Photometric Stereo, Specularity Removal and Surface Rendering

0.0.1 Yifan Xu

0.0.2 November 7, 2018

0.1 Photometric Stereo, Specularity Removal

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 specularties are removed, but the albedo is unknown and non-constant in the images.

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

0.1.1 Data

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

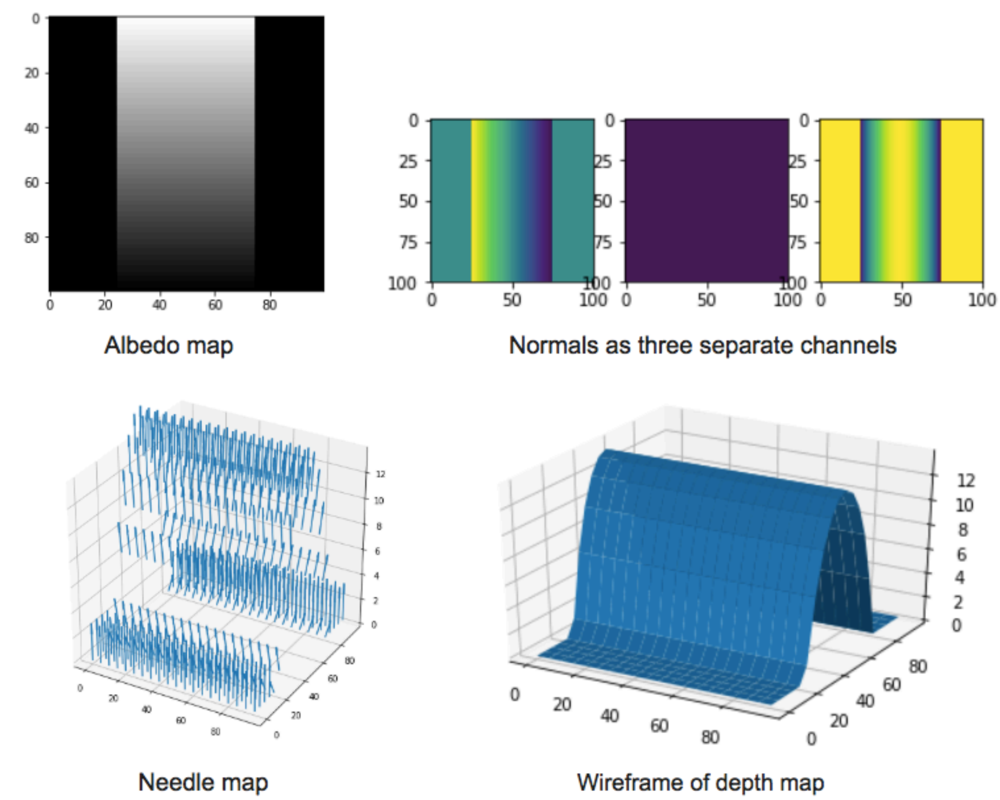
- `im1`, `im2`, `im3`, `im4`... images.
- `l1`, `l2`, `l3`, `l4`... light source directions.
- `c` (when required) color of light source.

0.1.2 Part 1:

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.



example.bb

Problem5 example

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:

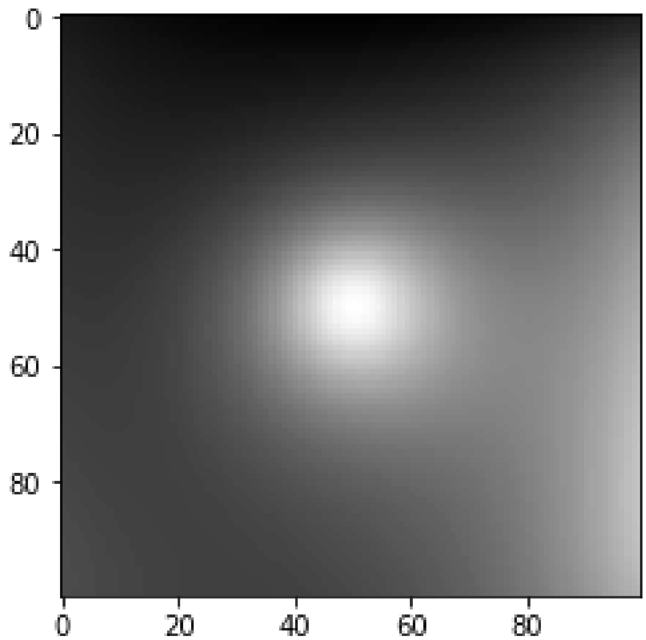
1. The estimated albedo map.
2. The estimated surface normals by showing both
    1. Needle map, and
    2. Three images showing components of surface normal.
3. A wireframe of depth map.

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 [1]:
```
## 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', '_
Image:
```



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

```
In [1]: 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 [10]: def photometric_stereo(images, lights, mask):
             '''
             your implementaion
             '''
             albedo = np.ones(images[0].shape)
             normals = np.dstack((np.zeros(images[0].shape),
                                  np.zeros(images[0].shape),
                                  np.ones(images[0].shape)))
```

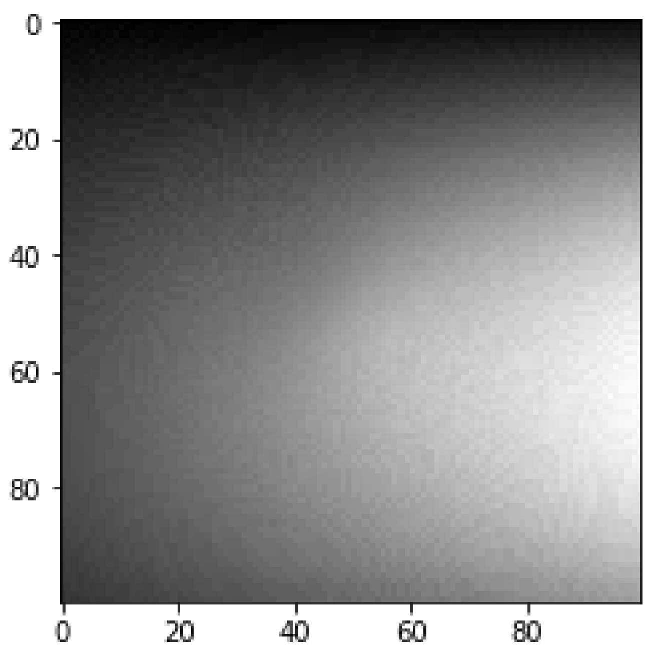
```
    H = np.ones(images[0].shape)
    H_horn = np.ones(images[0].shape)
    p = np.ones(images[0].shape)
    q = np.ones(images[0].shape)


    for i in range(images[0].shape[0]):
        for j in range(images[0].shape[1]):
            b = np.dot(np.dot(linalg.inv(np.dot(lights.T,lights)),lights.T),images[:,i
            albedo[i,j] = linalg.norm(b)
            normals[i,j,:] = b/albedo[i,j]
            p[i,j] = normals[i,j][1]/normals[i,j][2]
            q[i,j] = normals[i,j][0]/normals[i,j,][2]

    if np.any(mask) == 0:
        p[mask] = 0
        q[mask] = 0

    for i in range(1,images[0].shape[0]):
        H[i,0] = H[i-1,0] + p[i,0]
    for i in range(1,images[0].shape[0]):
        for j in range(1,images[0].shape[1]):
            H[i,j] = H[i,j-1] + q[i,j]

    H_horn = horn_integrate(normals[:,:,0], normals[:,:,1], mask, 1000)
#     print(normals)
    return albedo, normals, H, H_horn
```

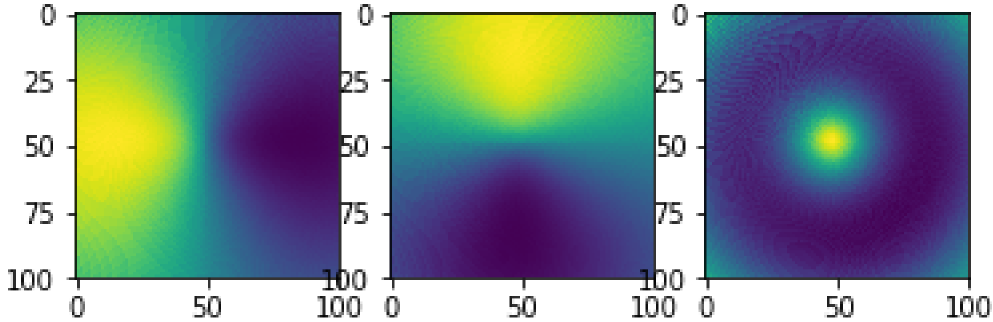
In [275]:
```
from mpl_toolkits.mplot3d import Axes3D

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"]))

# Use im1, im2 and im4
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)
```

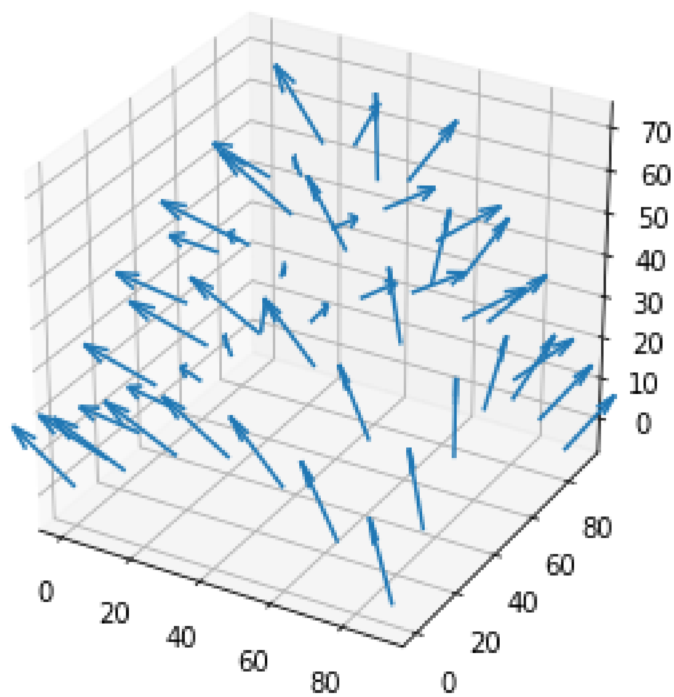
```python
print(albedo)
# ----------------------------------------------------------------------------
# Following code is just a working example so you don't get stuck with any
# of the graphs required. You may want to write your own code to align the
# results in a better layout.
# ----------------------------------------------------------------------------

# Stride in the plot, you may want to adjust it to different images
stride = 15

# 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 = 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=(5, 5))
ax = fig.gca(projection='3d')
plt.quiver(X,Y,Z,NX,NY,NZ, length=20.)
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()
```

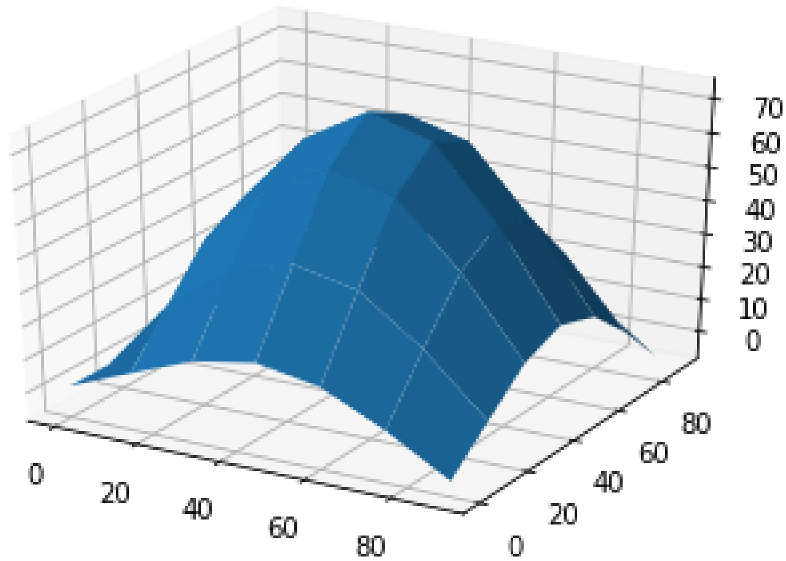
```python
H = horn[::stride,::stride]
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(X,Y, H.T)
plt.show()
```

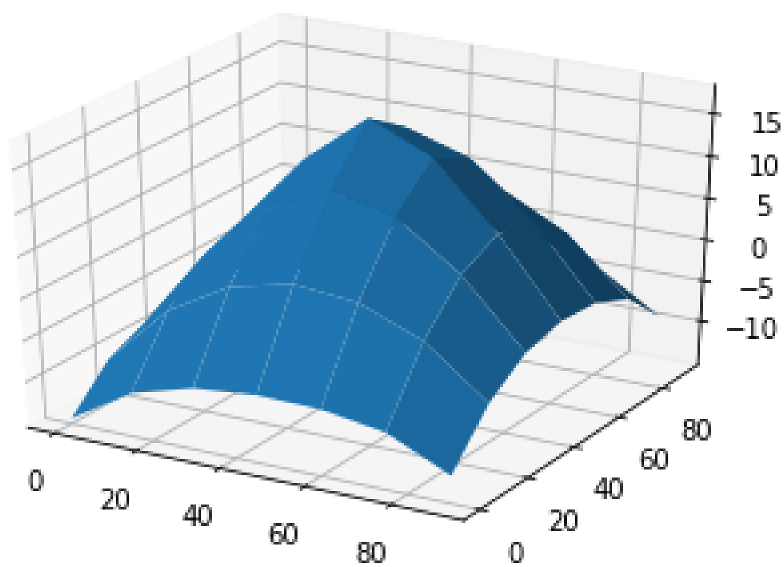
```
/Users/yifanxu/anaconda3/lib/python3.7/site-packages/mkl_fft/_numpy_fft.py:1044: FutureWarning
  output = mkl_fft.rfftn_numpy(a, s, axes)


[[ 48.98979486  47.16990566  48.17675788 ...  55.35341001  55.35341001
   54.53439282]
 [ 49.80963762  48.98979486  48.98979486 ...  56.88585061  58.83026432
   57.66281297]
 [ 49.80963762  49.80963762  53.38539126 ...  60.34069937  61.91930232
   61.10646447]
 ...
 [ 83.79140767  83.79140767  86.57944329 ... 153.52849898 154.98064395
  158.90248582]
 [ 81.21576202  83.79140767  83.79140767 ... 152.8070679  154.25303887
  155.71127127]
 [ 81.21576202  83.79140767  80.43009387 ... 147.2446943  151.14893318
  154.98064395]]
```




0
25
50
75
100
0
50
100


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



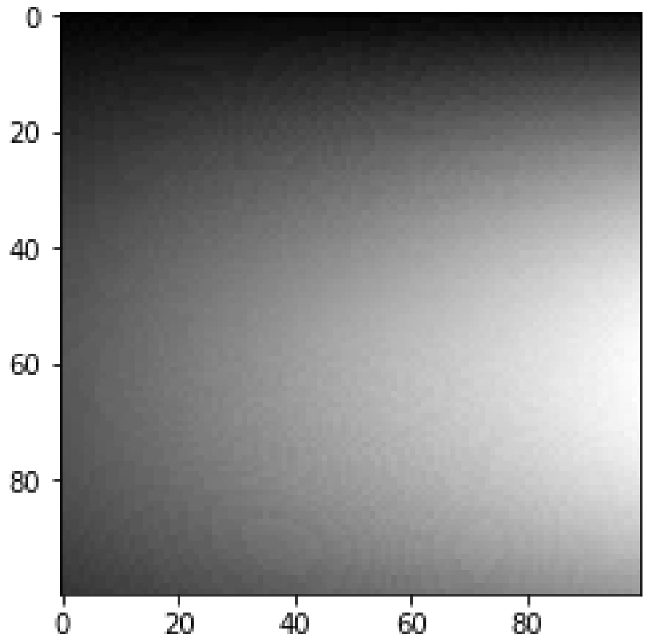


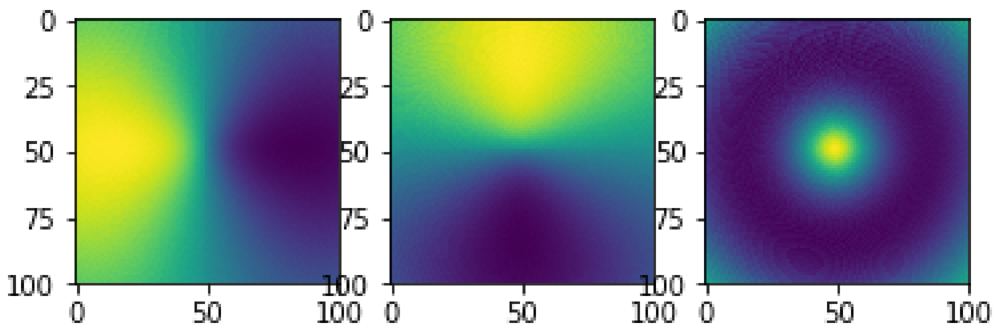
In [278]:
```
from mpl_toolkits.mplot3d import Axes3D

pickle_in = open("synthetic_data.pickle", "rb")
data = pickle.load(pickle_in, encoding="latin1")
lights = np.vstack((data["l1"], data["l2"], data["l3"], data["l4"]))
```

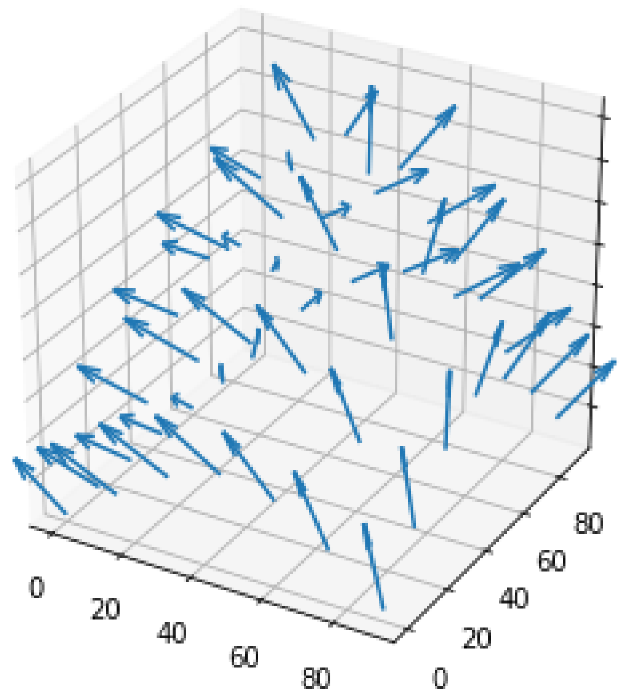
```python
images = []
images.append(data["im1"])
images.append(data["im2"])
images.append(data["im3"])
images.append(data["im4"])
images = np.array(images)

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

albedo, normals, depth, horn = photometric_stereo(images, lights, mask)
print(albedo)
# --------------------------------------------------------------------------
# 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 = 15

# 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 = depth[::stride,::stride].T
NX = normals[..., 0][::stride,::-stride].T
NY = normals[..., 1][::-stride,::stride].T
```

```python
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=20.)
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()
```

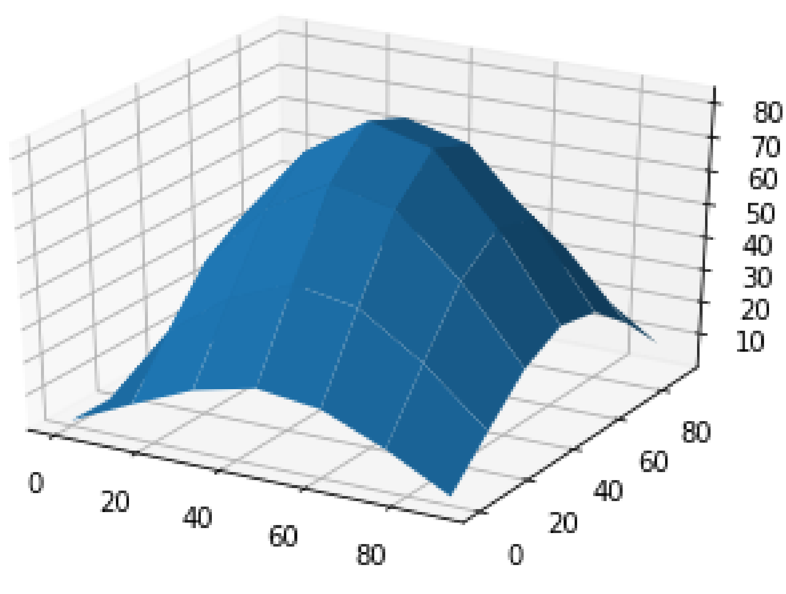
```
/Users/yifanxu/anaconda3/lib/python3.7/site-packages/mkl_fft/_numpy_fft.py:1044: FutureWarning
  output = mkl_fft.rfftn_numpy(a, s, axes)


[[ 50.4889977   51.5         49.70719823 ...  53.64595874  53.64595874
   51.67446178]
 [ 52.87931753  50.4889977   52.12058668 ...  55.20165054  58.08733846
   55.99007849]
 [ 52.87931753  52.87931753  55.14349967 ...  59.60541549  59.92680721
   60.37498562]
 ...
 [ 83.51796214  83.51796214  86.02470575 ... 144.64170292 146.16391103
  147.80101112]
 [ 81.12062349  83.51796214  83.51796214 ... 143.88498493 145.40136023
  144.80839831]
 [ 81.12062349  83.51796214  80.34232315 ... 138.57569131 142.46481047
  144.03404304]]
```


0
20
40
60
80
0
20
40
60
80

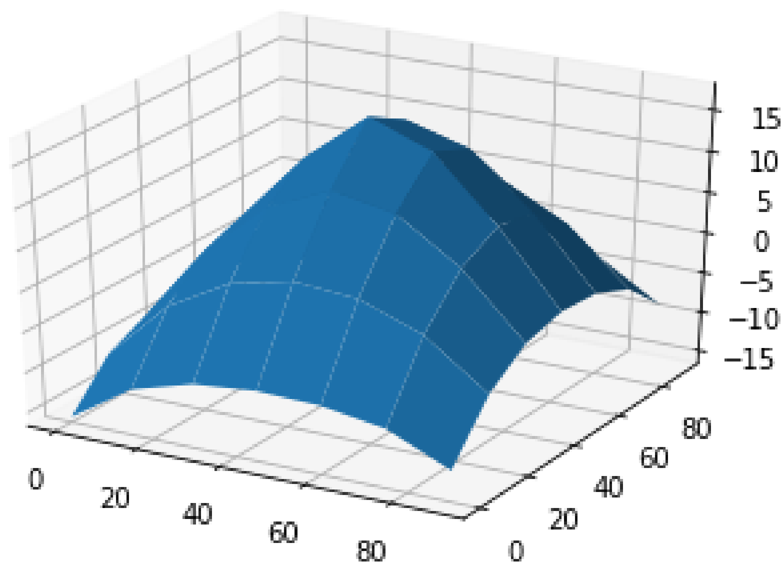

0
25
50
75
100
0
50
100



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



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



0.1.3 Part 2:

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`.

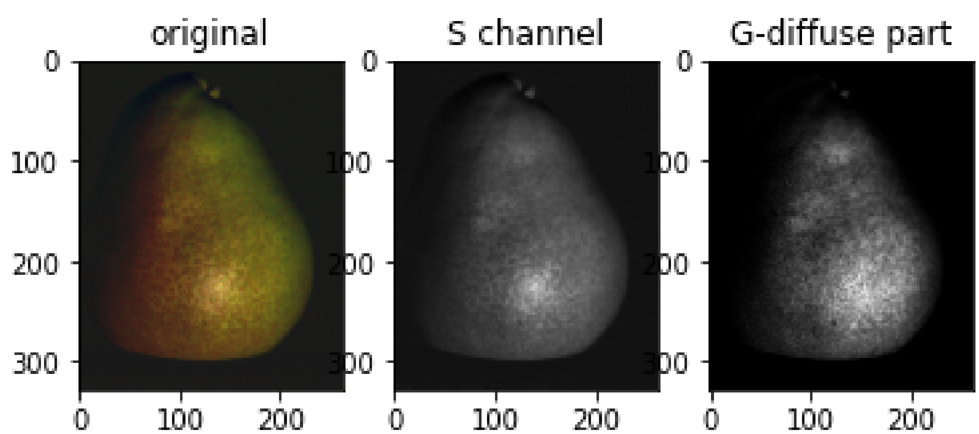
```
In [12]: def get_rot_mat(rot_v, unit=None):
             '''
             Takes a vector and returns the rotation matrix required to align the
             unit vector(2nd arg) to it.
             '''
             if unit is None:
                 unit = [1.0, 0.0, 0.0]

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

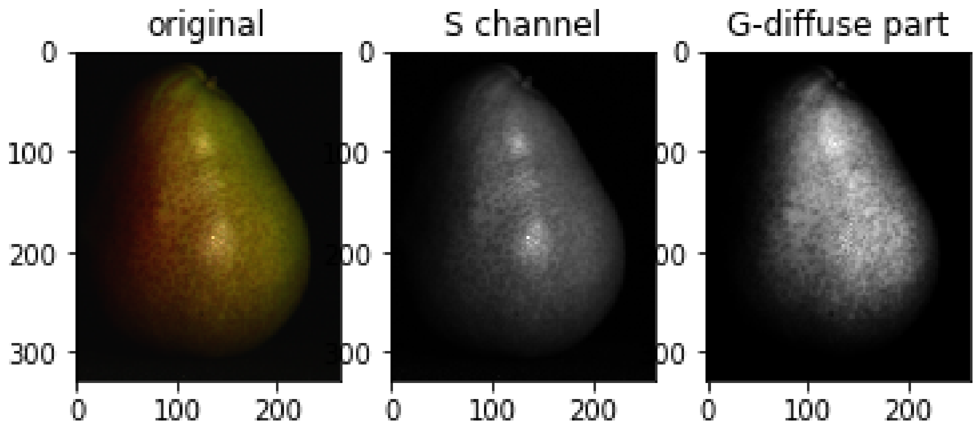
```
        rcos = np.dot(rot_v, unit) #cos by dot product
        rsin = np.linalg.norm(uvw) #sin by magnitude of cross product

        #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
        '''
        S = np.ones(I_rgb.shape[:2])
        G = np.ones(I_rgb.shape[:2])
        R = get_rot_mat(rot_vec)
        for i in range(I_rgb.shape[0]):
            for j in range(I_rgb.shape[1]):
                I_suv = np.dot(R, I_rgb[i,j,:])
                S[i,j] = I_suv[0]
                G[i,j] = linalg.norm(I_suv[1]*I_suv[1] + I_suv[2]*I_suv[2])
        return S, G
```

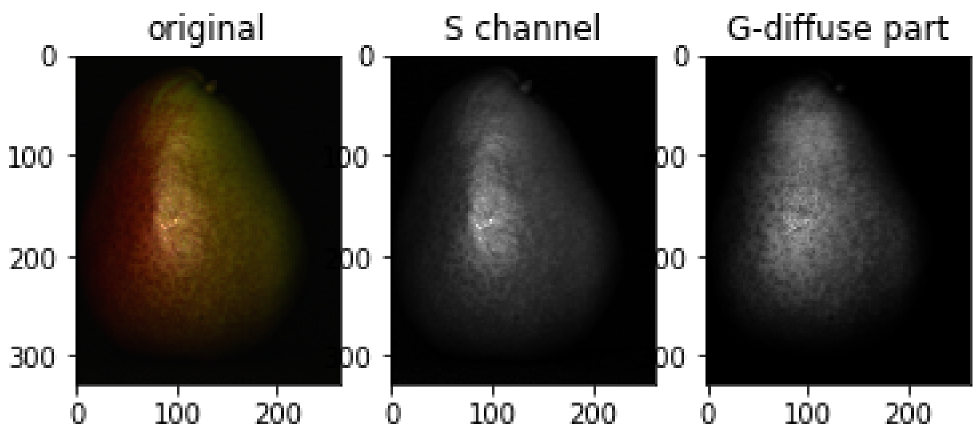
In [13]:
```
pickle_in = open("specular_sphere.pickle", "rb")
# data = pickle.load(pickle_in)
data = pickle.load(pickle_in, encoding="latin1")

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

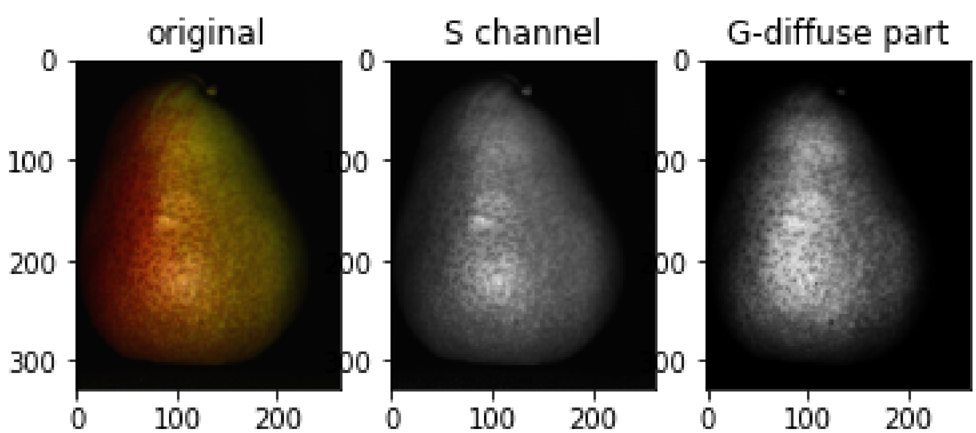
In [15]:
```
pickle_in = open("specular_pear.pickle", "rb")
# data = pickle.load(pickle_in)
```

```python
data = pickle.load(pickle_in, encoding="latin1")

def rgb255(image):
    img = (image - np.min(image))/(np.max(image) - np.min(image))
    return img
for i in range(images.shape[0]):
    images = []
    images.append(data["im1"])
    images.append(data["im2"])
    images.append(data["im3"])
    images.append(data["im4"])
    images = np.array(images)

    imgorg = rgb255(images[i])
    plt.subplot(1,3,1)
    plt.imshow(imgorg,cmap="gray")
    plt.title("original")
    S,G = RGBToSUV(images[i], np.hstack((data["c"][0][0],
                                         data["c"][1][0],
                                         data["c"][2][0])))
    plt.subplot(1,3,2)
    plt.imshow(S,cmap="gray")
    plt.title("S channel")
    plt.subplot(1,3,3)
    plt.title("G-diffuse part")
    plt.imshow(G,cmap="gray")
    plt.show()
```




original
S channel
G-diffuse part
0
100
200
300
0
100
200


original
S channel
G-diffuse part
0
100
200
300
0
100
200


original
S channel
G-diffuse part
0
100
200
300
0
100
200

0.1.4 Part 3:

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:

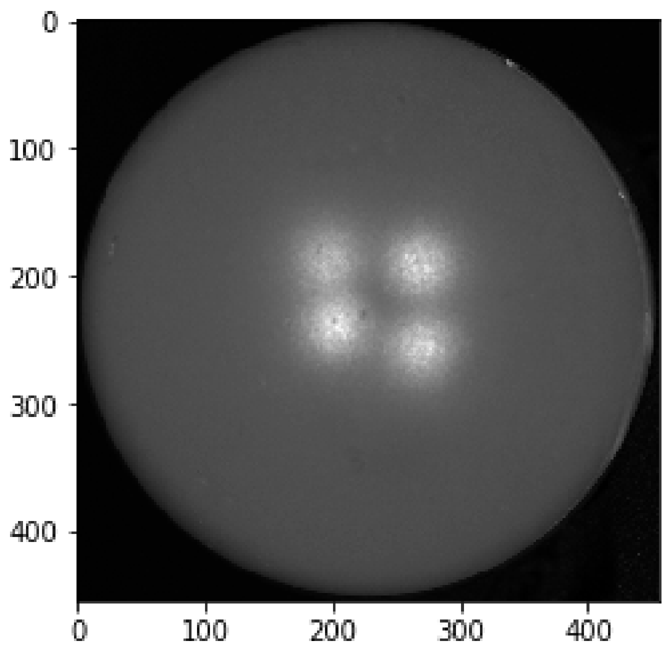
1. The estimated albedo map (original and diffuse)
2. The estimated surface normals (original and diffuse) by showing both
    1. Needle map, and
    2. Three images showing components of surface normal
3. A wireframe of depth map (original and diffuse)

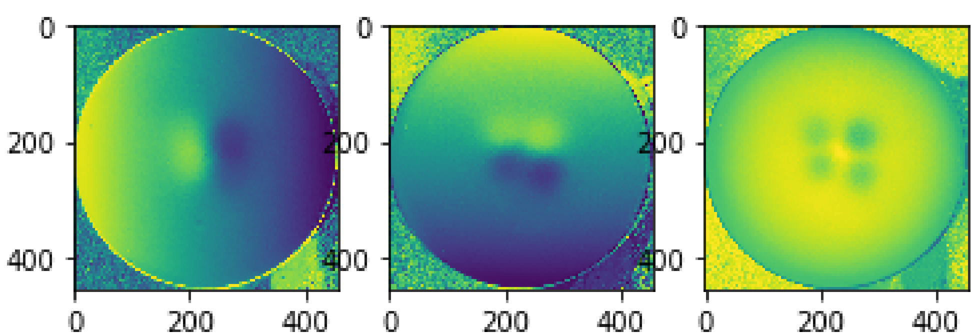
In [282]:
```
# ----------------------------------------------------------------------------
# You may reuse the code for photometric_stereo here.
# Write your code below to process the data and send it to photometric_stereo
# and display the albedo, normals and depth maps.
# ----------------------------------------------------------------------------
from mpl_toolkits.mplot3d import Axes3D
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])

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

images = []
images.append(rgb2gray(data["im1"]))
images.append(rgb2gray(data["im2"]))
images.append(rgb2gray(data["im3"]))
images.append(rgb2gray(data["im4"]))
images = np.array(images)


for img_idx in range(1,5):
    S_sphere, G_sphere = RGBToSUV(data["im" + str(img_idx)], np.hstack((data["c"][0]
                                    data["c"][1][0],
                                    data["c"][2][0])))


mask_sphere = np.ones(data["im1"].shape)[:,:,0]
```

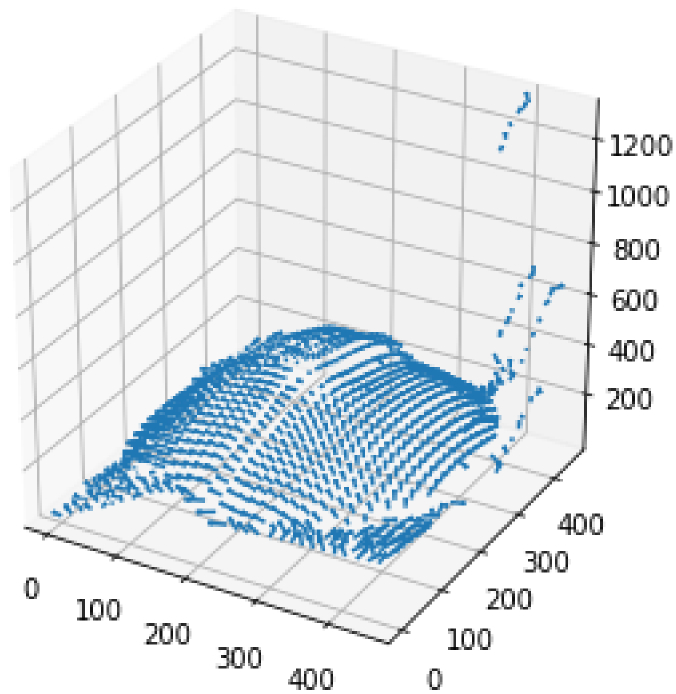
```python
albedo, normals, depth, horn = photometric_stereo(images, lights, mask_sphere)
normals_sphere_gray = normals


# --------------------------------------------------------------------------
# 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 = 15

# 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 = 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=(5, 5))
ax = fig.gca(projection='3d')
plt.quiver(X,Y,Z,NX,NY,NZ, length=20.)
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()
```

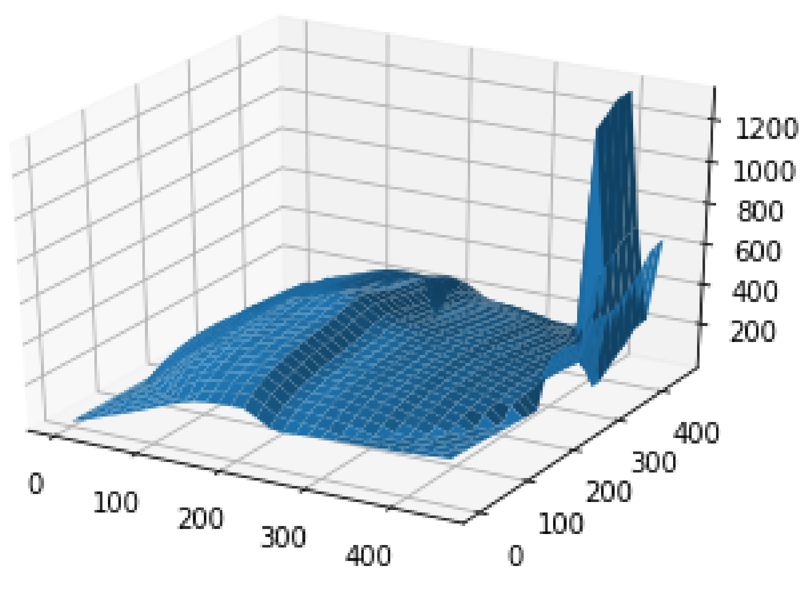
```
/Users/yifanxu/anaconda3/lib/python3.7/site-packages/mkl_fft/_numpy_fft.py:1044: FutureWarning
  output = mkl_fft.rfftn_numpy(a, s, axes)
```



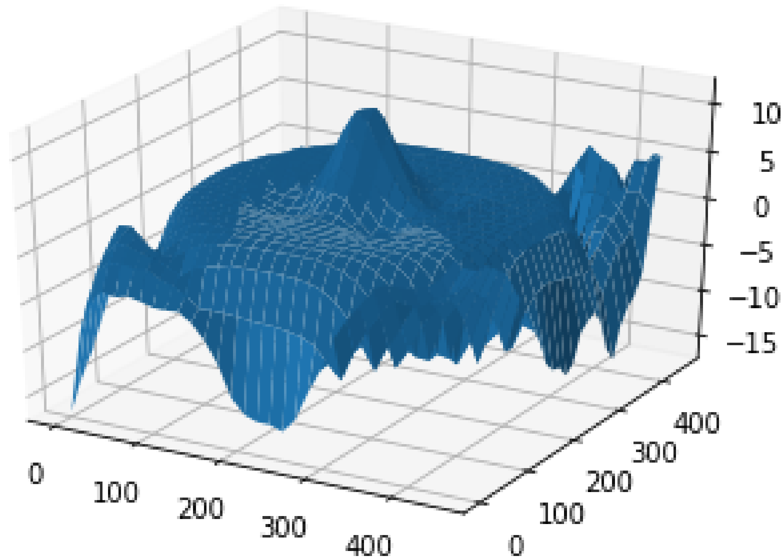




1200
1000
800
600
400
200
0
100
200
300
400
0
100
200
300
400



1200
1000
800
600
400
200
0
100
200
300
400
0
100
200
300
400

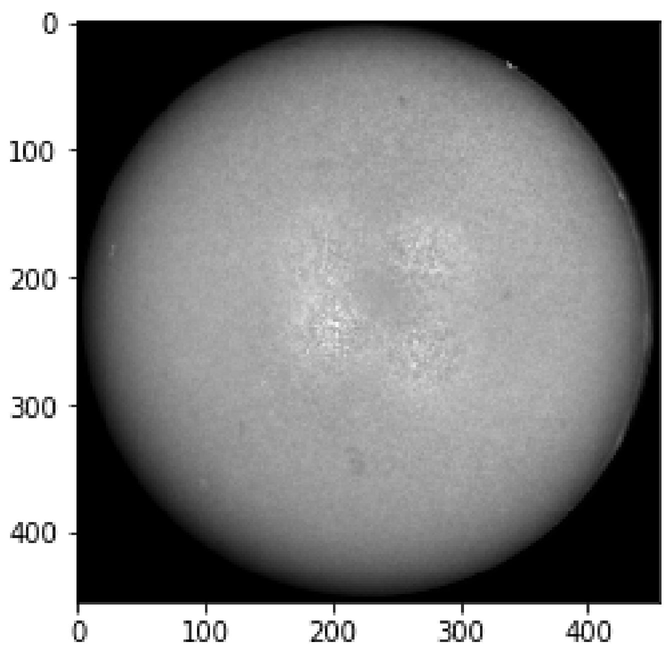


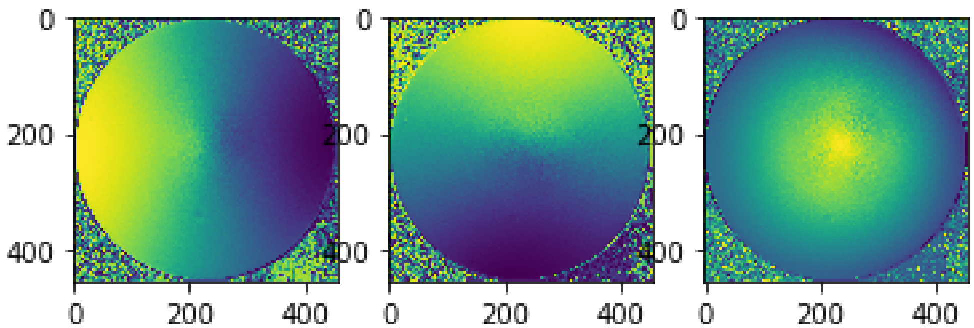
In [63]:
```python
from mpl_toolkits.mplot3d import Axes3D
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])


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["l4"]))
lights = np.vstack((data["l1"], data["l2"], data["l3"], data["l4"]))
S1,G1 = RGBToSUV(data["im1"], np.hstack((data["c"][0][0],
                                         data["c"][1][0],
                                         data["c"][2][0])))
S2,G2 = RGBToSUV(data["im2"], np.hstack((data["c"][0][0],
                                         data["c"][1][0],
                                         data["c"][2][0])))
S3,G3 = RGBToSUV(data["im3"], np.hstack((data["c"][0][0],
                                         data["c"][1][0],
                                         data["c"][2][0])))
S4,G4 = RGBToSUV(data["im4"], np.hstack((data["c"][0][0],
                                         data["c"][1][0],
                                         data["c"][2][0])))


images = []
images.append(G1)
```

```python
images.append(G2)
images.append(G3)
images.append(G4)
images = np.array(images)

mask_sphere = np.ones(data["im1"].shape)[:,:,0]

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


# --------------------------------------------------------------------------
# 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 = 15

# 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 = 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=(5, 5))
```

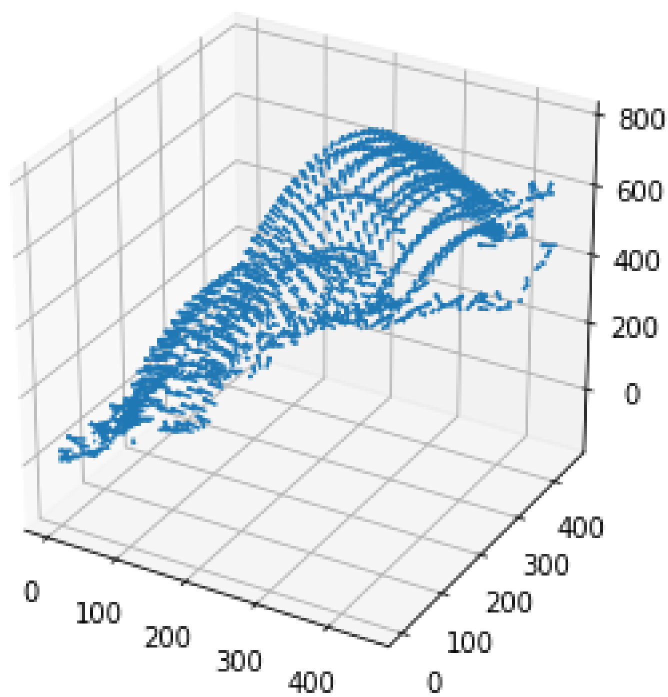
```
ax = fig.gca(projection='3d')
plt.quiver(X,Y,Z,NX,NY,NZ, length=20.)
plt.show()

# 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()
```

```
/Users/yifanxu/anaconda3/lib/python3.7/site-packages/mkl_fft/_numpy_fft.py:1044: FutureWarning
  output = mkl_fft.rfftn_numpy(a, s, axes)
```

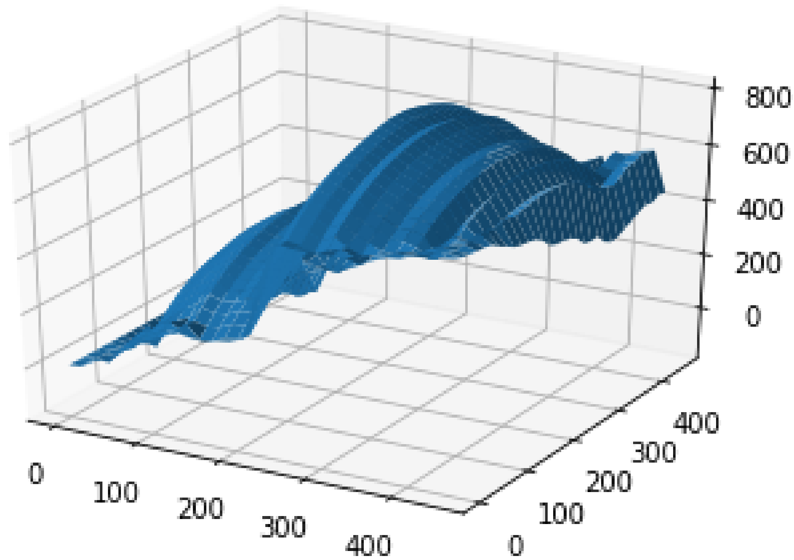


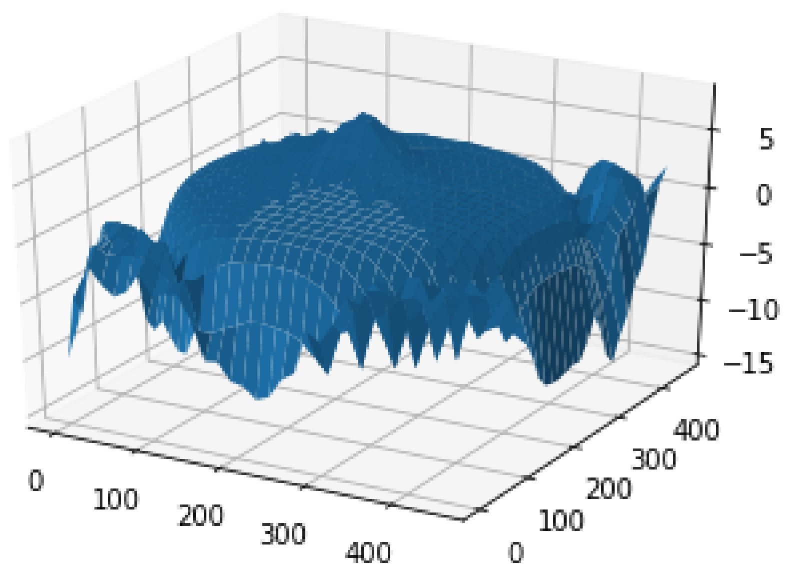


0
200
400
0
200
400



800
600
400
200
0
0
100
200
300
400
400
300
200
100
0



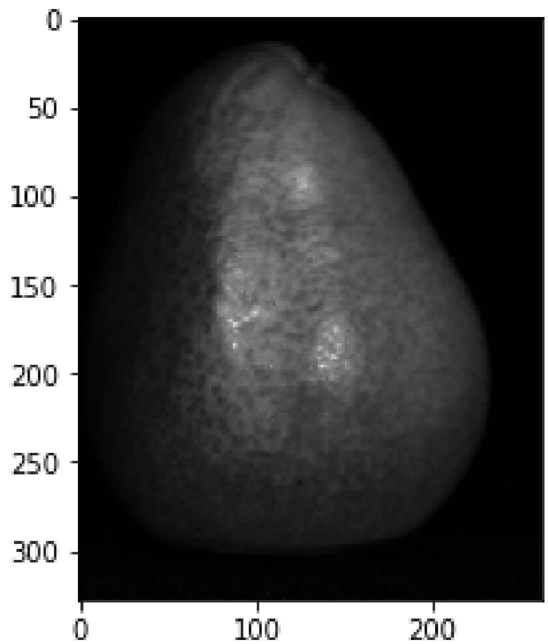


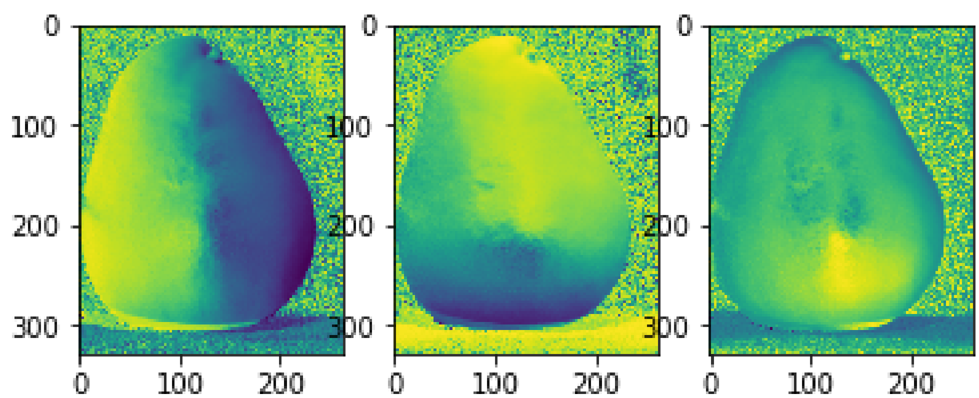
In [18]:
```
# ------------------------------------------------------------------------------
# You may reuse the code for photometric_stereo here.
# Write your code below to process the data and send it to photometric_stereo
# and display the albedo, normals and depth maps.
# ------------------------------------------------------------------------------
```

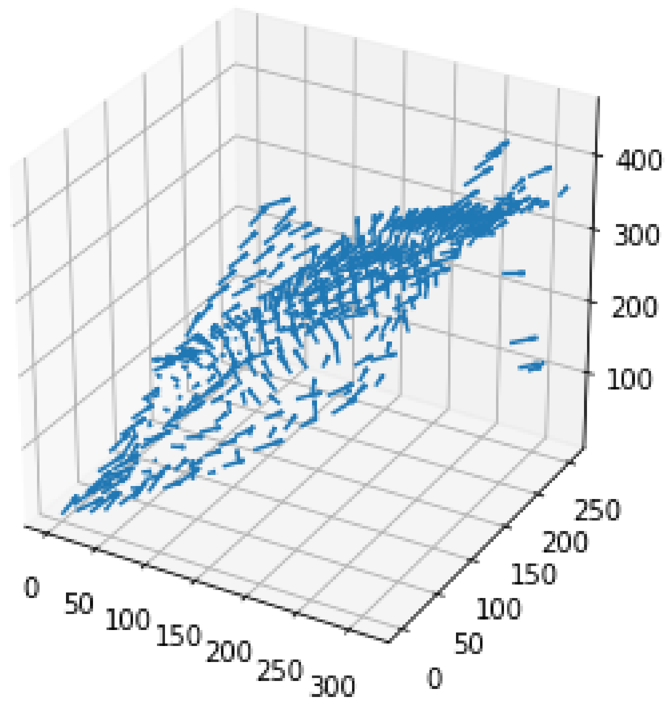
```python
from mpl_toolkits.mplot3d import Axes3D
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])


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

images = []
images.append(rgb2gray(data["im1"]))
images.append(rgb2gray(data["im2"]))
images.append(rgb2gray(data["im3"]))
images.append(rgb2gray(data["im4"]))
images = np.array(images)

mask_pear = np.ones(data["im1"].shape)[:,:,0]

albedo, normals, depth, horn = photometric_stereo(images, lights, mask_pear)
normals_pear_gray = normals
# ---------------------------------------------------------------------------
# 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 = 15

# 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()
```

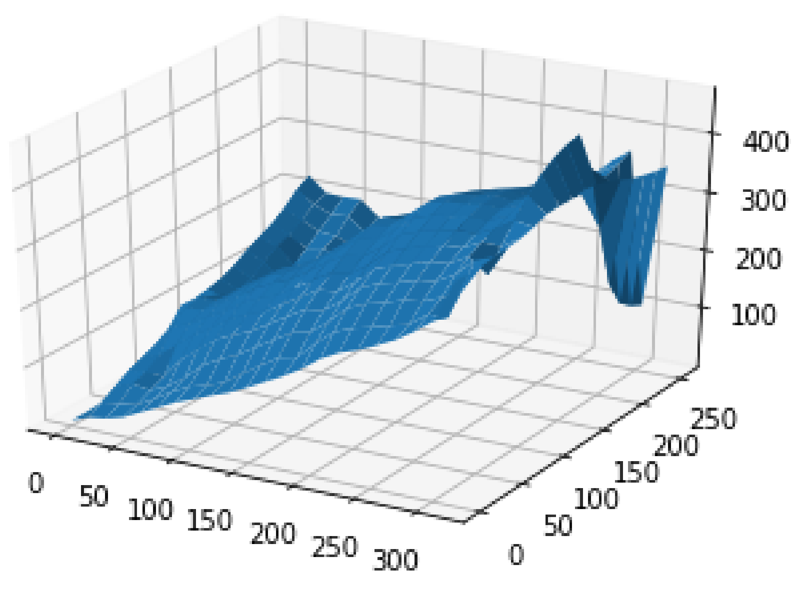
```
# showing normals as quiver
X, Y, _ = np.meshgrid(np.arange(0,np.shape(normals)[0], 15),
                      np.arange(0,np.shape(normals)[1], 15),
                      np.arange(1))
X = X[..., 0]
Y = Y[..., 0]
Z = depth[::stride,::stride].T
NX = normals[..., 0][::stride,::-stride].T
NY = normals[..., 1][::-stride,::stride].T
NZ = normals[..., 2][::stride,::stride].T
fig = plt.figure(figsize=(5, 5))
ax = fig.gca(projection='3d')
plt.quiver(X,Y,Z,NX,NY,NZ, length=20.)
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()
```

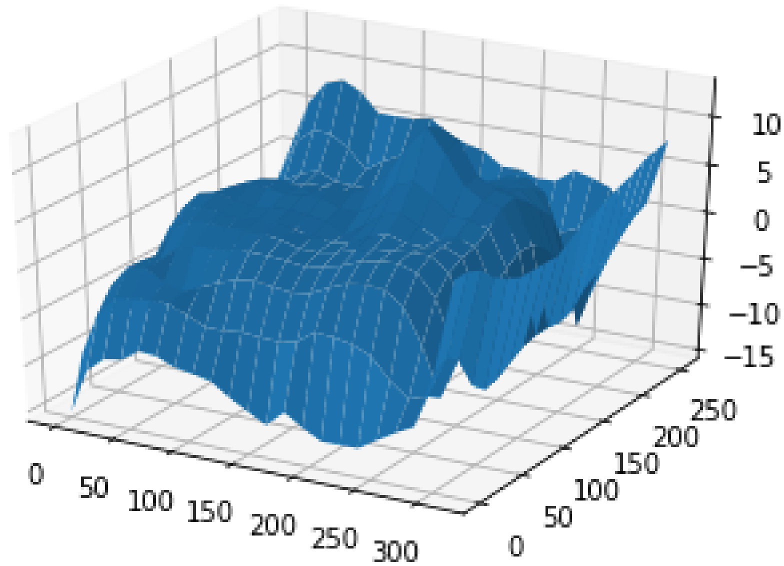
```
/Users/yifanxu/anaconda3/lib/python3.7/site-packages/mkl_fft/_numpy_fft.py:1044: FutureWarning
  output = mkl_fft.rfftn_numpy(a, s, axes)
```


0
50
100
150
200
250
300
0
100
200


0
100
200
300
0
100
200


400
300
200
100
0
50
100
150
200
250
300
0
50
100
150
200
250


400
300
200
100
0
50
100
150
200
250
300
0
50
100
150
200
250

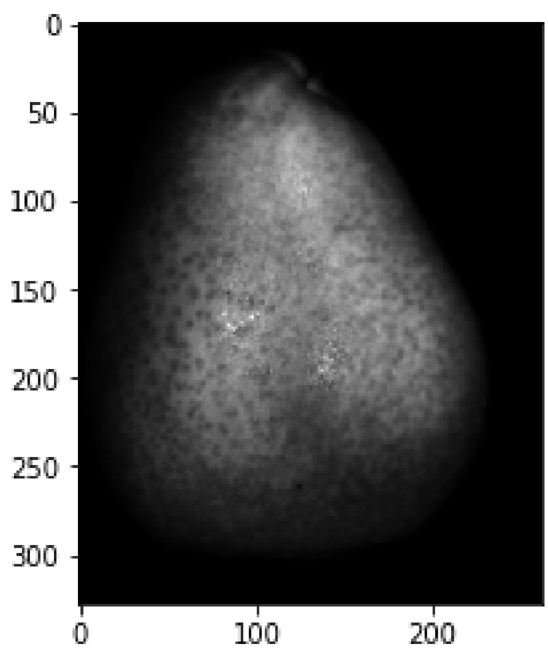


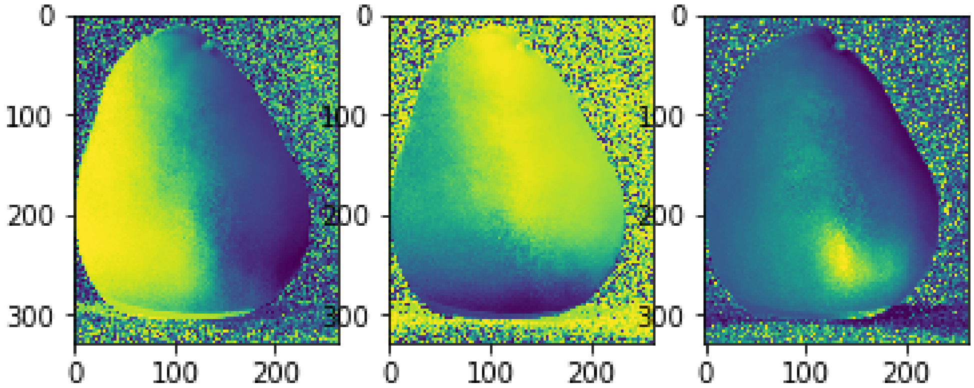
In [19]:
```
from mpl_toolkits.mplot3d import Axes3D
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])


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["l4"]))
lights = np.vstack((data["l1"], data["l2"], data["l3"], data["l4"]))
S1,G1 = RGBToSUV(data["im1"], np.hstack((data["c"][0][0],
                                          data["c"][1][0],
                                          data["c"][2][0])))
S2,G2 = RGBToSUV(data["im2"], np.hstack((data["c"][0][0],
                                          data["c"][1][0],
                                          data["c"][2][0])))
S3,G3 = RGBToSUV(data["im3"], np.hstack((data["c"][0][0],
                                          data["c"][1][0],
                                          data["c"][2][0])))
S4,G4 = RGBToSUV(data["im4"], np.hstack((data["c"][0][0],
                                          data["c"][1][0],
                                          data["c"][2][0])))


images = []
images.append(G1)
```

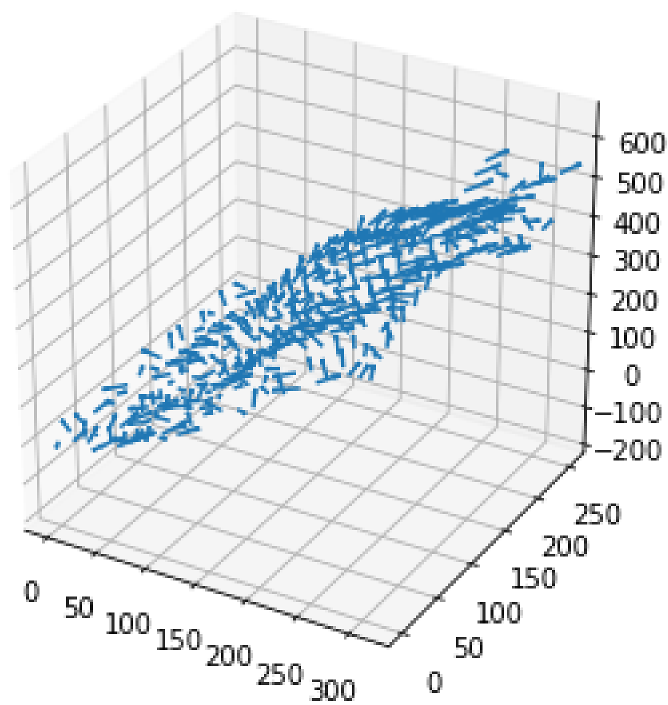
```python
images.append(G2)
images.append(G3)
images.append(G4)
images = np.array(images)

mask_pear = np.ones(data["im1"].shape)[:,:,0]

albedo, normals, depth, horn = photometric_stereo(images, lights, mask_pear)
normals_pear_SG = normals
# --------------------------------------------------------------------------
# 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 = 15

# 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 = 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=(5, 5))
ax = fig.gca(projection='3d')
plt.quiver(X,Y,Z,NX,NY,NZ, length=20.)
```

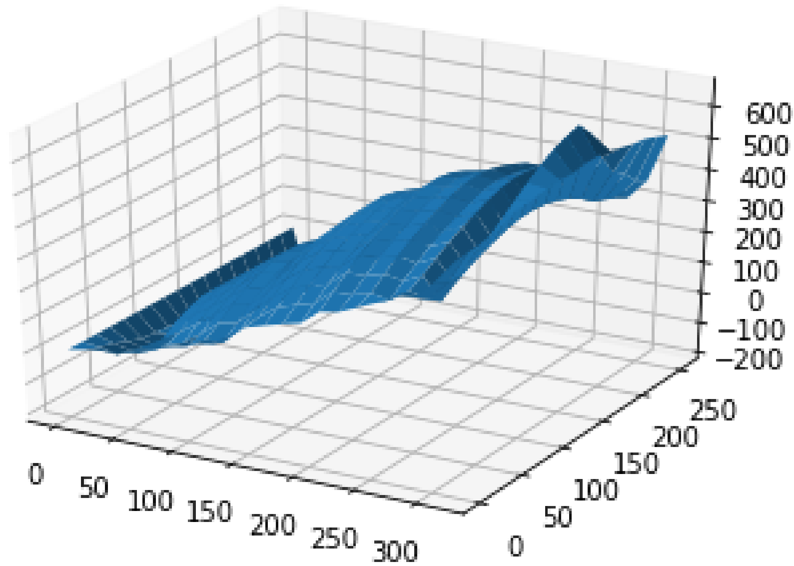
```
plt.show()

# 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()
```

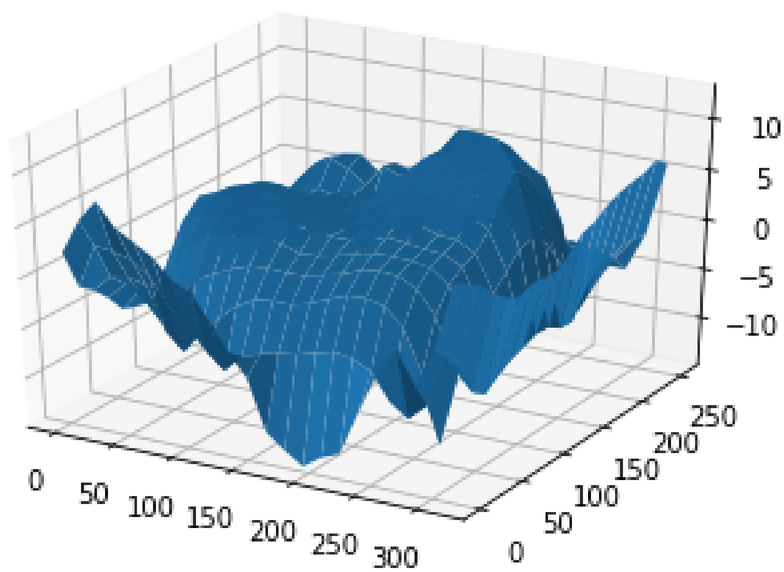
```
/Users/yifanxu/anaconda3/lib/python3.7/site-packages/mkl_fft/_numpy_fft.py:1044: FutureWarning
  output = mkl_fft.rfftn_numpy(a, s, axes)
```




0
100
200
300
0
100
200


600
500
400
300
200
100
0
−100
−200
0
50
100
150
200
250
300
0
50
100
150
200
250





0.2 Surface Rendering

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 photometeric stereo portion of this homework, 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.

0.2.1 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.

0.2.2 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.

0.2.3 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 α 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:

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

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

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 mth light source.

0.2.4 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, α), $\mathbf{V} = (0, 0, 1)^\top$, $\mathbf{N}$ 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

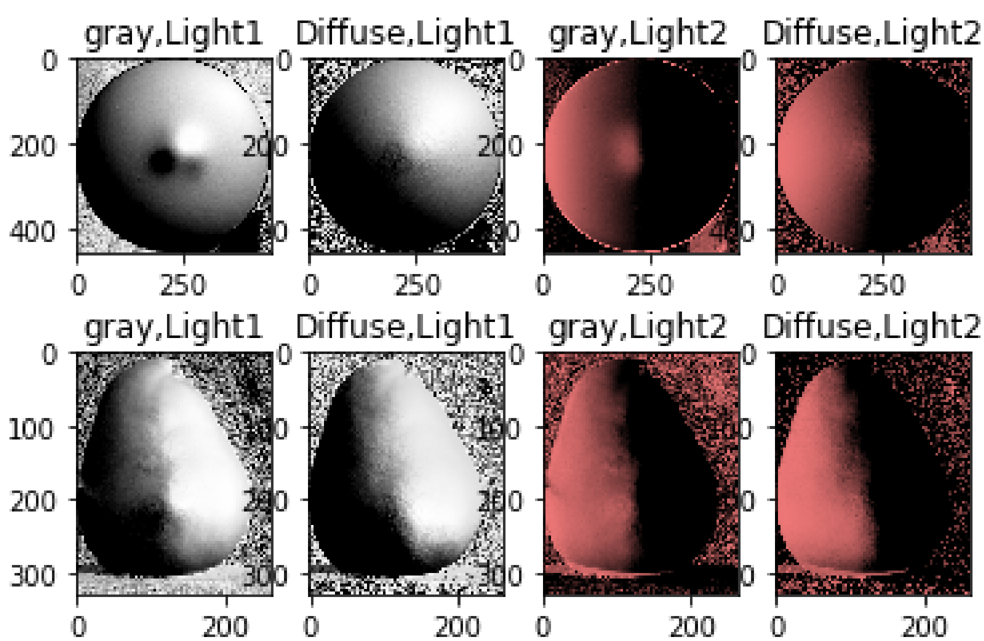
Mat.	k_a	k_d	k_s	α
1	0	0.1	0.75	5
2	0	0.5	0.1	5
3	0	0.5	0.5	10

0.2.5 Part 1. Lambertian model

In [283]:
```
def lambertian(normals, lights, color, intensity, mask):
    '''Your implementation'''
    image = np.zeros((normals.shape[0], normals.shape[1], 3))
    light = np.array(lights/linalg.norm(lights))
    color = np.array(color)
    for i in range(normals.shape[0]):
        for j in range(normals.shape[1]):
            n = linalg.norm(normals[i,j,:])
            n = np.array(normals[i,j,:]/n)
            image[i,j,:] = np.dot(light,n)*intensity*color
    image[mask==0] = 0
    image[image<0] = 0
    return image
```

In [284]:
```
# Output the rendering results
light1 = [-1/3, 1/3, 1/3]
```

```
light2 = [1, 0, 0]
color1 = [1, 1, 1]
color2 = [1, 0.5, 0.5]
form = np.array([[normals_sphere_gray,light1,color1,'gray,Light1'],
                 [normals_sphere_SG,light1,color1,'Diffuse,Light1'],
                 [normals_sphere_gray,light2,color2,'gray,Light2'],
                 [normals_sphere_SG,light2,color2,'Diffuse,Light2']])

for i in range(images.shape[0]):
    sphere = lambertian(form[i][0], form[i][1], form[i][2],
                        intensity=1.0, mask=mask_sphere)
    plt.subplot(2,4,i+1)
    plt.imshow(sphere,cmap="gray")
    plt.title(form[i][3])


form = np.array([[normals_pear_gray,light1,color1,'gray,Light1'],
                 [normals_pear_SG,light1,color1,'Diffuse,Light1'],
                 [normals_pear_gray,light2,color2,'gray,Light2'],
                 [normals_pear_SG,light2,color2,'Diffuse,Light2']])

for i in range(images.shape[0]):
    pear = lambertian(form[i][0], form[i][1], form[i][2],
                        intensity=1.0, mask=mask_pear)
    plt.subplot(2,4,i+5)
    plt.imshow(pear,cmap="gray")
    plt.title(form[i][3])
```



0.2.6 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 **L** and the normal vector on the surface of the object **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, **C** and I_l are the the color and intensity of the light source respectively.

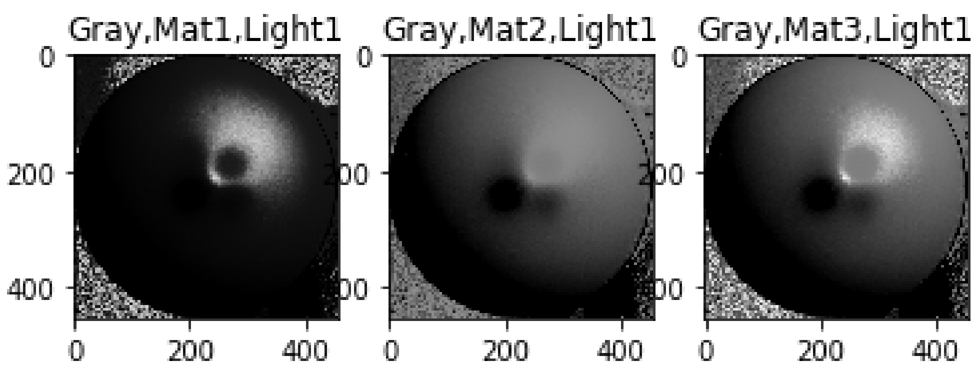
0.2.7 Part 2. Phong model

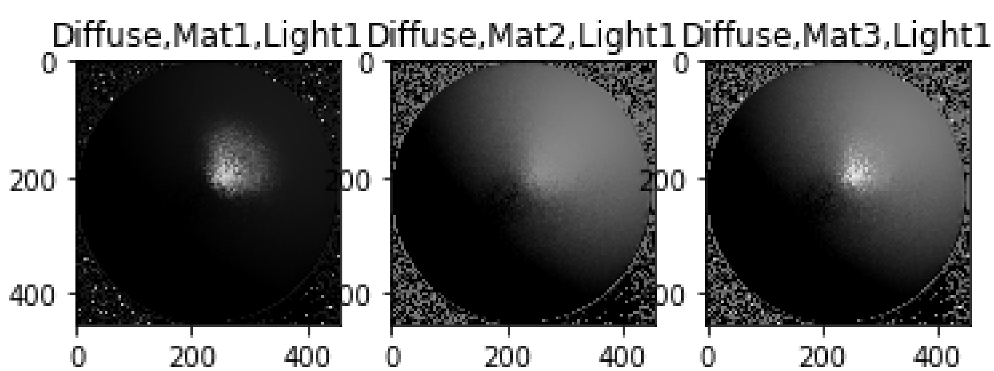
```
In [285]: def phong(normals, lights, color, material, view, mask):

              [ka,kd,ks,a] = material
              lights = np.array(lights)
              color = np.array(color)


              images = []
              for i in range(len(lights)):
                  image = np.zeros((normals.shape[0], normals.shape[1], 3))
                  images.append(image)
              h, w = normals.shape[0], normals.shape[1]

              for i in range(h):
                  for j in range(w):
                      n = linalg.norm(normals[i,j,:])
                      n = np.array(normals[i,j,:]/n).T

                      for k, l in enumerate(lights):
                          light = np.array([l/linalg.norm(l)])

                          first = kd*np.dot(light, n)*color[0]
                          first[first<0] = 0
                          Rm = 2*n*np.dot(light, n)-light
                          if np.dot(Rm, view) >= 0:
                              second = ks*(np.dot(Rm, view)**a)*color[0]
                          else:
                              second = 0
                          images[k][i, j, :] += (first+second)

              image_tot = np.zeros((normals.shape[0], normals.shape[1], 3))
              for img in images:
                  img[img<0] = 0
                  img[mask==0] = 0
```

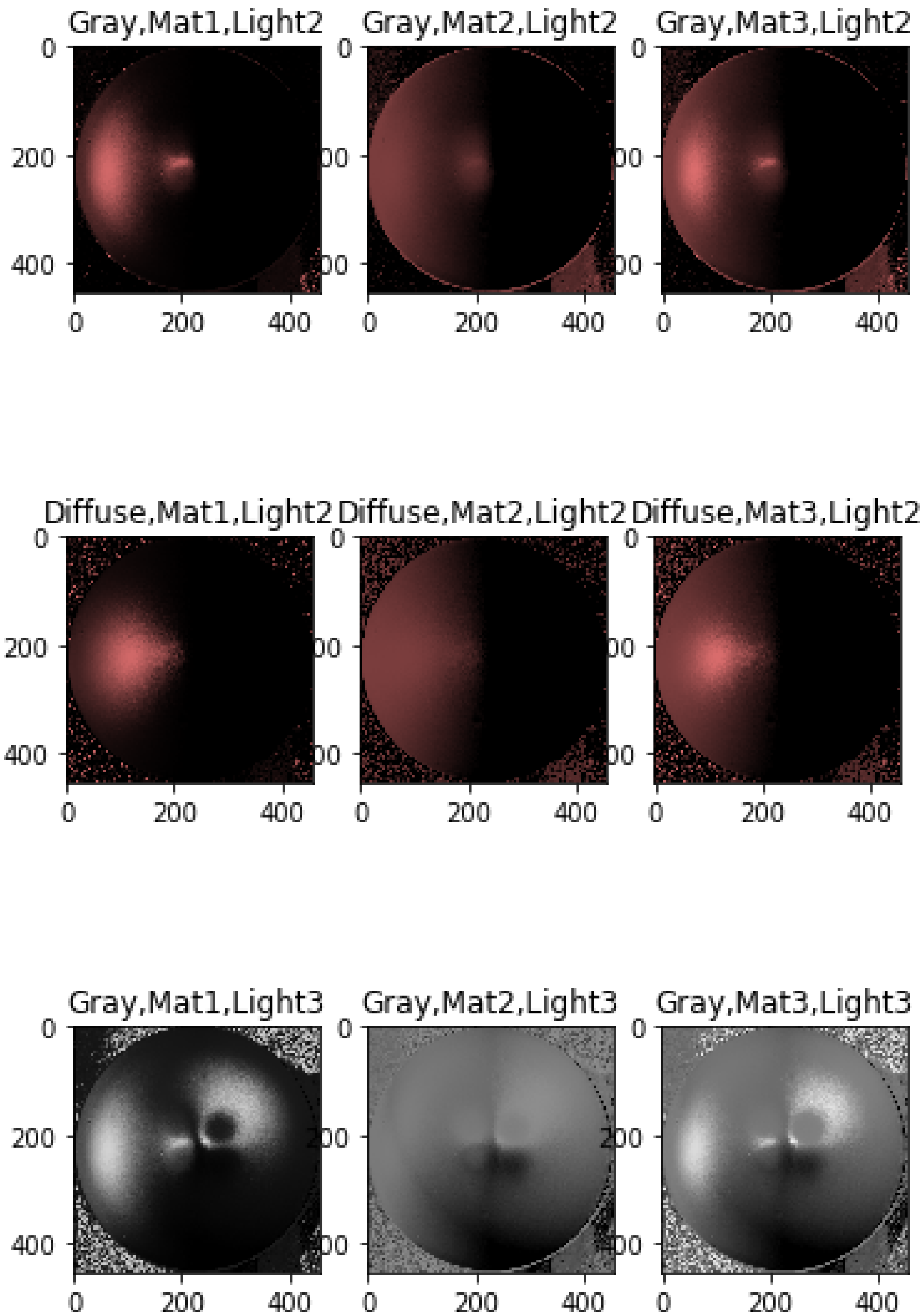
```
        image_tot += img
    return image_tot
```

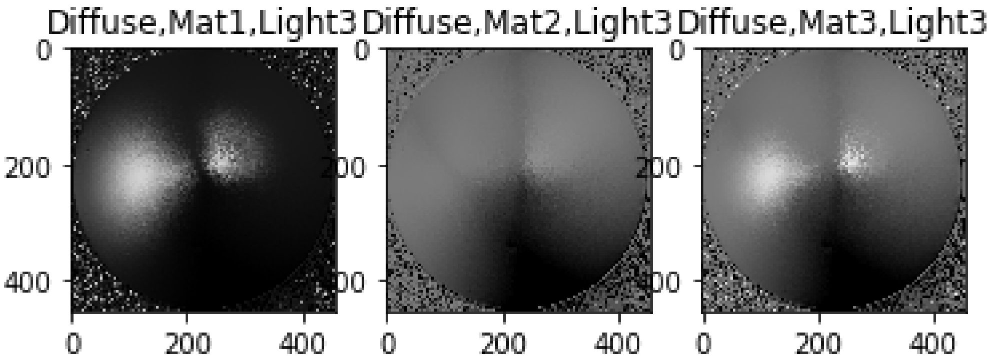
In [286]:
```
# Sphere
view=np.array([0, 0, 1]).T
materials = [[0, 0.1, 0.75, 5], [0, 0.5, 0.1, 5], [0, 0.5, 0.5, 10]]
light1 = [[-1/3, 1/3, 1/3]]
color1 = [[1, 1, 1]]

plt.figure()
normals = normals_sphere_gray
form = ["Gray,Mat1,Light1","Gray,Mat2,Light1","Gray,Mat3,Light1"]
for i in range(len(materials)):
    img = phong(normals,light1,color1,materials[i],view,mask_sphere)
    plt.subplot(1,3,i+1)
    plt.imshow(img,cmap="gray")
    plt.title(form[i])

plt.figure()
normals = normals_sphere_SG
form = ["Diffuse,Mat1,Light1","Diffuse,Mat2,Light1","Diffuse,Mat3,Light1"]
for i in range(len(materials)):
    img = phong(normals,light1,color1,materials[i],view,mask_sphere)
    plt.subplot(1,3,i+1)
    plt.imshow(img,cmap="gray")
    plt.title(form[i])


light2 = [[1, 0, 0]]
color2 = [[1, 0.5, 0.5]]

plt.figure()
normals = normals_sphere_gray
form = ["Gray,Mat1,Light2","Gray,Mat2,Light2","Gray,Mat3,Light2"]
for i in range(len(materials)):
    img = phong(normals,light2,color2,materials[i],view,mask_sphere)
    plt.subplot(1,3,i+1)
    plt.imshow(img,cmap="gray")
    plt.title(form[i])

plt.figure()
normals = normals_sphere_SG
form = ["Diffuse,Mat1,Light2","Diffuse,Mat2,Light2","Diffuse,Mat3,Light2"]
for i in range(len(materials)):
    img = phong(normals,light2,color2,materials[i],view,mask_sphere)
    plt.subplot(1,3,i+1)
    plt.imshow(img,cmap="gray")
    plt.title(form[i])
```

```
light3 = light1 + light2
color3 = color1 + color2

plt.figure()
normals = normals_sphere_gray
form = ["Gray,Mat1,Light3","Gray,Mat2,Light3","Gray,Mat3,Light3"]
for i in range(len(materials)):
    img = phong(normals,light3,color3,materials[i],view,mask_sphere)
    plt.subplot(1,3,i+1)
    plt.imshow(img,cmap="gray")
    plt.title(form[i])

plt.figure()
normals = normals_sphere_SG
form = ["Diffuse,Mat1,Light3","Diffuse,Mat2,Light3","Diffuse,Mat3,Light3"]
for i in range(len(materials)):
    img = phong(normals,light3,color3,materials[i],view,mask_sphere)
    plt.subplot(1,3,i+1)
    plt.imshow(img,cmap="gray")
    plt.title(form[i])
```



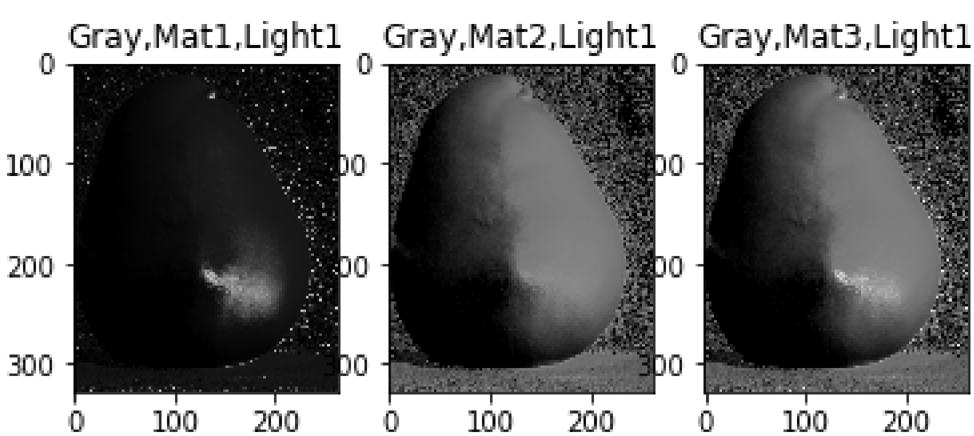


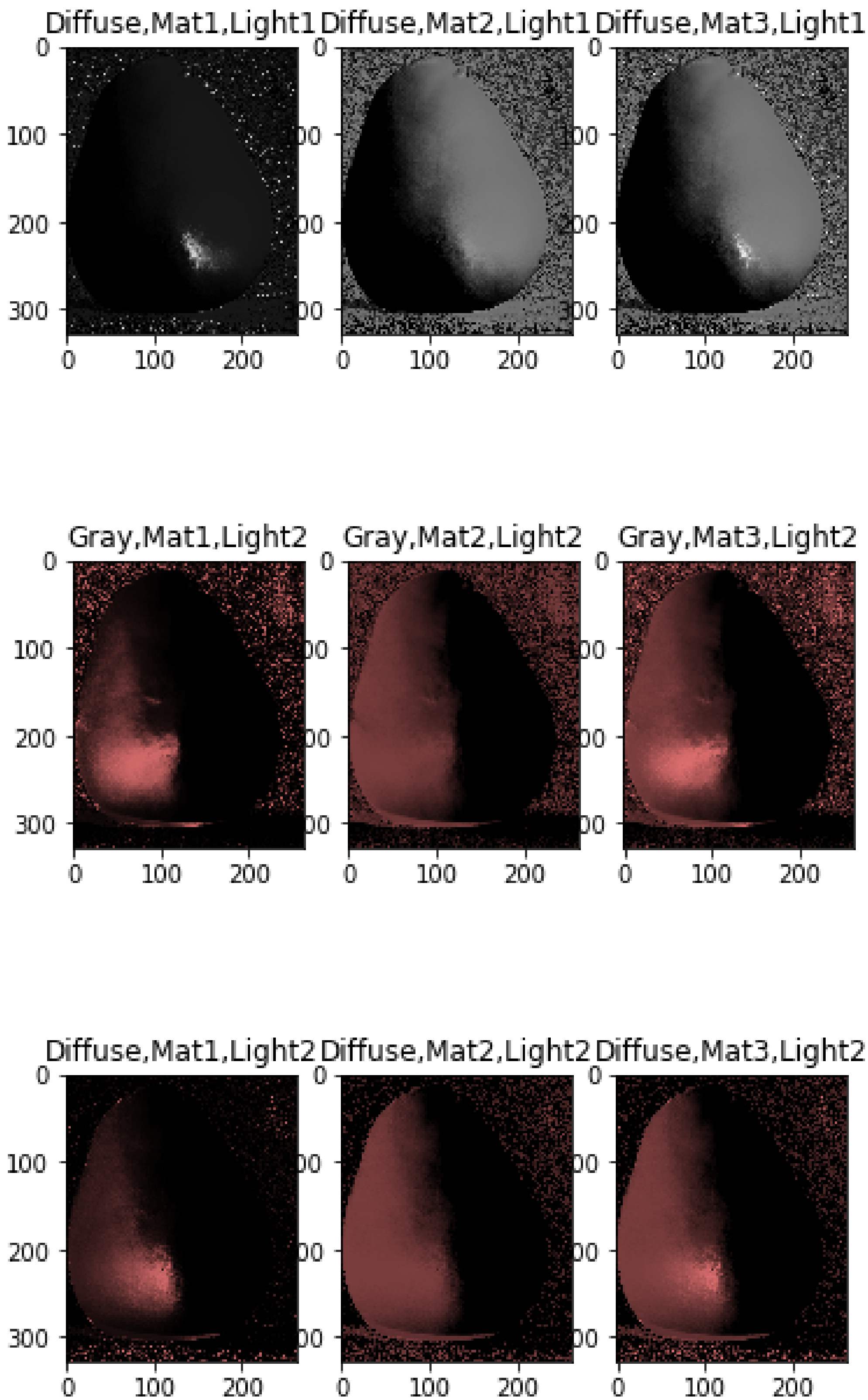

Gray,Mat1,Light2
Gray,Mat2,Light2
Gray,Mat3,Light2
Diffuse,Mat1,Light2
Diffuse,Mat2,Light2
Diffuse,Mat3,Light2
Gray,Mat1,Light3
Gray,Mat2,Light3
Gray,Mat3,Light3

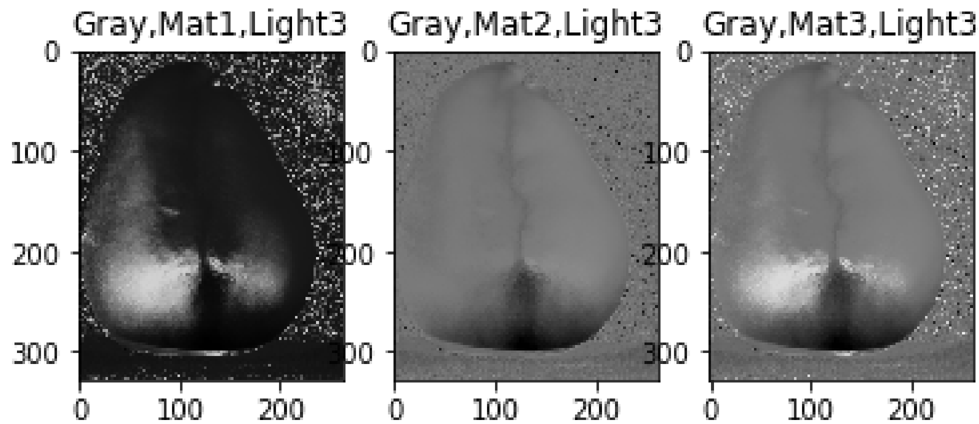


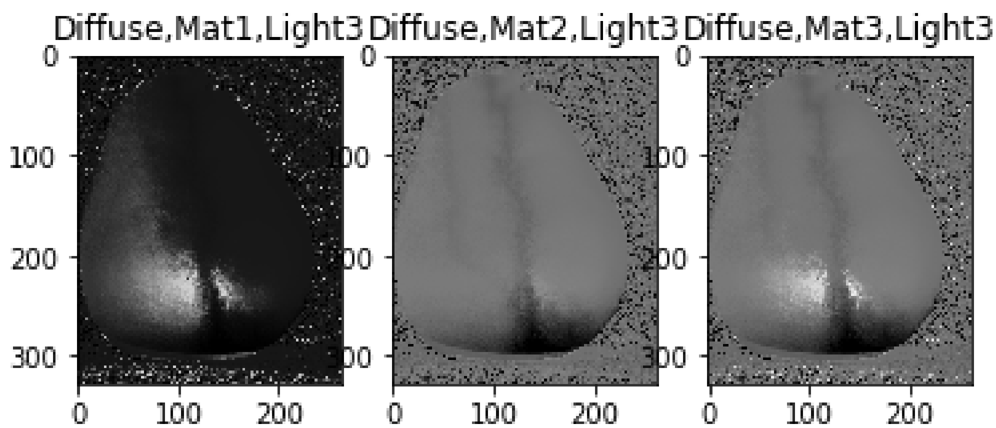
In [287]:
```
# Sphere
view=np.array([0, 0, 1]).T
materials = [[0, 0.1, 0.75, 5], [0, 0.5, 0.1, 5], [0, 0.5, 0.5, 10]]
light1 = [[-1/3, 1/3, 1/3]]
color1 = [[1, 1, 1]]

plt.figure()
normals = normals_pear_gray
form = ["Gray,Mat1,Light1","Gray,Mat2,Light1","Gray,Mat3,Light1"]
for i in range(len(materials)):
    img = phong(normals,light1,color1,materials[i],view,mask_pear)
    plt.subplot(1,3,i+1)
    plt.imshow(img,cmap="gray")
    plt.title(form[i])

plt.figure()
normals = normals_pear_SG
form = ["Diffuse,Mat1,Light1","Diffuse,Mat2,Light1","Diffuse,Mat3,Light1"]
for i in range(len(materials)):
    img = phong(normals,light1,color1,materials[i],view,mask_pear)
    plt.subplot(1,3,i+1)
    plt.imshow(img,cmap="gray")
    plt.title(form[i])


light2 = [[1, 0, 0]]
color2 = [[1, 0.5, 0.5]]

plt.figure()
normals = normals_pear_gray
form = ["Gray,Mat1,Light2","Gray,Mat2,Light2","Gray,Mat3,Light2"]
for i in range(len(materials)):
    img = phong(normals,light2,color2,materials[i],view,mask_pear)
```

```
    plt.subplot(1,3,i+1)
    plt.imshow(img,cmap="gray")
    plt.title(form[i])

plt.figure()
normals = normals_pear_SG
form = ["Diffuse,Mat1,Light2","Diffuse,Mat2,Light2","Diffuse,Mat3,Light2"]
for i in range(len(materials)):
    img = phong(normals,light2,color2,materials[i],view,mask_pear)
    plt.subplot(1,3,i+1)
    plt.imshow(img,cmap="gray")
    plt.title(form[i])


light3 = light1 + light2
color3 = color1 + color2

plt.figure()
normals = normals_pear_gray
form = ["Gray,Mat1,Light3","Gray,Mat2,Light3","Gray,Mat3,Light3"]
for i in range(len(materials)):
    img = phong(normals,light3,color3,materials[i],view,mask_pear)
    plt.subplot(1,3,i+1)
    plt.imshow(img,cmap="gray")
    plt.title(form[i])

plt.figure()
normals = normals_pear_SG
form = ["Diffuse,Mat1,Light3","Diffuse,Mat2,Light3","Diffuse,Mat3,Light3"]
for i in range(len(materials)):
    img = phong(normals,light3,color3,materials[i],view,mask_pear)
    plt.subplot(1,3,i+1)
    plt.imshow(img,cmap="gray")
    plt.title(form[i])
```




Diffuse,Mat1,Light1
Diffuse,Mat2,Light1
Diffuse,Mat3,Light1
Gray,Mat1,Light2
Gray,Mat2,Light2
Gray,Mat3,Light2
Diffuse,Mat1,Light2
Diffuse,Mat2,Light2
Diffuse,Mat3,Light2
0
100
200
300


Gray,Mat1,Light3
Gray,Mat2,Light3
Gray,Mat3,Light3


Diffuse,Mat1,Light3
Diffuse,Mat2,Light3
Diffuse,Mat3,Light3

```
In [ ]:
```

```
In [ ]:
```