt.imshow(normalize_rgb_image(img2_2))
plt.title("Ligth2")
plt.subplot(1,3,3)
plt.imshow(normalize_rgb_image(img2_3))
plt.title("Ligth1 and Light2")
plt.show()

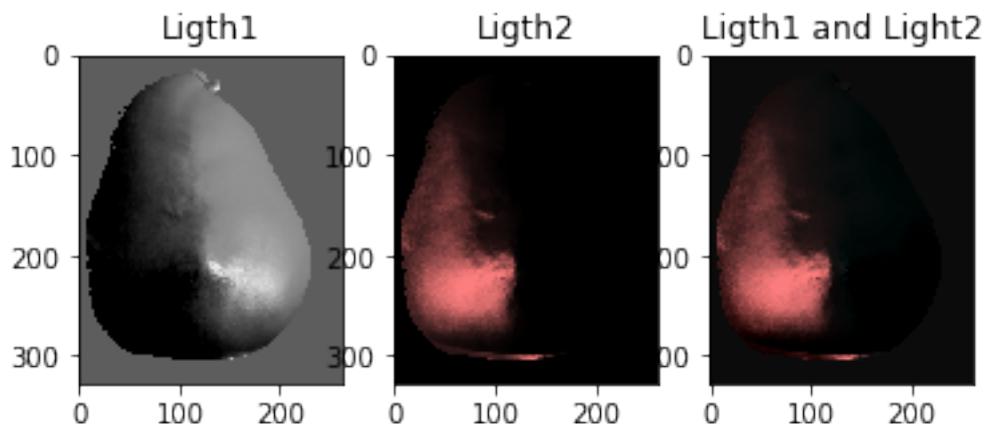
print("")
print("Using original normal and Material-3")

plt.subplot(1,3,1)
plt.imshow(normalize_rgb_image(img3_1))
plt.title("Ligth1")
plt.subplot(1,3,2)
plt.imshow(normalize_rgb_image(img3_2))
plt.title("Ligth2")
plt.subplot(1,3,3)
plt.imshow(normalize_rgb_image(img3_3))
plt.title("Ligth1 and Light2")
plt.show()

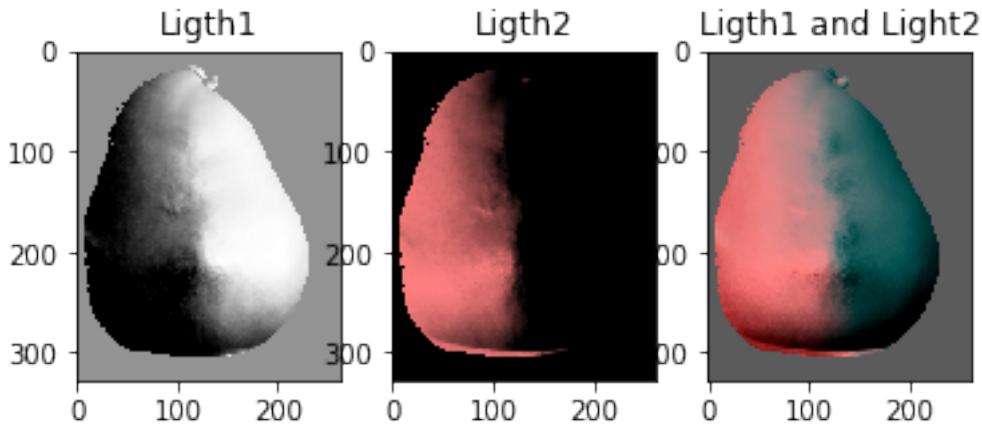
```

Pear-Phong Model:

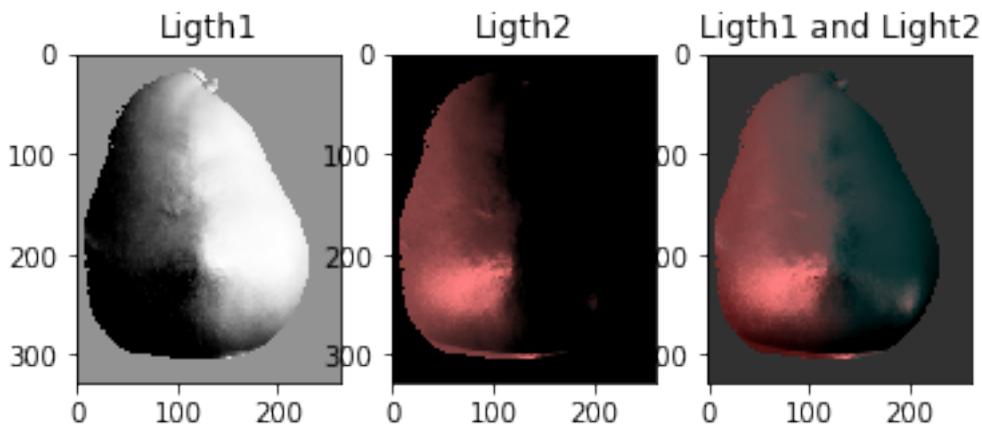
Using original normal and Material-1



Using original normal and Material-2



Using original normal and Material-3



```
In [24]: #####
##### PEAR: Diffused Normal
#####
img1_1 = phong(normals_p, [light_1], color_1, material_1, view)
img1_2 = phong(normals_p, [light_2], color_2, material_1, view)
```

```

img1_3 = phong(normals_p, [light_1, light_2], [color_1, color_2], material_1, view)

img2_1 = phong(normals_p, [light_1], color_1, material_2, view)
img2_2 = phong(normals_p, [light_2], color_2, material_2, view)
img2_3 = phong(normals_p, [light_1, light_2], [color_1, color_2], material_2, view)

img3_1 = phong(normals_p, [light_1], color_1, material_3, view)
img3_2 = phong(normals_p, [light_2], color_2, material_3, view)
img3_3 = phong(normals_p, [light_1, light_2], [color_1, color_2], material_3, view)

print("Pear-Phong Model:")
print("")
print("Using diffused normal and Material-1")

plt.subplot(1,3,1)
plt.imshow(normalize_rgb_image(img1_1))
plt.title("Ligth1")
plt.subplot(1,3,2)
plt.imshow(normalize_rgb_image(img1_2))
plt.title("Ligth2")
plt.subplot(1,3,3)
plt.imshow(normalize_rgb_image(img1_3))
plt.title("Ligth1 and Light2")
plt.show()

print("")
print("Using diffused normal and Material-2")

plt.subplot(1,3,1)
plt.imshow(normalize_rgb_image(img2_1))
plt.title("Ligth1")
plt.subplot(1,3,2)
plt.imshow(normalize_rgb_image(img2_2))
plt.title("Ligth2")
plt.subplot(1,3,3)
plt.imshow(normalize_rgb_image(img2_3))
plt.title("Ligth1 and Light2")
plt.show()

print("")
print("Using diffused normal and Material-3")

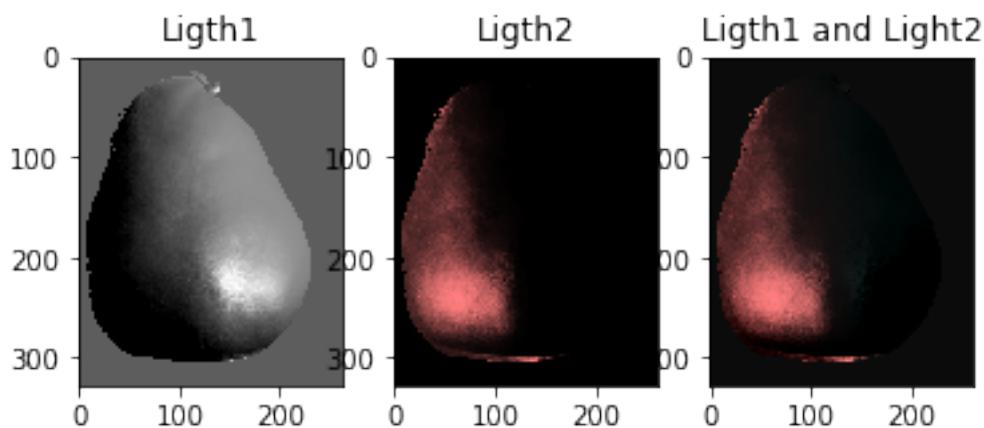
plt.subplot(1,3,1)
plt.imshow(normalize_rgb_image(img3_1))
plt.title("Ligth1")
plt.subplot(1,3,2)
plt.imshow(normalize_rgb_image(img3_2))
plt.title("Ligth2")

```

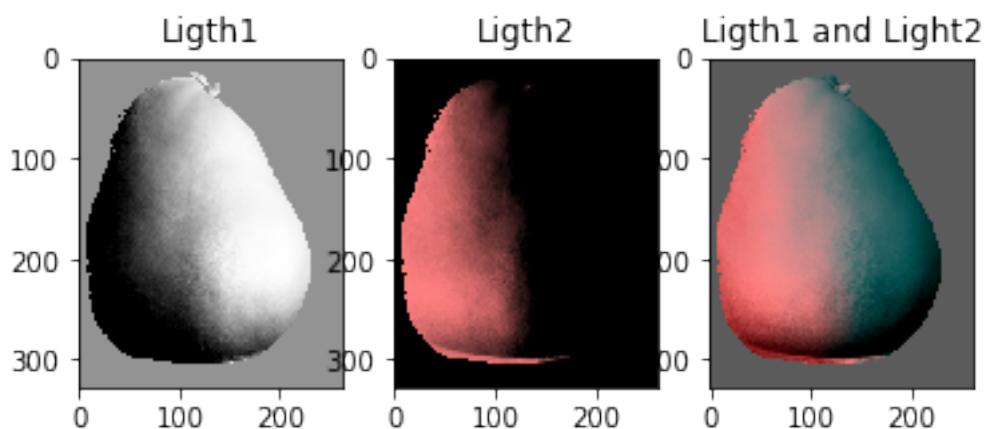
```
plt.subplot(1,3,3)
plt.imshow(normalize_rgb_image(img3_3))
plt.title("Ligth1 and Light2")
plt.show()
```

Pear-Phong Model:

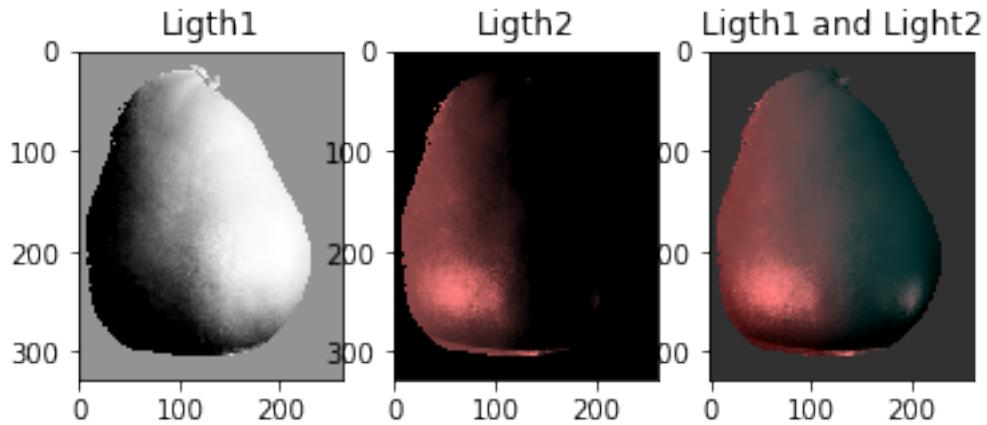
Using diffused normal and Material-1



Using diffused normal and Material-2



Using diffused normal and Material-3



Q.1 For Spherical wedge, given Values are following  
 $\theta = \pi/6$ ;  $\phi = \pi/6$  and it is Centered

around  $\vec{r} = \left( \frac{\sqrt{2}}{4}, \frac{\sqrt{2}}{4}, \frac{\sqrt{3}}{2} \right)$

radius ( $r$ ) = 1

we know solid angle is given by

$$\iint_{\theta=0}^{\pi} \sin \theta d\theta d\phi$$

$\therefore$  Let's find range of angle  $\theta$ !

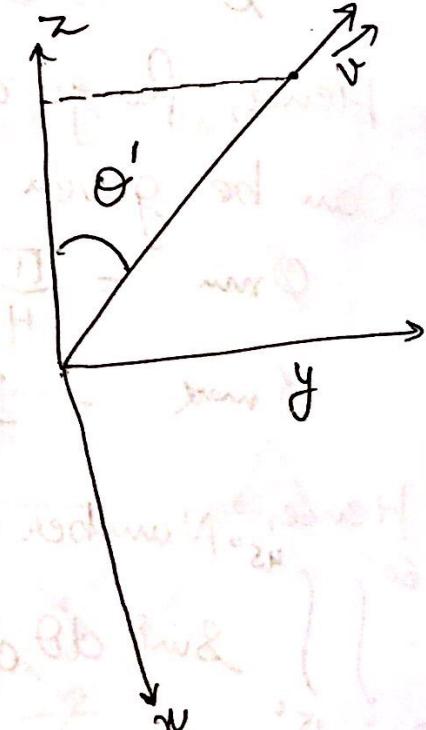
Angle that vector  $\vec{r}$  makes with z-axis is given by!

$$\theta = \cos^{-1} \left( \frac{z}{\sqrt{x^2 + y^2 + z^2}} \right)$$

$$\therefore \theta' = \cos^{-1} \left[ \frac{\sqrt{3}/2}{\sqrt{\left(\frac{\sqrt{2}}{4}\right)^2 + \left(\frac{\sqrt{2}}{4}\right)^2 + \left(\frac{\sqrt{3}}{2}\right)^2}} \right]$$

$$\theta' = \cos^{-1} \left[ \frac{\sqrt{3}/2}{\sqrt{\frac{2}{16} + \frac{2}{16} + \frac{3}{4}}} \right]$$

$$\theta' = \cos^{-1} \left( \frac{\sqrt{3}}{2} \right) \Rightarrow \theta' = 30^\circ = \pi/6$$



②

Since wedge is symmetric about  $\vec{V}$ .

$$\therefore \theta_{\min} = \frac{\pi}{6} - \frac{\pi}{12} \Rightarrow \frac{\pi}{12} = 15^\circ$$

$$\theta_{\max} = \frac{\pi}{6} + \frac{\pi}{12} \Rightarrow \frac{\pi}{4} = 45^\circ$$

$\therefore$  Let's find Range angle  $\phi'$ !

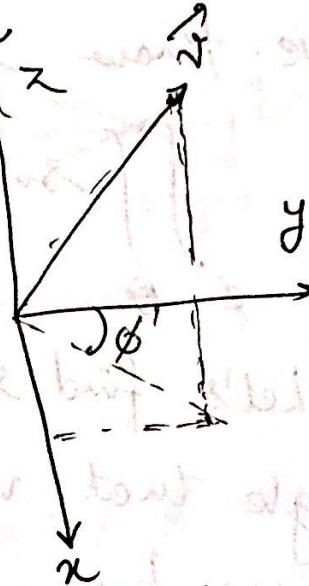
angle that  $\vec{V}$  makes on xy-plane

from y-axis

$$\phi' = \tan^{-1} \left( \frac{x}{y} \right)$$

$$\phi' = \tan^{-1} \left( \frac{\sqrt{2}/4}{\sqrt{2}/4} \right)$$

$$\phi' = 45^\circ = \pi/4$$



Hence, Range of  $\phi$ , Considering Symmetry

Can be given by!

$$\phi_{\min} = \frac{\pi}{4} - \frac{\pi}{12} = 45^\circ - 15^\circ = 30^\circ$$

$$\phi_{\max} = \frac{\pi}{4} + \frac{\pi}{12} = 45^\circ + 15^\circ = 60^\circ$$

Hence, Number of Steradians!

$$\therefore \int_{30^\circ}^{60^\circ} \int_{15^\circ}^{45^\circ} \sin \theta d\theta d\phi \Rightarrow \int_{30^\circ}^{60^\circ} -\cos \theta \Big|_{15^\circ}^{45^\circ} d\phi$$

$$\therefore \int_{30^\circ}^{60^\circ} (\cos 15^\circ - \cos 45^\circ) d\phi \Rightarrow 0.2587 \phi \Big|_{30^\circ}^{60^\circ} \Rightarrow 0.2588 \phi \Big|_{\pi/6}^{\pi/3}$$

$$\therefore (0.2588) \times \frac{\pi}{6} \Rightarrow 0.1355 \text{ str.}$$

Q.2. Radiance emitted by wall =  $L$

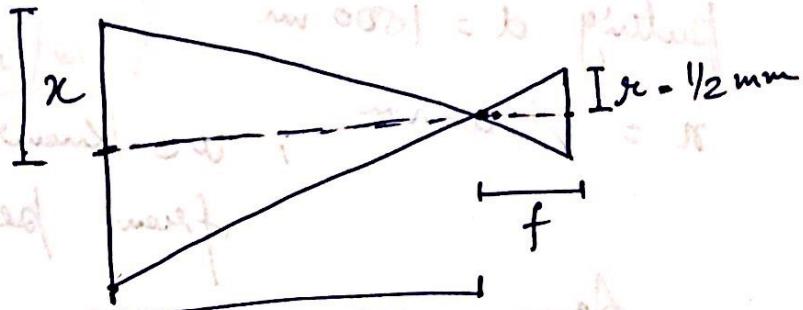
angle b/w Normal & optical axis =  $(\theta) = 60^\circ$

$$(f) = 50 \text{ mm}$$

$$\text{pixels} = 1 \times 1 \text{ mm}$$

I. (A)  $d = 1000 \text{ mm}$

By triangle similarity



$$\therefore \frac{x}{rx} = \frac{d}{f}$$

$$x = (0.5) \frac{d}{f}$$

$$x = (0.5) \frac{(1000)}{50} = 10 \text{ mm} = 0.01 d$$

$$\therefore 2x = 20 \text{ mm} \quad (\text{height of projected area}) = 0.02d$$

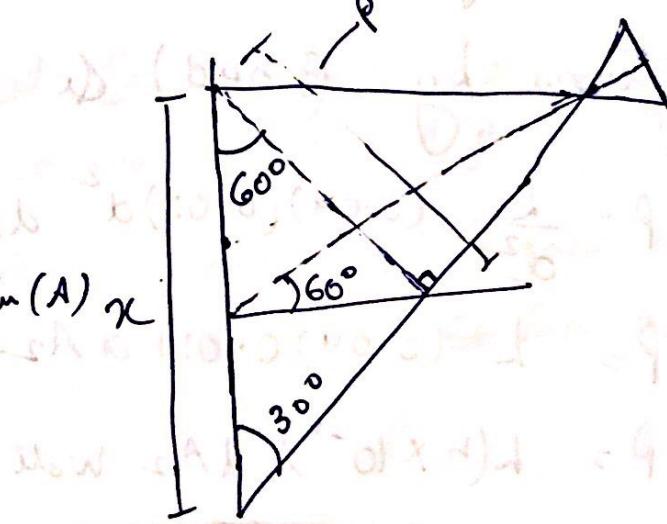
$\therefore$  Projected area will be a square.

$$\therefore \text{Area} = \frac{20 \times 20}{2} = 400 \text{ mm}^2$$

(B)  $f$  from triangle.

$$\sin 30^\circ = \frac{p}{x}$$

and by previous method in (A)  $x$   
ignoring perspective  
distortion.



$$\frac{P/2}{L/2} = \frac{d}{f} \Rightarrow P = \frac{d}{f} \quad \text{Eqn (ii)}$$

~~for (i) = intensity of waves w.r.t. angle~~

$$\therefore x = \frac{P}{8 \sin 30^\circ} = \left( \frac{2d}{f} \right)$$

$$x = \frac{2}{50} d = 0.04d \text{ mm} \quad \text{Eqn (iii)}$$

putting  $d = 1000 \text{ mm}$

~~$x = 40 \text{ mm}$ , we knew width of wall from previous part = 20mm~~

$$\therefore \text{Area} = \frac{40 \times 20}{\text{---}} = 800 \text{ mm}^2$$

(iQ) Power received at  $dA_2$  is given by

$$P = \frac{L}{d^2} dA_1 dA_2 \cos \theta_1 \cos \theta_2 \quad [ \because \theta_2 = 0^\circ ]$$

$$P = \left[ \frac{L}{d^2} \right] dA_1 dA_2 \cos 60^\circ \cdot (1)$$

Since length and width is function of  $d'$

given by  $(0.04d)$  &  $(0.02d)$  respectively

$$\therefore P = \frac{L}{d^2} (0.04)(0.02)d^2 dA_2 \left( \frac{1}{2} \right)$$

$$P = L (0.04)(0.01) dA_2 W$$

$$P = L (4 \times 10^{-4}) dA_2 W \quad \text{--- Eqn (iii)}$$

$$(P) E = \frac{P}{A} \rightarrow \text{area of pixel}$$

(irradiance at

Center

$$\text{pixel}) \therefore E = \frac{L d A_2 (4 \times 10^{-4})}{10^{-6}} \text{ W/m}^2$$

[area of pixel in  $\text{m}^2$ ]

$$\therefore E = L d A_2 (4 \times 10^2) \text{ W/m}^2$$

$$2. \quad \left\{ \begin{array}{l} d = 2000 \text{ mm} \\ d = 4000 \text{ mm} \end{array} \right.$$

$$3. \quad \left\{ \begin{array}{l} d = 2000 \text{ mm} \\ d = 4000 \text{ mm} \end{array} \right.$$

for both cases the irradiance received at Center pixel will be same as before

$$\text{i.e. } E = L d A_2 (4 \times 10^2) \text{ W/m}^2$$

Because 'Power' ( $P$ ) is not a function of distance ' $d$ ' as proved in Qn (iii) and ' $E$ ' depends only on ' $P$ ' & ' $A$ ' (which is same for both)

4. Image irradiance at Center pixel remains constant, irrespective of distance ' $d$ ' b/w pinhole and wall, as ' $d$ ' increases proportionally area of Wall ( $dA_1$ ) projected on ~~center pixel~~ Camera ( $dA_2$ ) changes,

Q.3.

Consider a cross section  
of torus: with radius( $r$ )

we know that for  
Lambertian

$$L_r = \kappa \hat{N} \cdot \vec{S}$$

$$L_r = \kappa |\hat{N}| |\vec{S}| \cos \theta$$

$$\text{Hence, } |\vec{S}| = 1$$

since light source is  $(0, 0, 1)$

$$\therefore L_r = \rho \cos \theta \quad [\rho \rightarrow \text{albedo}; |\hat{N}| |\vec{S}| = 1]$$

$\theta$ : angle b/w Normal & light source direction.

For right angle triangle in fig. 11

$$\underline{r \sin \theta = p}$$

$$\therefore \underline{r \sin \theta = 4 - \sqrt{x^2 + y^2}} \quad [\because r=1]$$

$$\underline{\sin \theta = 4 - \sqrt{x^2 + y^2}}$$

$$\cos \theta = \underline{\sqrt{1 - \sin^2 \theta}} = \underline{\sqrt{1 - (4 - \sqrt{x^2 + y^2})^2}}$$

$$\underline{L = \rho \sqrt{1 - (4 - \sqrt{x^2 + y^2})^2}} = \underline{L(x, y)}$$

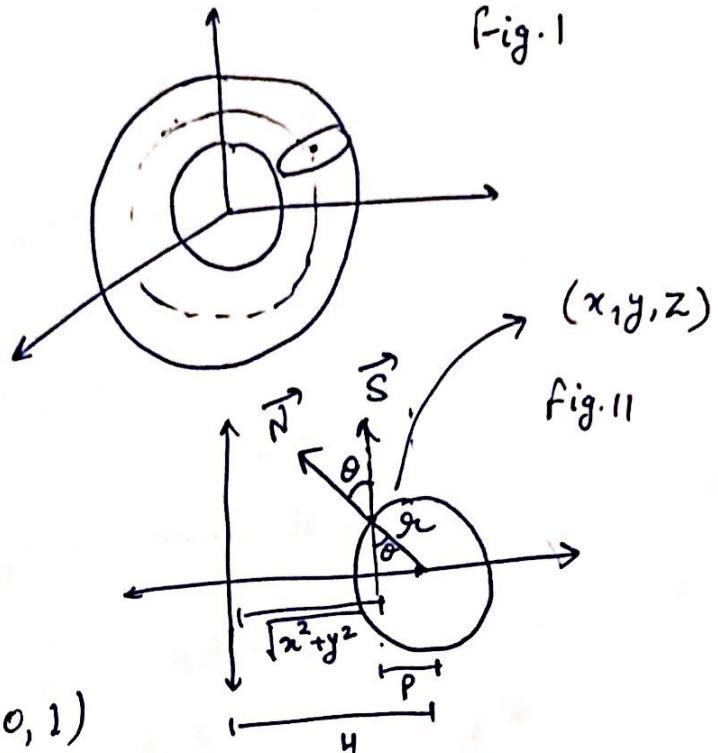
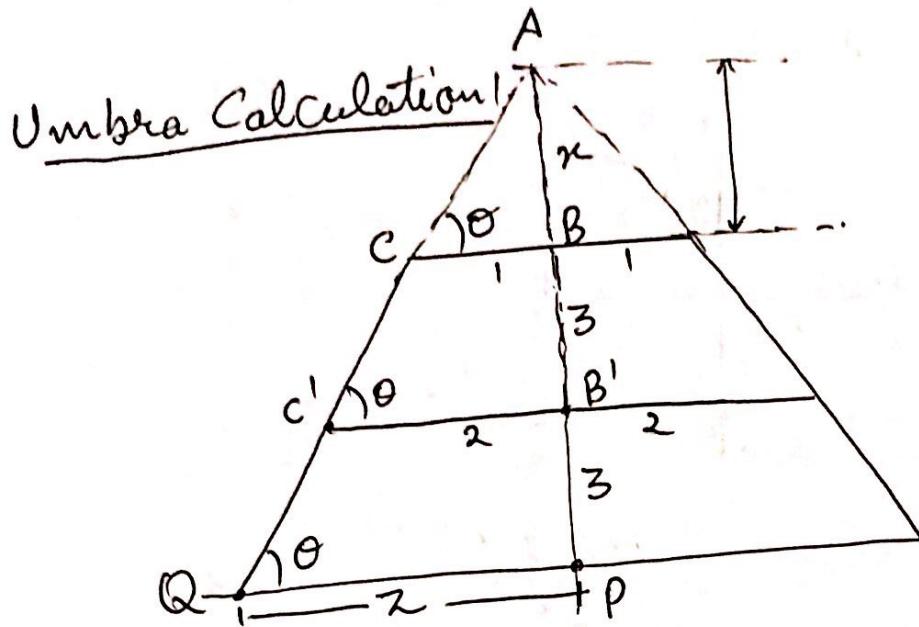


Fig. 1

Fig. 11

①

Q.4.



$$\tan \theta = \frac{6+x}{z} = \frac{3+n}{2} = \frac{n}{1} \quad (\Delta ABC)$$

(Using Right angle  
triangles)

$$\therefore \frac{3+x}{2} = n$$

$$n = 3$$

$$\text{and, } \frac{6+x}{z} = n \Rightarrow \frac{9}{3} = z$$

$$\therefore \boxed{z = 3}$$

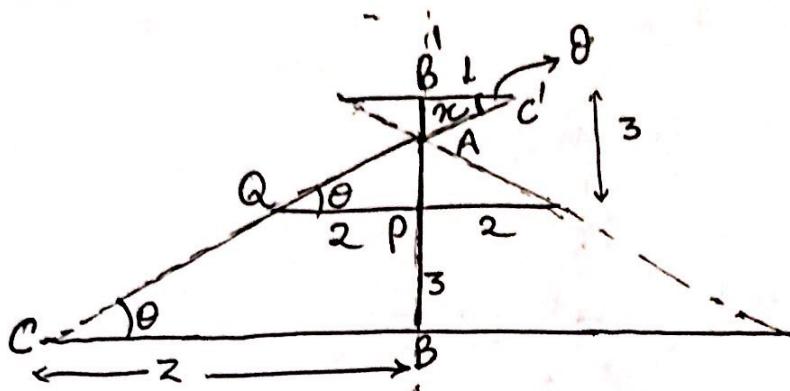
Hence total side length of umbra region = 6 units

$$\therefore \text{Area of Square} = 6 \times 6$$

(Umbra) = 36 sq. units

## Penumbra Calculation!

(2)



from eight angle triangles.

$$\therefore \frac{6-x}{z} = \tan \theta = \frac{3-x}{2} = \frac{x}{1}$$

$$(\Delta ABC) \quad (\Delta APQ) \quad (\Delta AB'C')$$

$$\therefore \frac{3-x}{2} = x \Rightarrow 2x + x = 3 \Rightarrow \boxed{x=1}$$

$$\therefore \frac{6-x}{z} = x \Rightarrow \frac{(6-1)}{(1)} = z$$

$$\therefore \boxed{z=5}$$

$\therefore$  Total size of square region that contains both Umbra & Penumbra

$$\Rightarrow \boxed{10 \text{ units}}$$

$$\therefore \text{Area} = 10 \times 10 = 100 \text{ sq. units}$$

Hence, area of Penumbra =  $\frac{100 - 36}{\text{area of Umbra only}} \Rightarrow$

$$= \boxed{64 \text{ sq. units}}$$