HW1_Solutions

October 28, 2022

1 CSE 152A Fall 2022 – Assignment 1

- Assignment Published On: **Wed, Oct 05, 2022**
- Due On: **Fri, Oct 14, 2022 11:59 PM (Pacific Time)**

1.1 Instructions

Please answer the questions below using Python in the attached Jupyter notebook and follow the guidelines below:

- This assignment must be completed **individually**. For more details, please follow the Academic Integrity Policy and Collaboration Policy posted on lecture slides.
- All the solutions must be written in this Jupyter notebook.
- After finishing the assignment in the notebook, please export the notebook as a PDF and submit both the notebook and the PDF (i.e. the `.ipynb` and the `.pdf` files) on Gradescope.
- You may use basic algebra packages (e.g. `NumPy`, `SciPy`, etc) but you are not allowed to use open source codes that directly solve the problems. Feel free to ask the instructor and the teaching assistants if you are unsure about the packages to use.
- It is highly recommended that you begin working on this assignment early.

Late Policy: Assignments submitted late will receive a 15% grade reduction for each 12 hours late (that is, 30% per day).

1.2 Problem 1: Photometric Stereo [20 pts]

The goal of this problem is to implement Lambertian photometric stereo.

Note that the albedo is unknown and non-constant in the images you will use.

As input, your program should take in multiple images along with the light source direction for each image.

1.2.1 Data

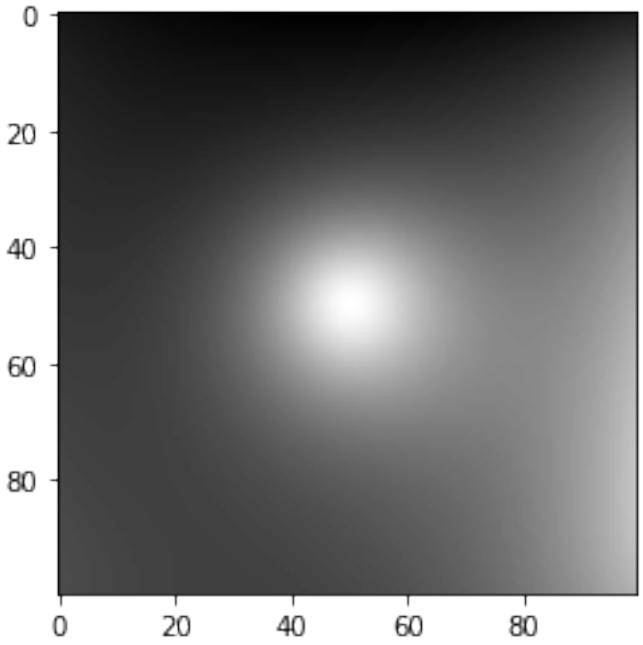
You will use synthetic images as data. These images are stored in `.pickle` files which were graciously provided by Satya Mallick. Each `.pickle` file contains

- `im1`, `im2`, `im3`, `im4`, … images.
- `l1`, `l2`, `l3`, `l4`, … light source directions.

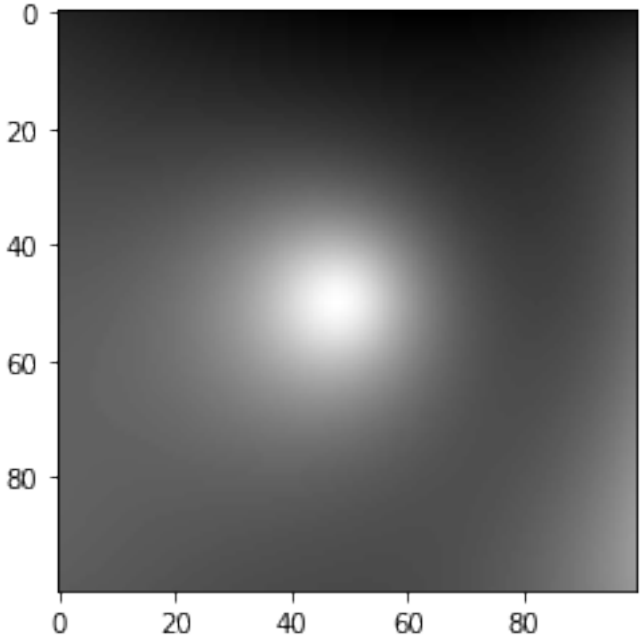
You will find all the data for this part in `synthetic_data.pickle`.

[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, 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"]))

plt.imshow(data["im2"], cmap = "gray")
plt.show()

print("Light source direction: " + str(data["l2"]))

plt.imshow(data["im3"], cmap = "gray")
plt.show()

print("Light source direction: " + str(data["l3"]))

plt.imshow(data["im4"], cmap = "gray")
plt.show()

print("Light source direction: " + str(data["l4"]))
```

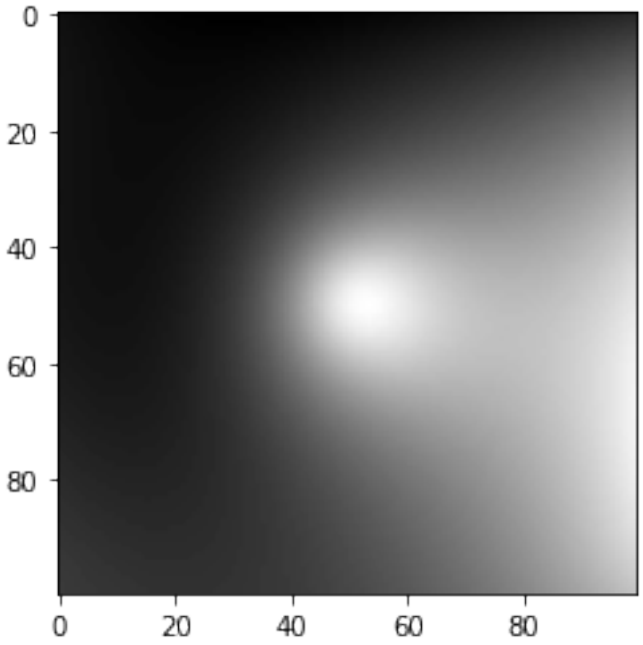
```
Keys: dict_keys(['__version__', 'l4', '__header__', 'im1', 'im3', 'im2', 'l2',
'im4', 'l1', '__globals__', 'l3'])
Image:
```



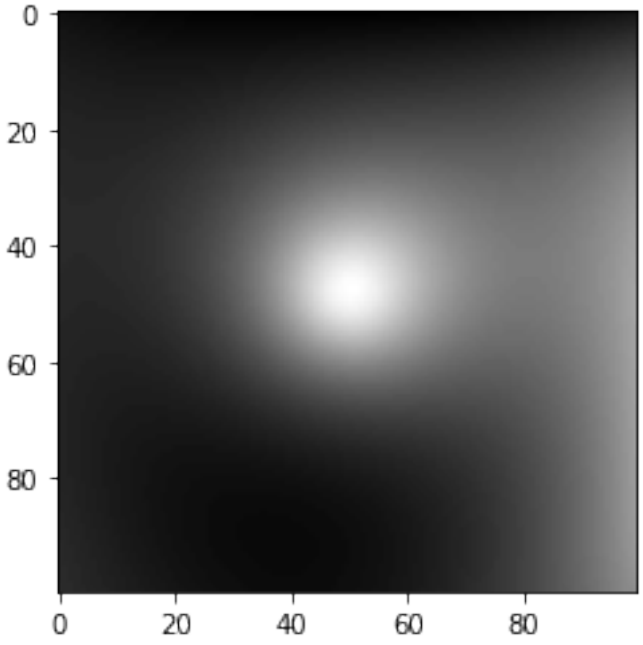

```
Light source direction: [[0 0 1]]
```




```
Light source direction: [[0.2 0.  1. ]]
```




```
Light source direction: [[-0.2  0.   1. ]]
```




```
Light source direction: [[0.  0.2 1. ]]
```

1.2.2 1(a) Photometric Stereo [8 pts]

Implement the photometric stereo technique described in the lecture. Your program should have two parts:

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

Reconstruct the depth map from the surface with the implementation of the Horn integration technique given below in `horn_integrate` function. Note that you will typically want to run the `horn_integrate` function with 10000 - 100000 iterations, meaning it will take a while.

[2]:
```
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
```

[3]:
```
def photometric_stereo(images, lights, mask, horn_niter=25000):

    """mask is an optional parameter which you are encouraged to use.
    It can be used e.g. to ignore the background when integrating the normals.
    It should be created by converting the images to grayscale and thresholding
    (only using locations for which the pixel value is above some threshold).

    The choice of threshold is something you can experiment with,
    but in practice something like 0.05 tends to work well.
    """

    # note:
    # images : (n_ims, h, w)
    # lights : (n_ims, 3)
    # mask   : (h, w)

    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_horn = np.ones(images[0].shape)

    """ ==========
    YOUR CODE HERE
    ========== """

    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_horn = np.ones(images[0].shape)
    n = images.shape[0]
    h = images.shape[1]
    w = images.shape[2]
```

```
    for i in range(h):
        for j in range(w):
            # Construct result vector
            y = []
            for k in range(n):
                y.append(images[k,i,j])
            y = np.array(y).T
            x = np.linalg.pinv(lights) @ y
            albedo[i,j] = np.linalg.norm(x)
            normals[i,j,:] = x/np.linalg.norm(x)

    H_horn = horn_integrate(normals[:,:,0]/normals[:,:,2],normals[:,:,1]/
↪normals[:,:,2],mask,horn_niter)
    return albedo, normals, H_horn
```

1.2.3 1(b) Display outputs using `im1`, `im2` and `im4` [6 Points]

The estimated albedo map.

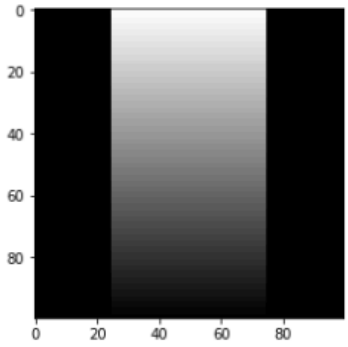
The estimated surface normals by showing both

Needle map, and

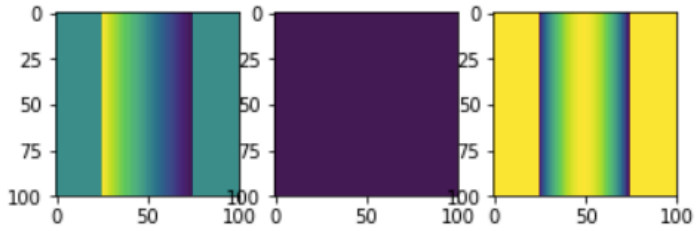
Three images showing components of surface normal.

A wireframe of depth map.

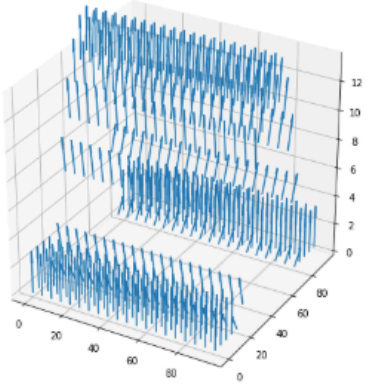
An example of outputs is shown in the figure below.



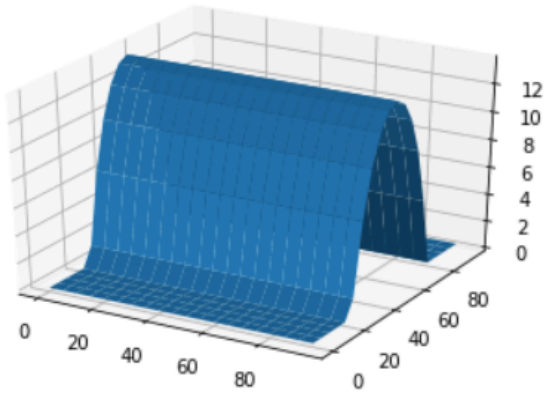
Albedo map



Normals as three separate channels



Needle map



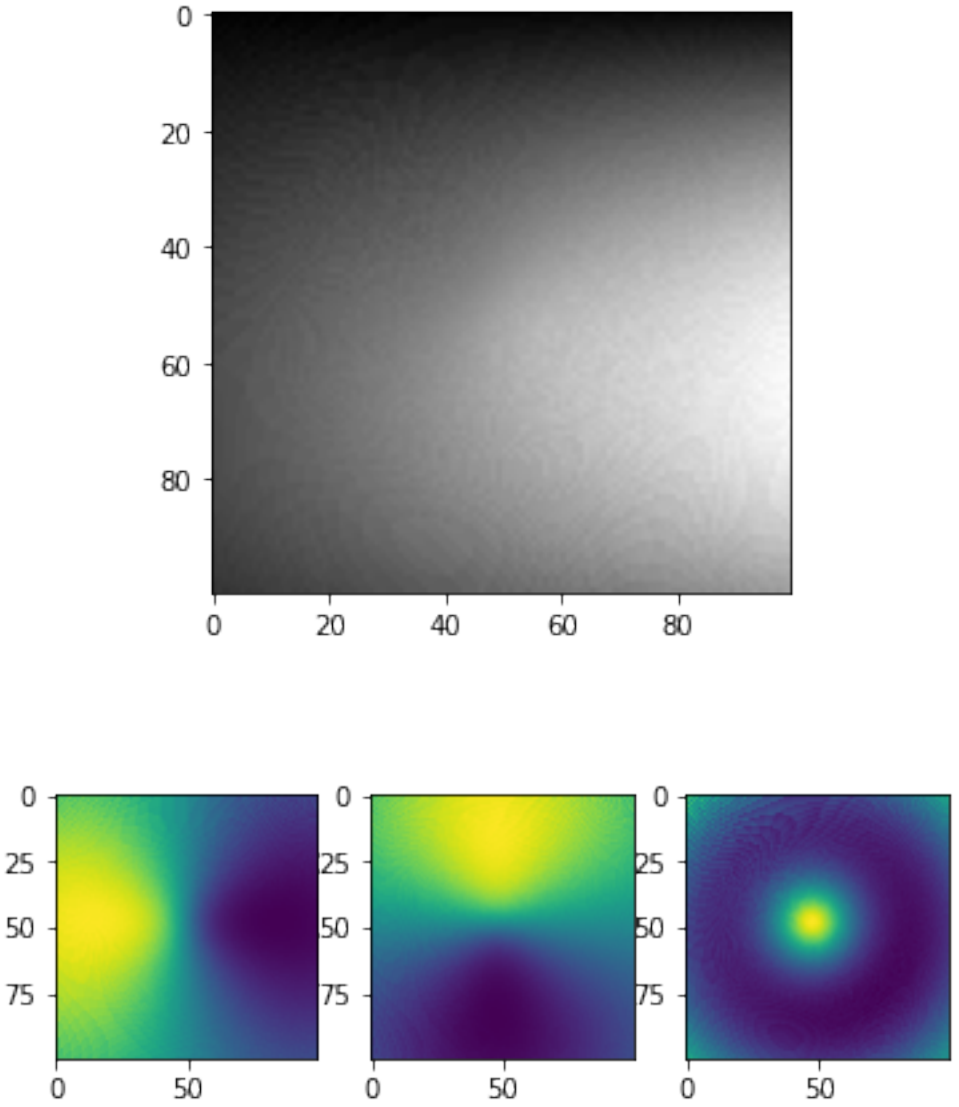
Wireframe of depth map

[4]:
```
from mpl_toolkits.mplot3d import Axes3D

# ------------------------------------------------------------------------
# The 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. You are also free to change the function
# however you wish; just make sure you get all of the required outputs.
# ------------------------------------------------------------------------

def visualize(albedo, normals, horn_depth):
    # 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()
```

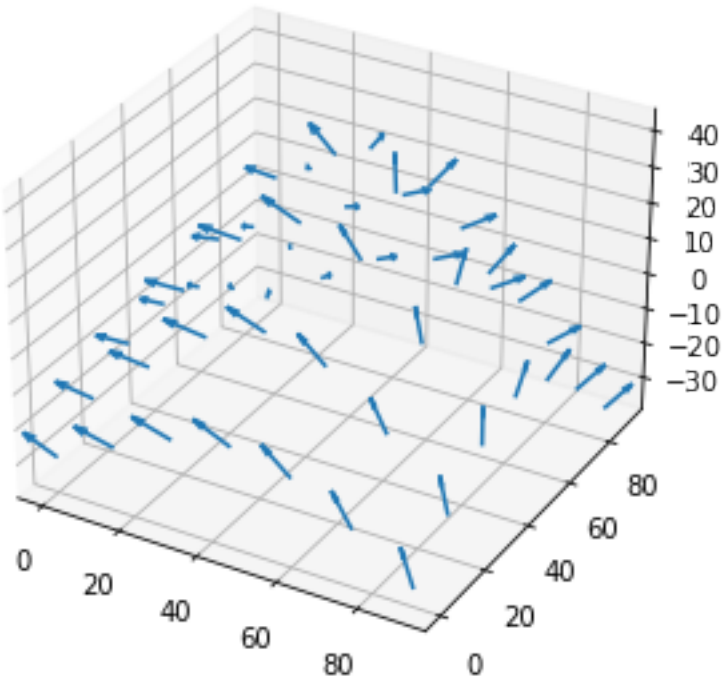
```
    # showing normals as three separate channels
    figure = plt.figure()
    ax1 = figure.add_subplot(131)
    ax1.imshow(normals[..., 0])
    ax2 = figure.add_subplot(132)
    ax2.imshow(normals[..., 1])
    ax3 = figure.add_subplot(133)
    ax3.imshow(normals[..., 2])
    plt.show()

    # 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 = horn_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=10)
    plt.show()

    # plotting wireframe depth map

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

[6]:
```
pickle_in = open("synthetic_data.pickle", "rb")
data = pickle.load(pickle_in, encoding="latin1")

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

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

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

albedo, normals, horn_depth = photometric_stereo(images, lights, mask)
```

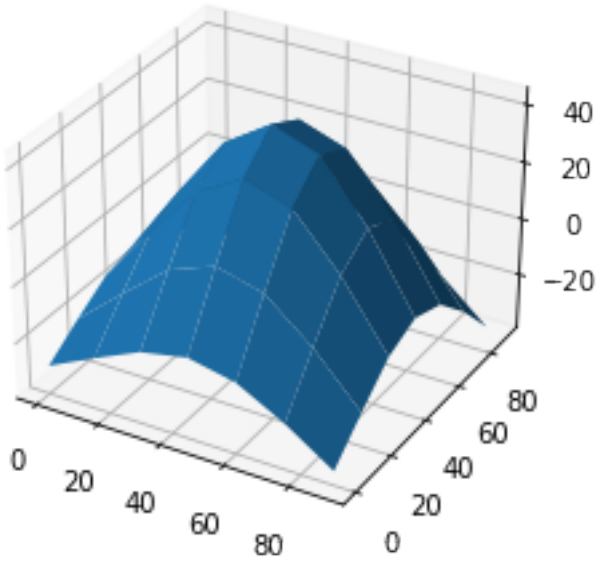
```
visualize(albedo, normals, horn_depth)
```




```
/tmp/ipykernel_2056804/4057568082.py:42: MatplotlibDeprecationWarning: Calling
gca() with keyword arguments was deprecated in Matplotlib 3.4. Starting two
minor releases later, gca() will take no keyword arguments. The gca() function
should only be used to get the current axes, or if no axes exist, create new
axes with default keyword arguments. To create a new axes with non-default
arguments, use plt.axes() or plt.subplot().
  ax = fig.gca(projection='3d')
```




```
/tmp/ipykernel_2056804/4057568082.py:50: MatplotlibDeprecationWarning: Calling
gca() with keyword arguments was deprecated in Matplotlib 3.4. Starting two
minor releases later, gca() will take no keyword arguments. The gca() function
should only be used to get the current axes, or if no axes exist, create new
axes with default keyword arguments. To create a new axes with non-default
arguments, use plt.axes() or plt.subplot().
  ax = fig.gca(projection='3d')
```



1.2.4 Display outputs using all four images (most accurate result) [6 points]

The estimated albedo map.

The estimated surface normals by showing both

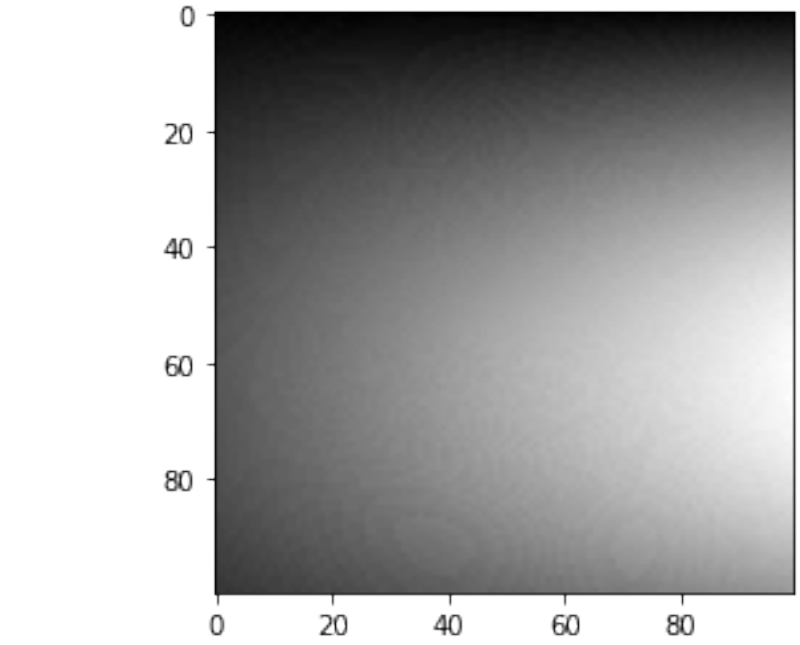
Needle map, and

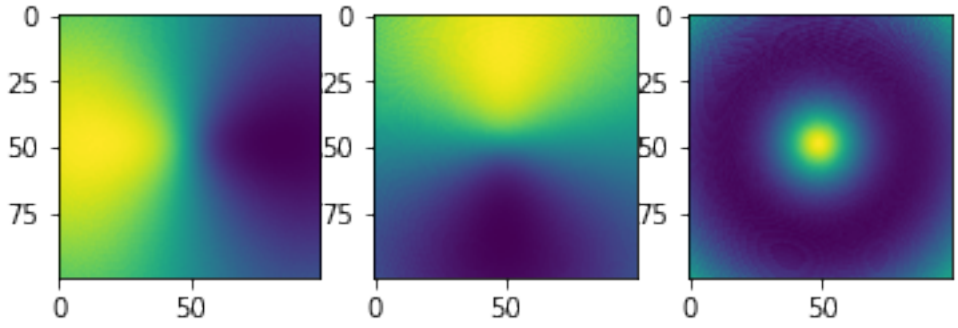
Three images showing components of surface normal.

A wireframe of depth map.

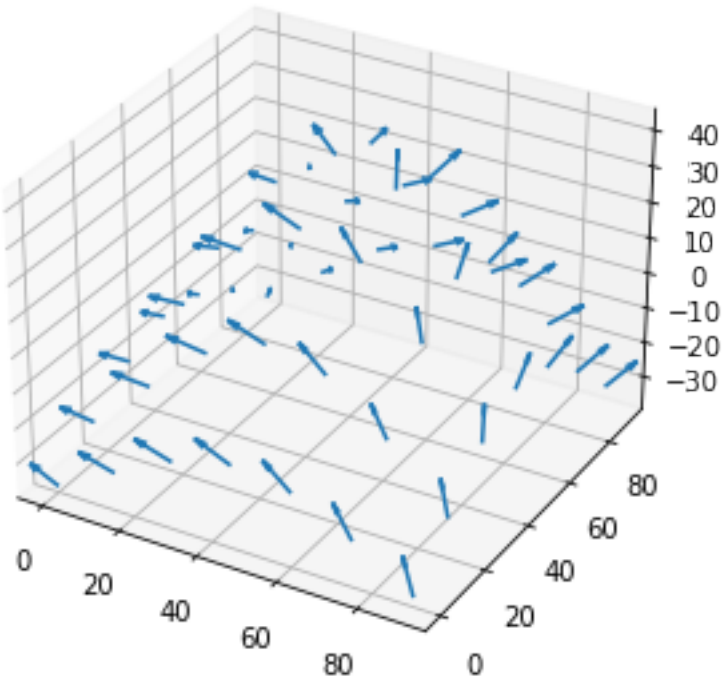
[7]:
```
""" ==========
YOUR CODE HERE
========== """

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, horn_depth = photometric_stereo(images, lights, mask)
```

```
visualize(albedo, normals, horn_depth)
```

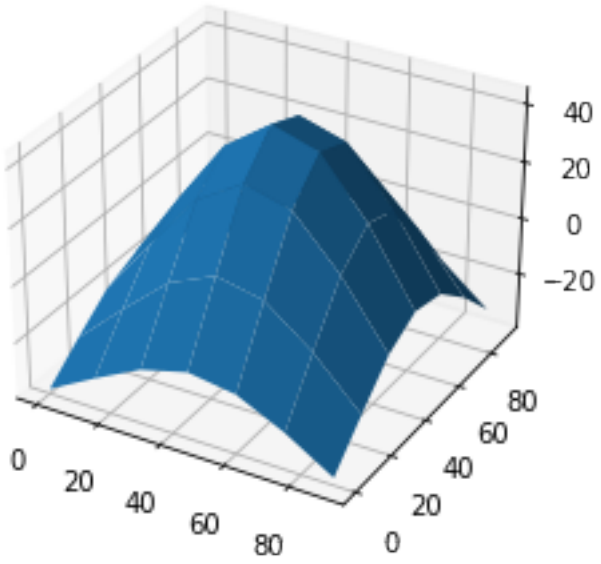





```
/tmp/ipykernel_2056804/4057568082.py:42: MatplotlibDeprecationWarning: Calling
gca() with keyword arguments was deprecated in Matplotlib 3.4. Starting two
minor releases later, gca() will take no keyword arguments. The gca() function
should only be used to get the current axes, or if no axes exist, create new
axes with default keyword arguments. To create a new axes with non-default
arguments, use plt.axes() or plt.subplot().
  ax = fig.gca(projection='3d')
```




```
/tmp/ipykernel_2056804/4057568082.py:50: MatplotlibDeprecationWarning: Calling
gca() with keyword arguments was deprecated in Matplotlib 3.4. Starting two
minor releases later, gca() will take no keyword arguments. The gca() function
should only be used to get the current axes, or if no axes exist, create new
axes with default keyword arguments. To create a new axes with non-default
arguments, use plt.axes() or plt.subplot().
  ax = fig.gca(projection='3d')
```



1.3 Problem 2: Image Rendering [20 points]

In this exercise, we will render the image of a face with two different point light sources using a Lambertian reflectance model. We will use two albedo maps, one uniform and one that is more realistic. The face heightmap, the light sources, and the two albedo are given in facedata.npy for Python (each row of the `lightsource` variable encode a light location). The data from facedata.npy is already provided to you.

Note: Please make good use out of subplot to display related image next to eachother.

2(a) Plot the face in 2-D [2 pts]

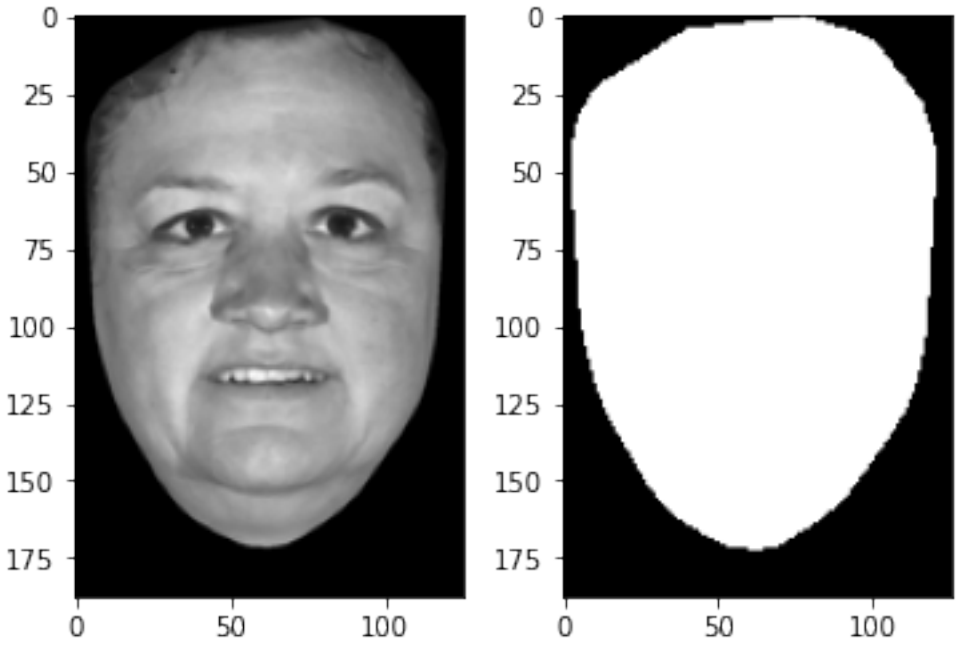
Plot both albedo maps using imshow. Explain what you see.

[8]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.path import Path
import matplotlib.patches as patches

# Load facedata.npy as ndarray
face_data = np.load('facedata.npy',encoding='latin1', allow_pickle=True)

# Load albedo matrix
albedo = face_data.item().get('albedo')

# Load uniform albedo matrix
uniform_albedo = face_data.item().get('uniform_albedo')
```

```
# Load heightmap
heightmap = face_data.item().get('heightmap')

# Load light source
light_source = face_data.item().get('lightsource')
```

[9]:
```
# Plot the face in 2-D (plot both albedo maps using imshow)
""" ==========
YOUR CODE HERE
========== """

fig, ax = plt.subplots(1,2)
ax[0].imshow(albedo, cmap=plt.get_cmap('gray'))
ax[1].imshow(uniform_albedo, cmap=plt.get_cmap('gray'))
```

[9]:
```
<matplotlib.image.AxesImage at 0x7fee30c64310>
```



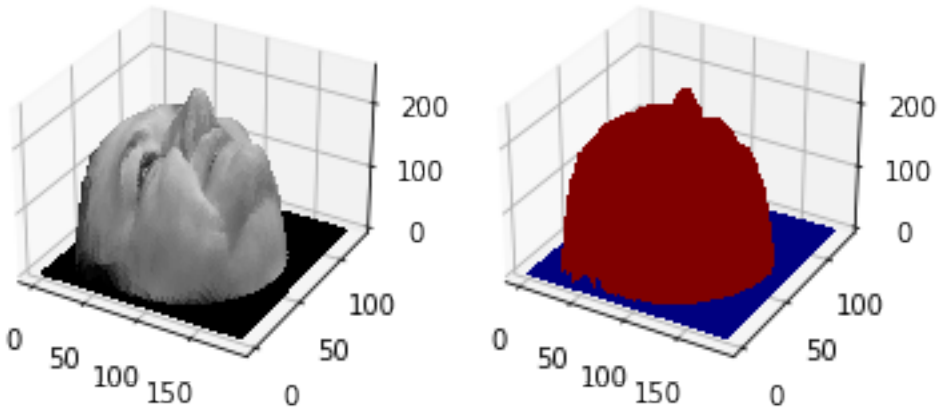
2(b) Plot the face in 3-D [2 pts]

Using both the heightmap and the albedo, plot the face using plot_surface. Do this for both albedos. Explain what you see.

[11]:
```
# Plot the face in 3-D
# (Using the heightmap & albedo plot the faces using plot_surface)
""" ==========
YOUR CODE HERE
```

```
========== """

xx, yy = np.mgrid[0:heightmap.shape[0], 0:heightmap.shape[1]]

fig = plt.figure()

ax = fig.add_subplot(1, 2, 1, projection='3d')
ax.plot_surface(xx, yy, heightmap, rstride=1, cstride=1, cmap=plt.cm.gray,␣
↪facecolors=plt.cm.gray(albedo),
                        linewidth=0, antialiased=False)
ax = fig.add_subplot(1, 2, 2, projection='3d')
ax.plot_surface(xx, yy, heightmap, rstride=1, cstride=1, cmap=plt.cm.gray,␣
↪facecolors=plt.cm.jet(uniform_albedo*1.0),
                        linewidth=0, antialiased=False)
```

[11]: `<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x7fee2c73aee0>`



2(c) Surface normals [8 pts]

Calculate the surface normals and display them as a quiver plot using quiver in matplotlib.pyplot in Python. Recall that the surface normals are given by

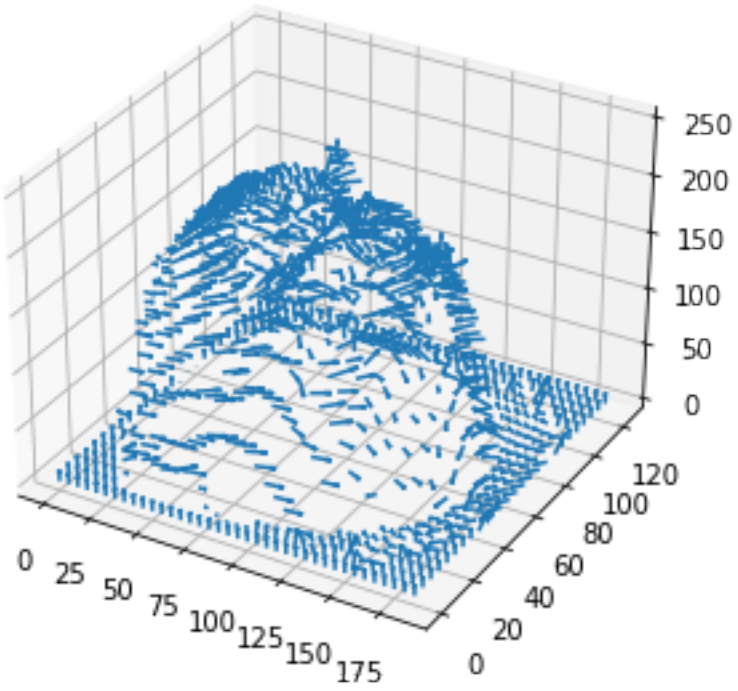
$$[-\frac{\delta f}{\delta x}, -\frac{\delta f}{\delta y}, 1]. \tag{1}$$

Also, recall, that each normal vector should be normalized to unit length.

[12]:
```
# Compute and plot the surface normals
# (make sure that the normal vector is normalized to unit length)
""" ==========
YOUR CODE HERE
========== """
```

```
from scipy.signal import convolve

A = np.array([[0,1/2,0],[0,0,0],[0,-1/2,0]])
B = np.array([[0,0,0],[1/2,0,-1/2],[0,0,0]])
dx = convolve(heightmap, B, mode='same')
dy = convolve(heightmap, A, mode='same')

normals = np.stack((np.zeros(heightmap.shape),np.zeros(heightmap.shape),np.
 ↪ones(heightmap.shape)))
for i in range(normals.shape[1]):
    for j in range(normals.shape[2]):
        normals[0,i,j] = -dx[i,j]
        normals[1,i,j] = -dy[i,j]
        normals[:,i,j] = normals[:,i,j]/np.linalg.norm(normals[:,i,j])

stride = 5
X, Y = np.meshgrid(np.arange(0,np.shape(normals)[1], stride),
                           np.arange(0,np.shape(normals)[2], stride))

Z = heightmap[::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=10)
plt.show()
```

```
/tmp/ipykernel_2056804/1052691021.py:30: MatplotlibDeprecationWarning: Calling
gca() with keyword arguments was deprecated in Matplotlib 3.4. Starting two
minor releases later, gca() will take no keyword arguments. The gca() function
should only be used to get the current axes, or if no axes exist, create new
axes with default keyword arguments. To create a new axes with non-default
arguments, use plt.axes() or plt.subplot().
  ax = fig.gca(projection='3d')
```



2(d) Render images [8 pts]

For each of the two albedos, render three images. One for each of the two light sources, and one for both light-sources combined. Display these in a 2×3 subplot figure with titles. Recall that the general image formation equation is given by

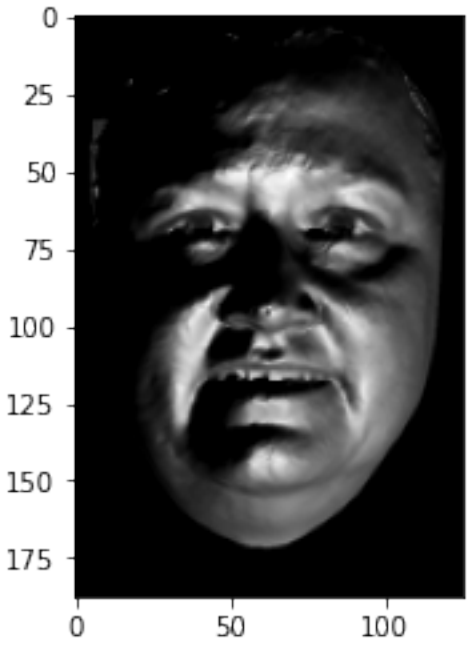
$$I = a(x,y)\hat{\mathbf{n}}(x,y)^{\top}\hat{\mathbf{s}}(x,y)s_0 \tag{2}$$

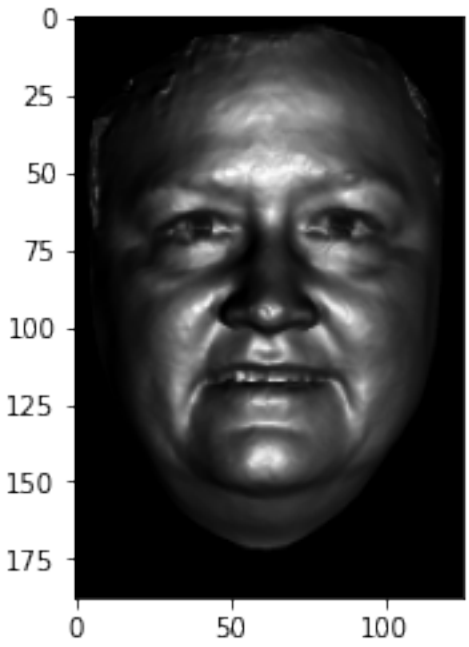
where $a(x,y)$ is the albedo for pixel (x,y), $\hat{n}(x,y)$ is the corresponding surface normal, $\hat{s}(x,y)$ the light source direction, s_0 the light source intensity. Let the light source intensity be 1 and make the 'distant light source assumption'. Use imshow with appropriate keyword arguments .

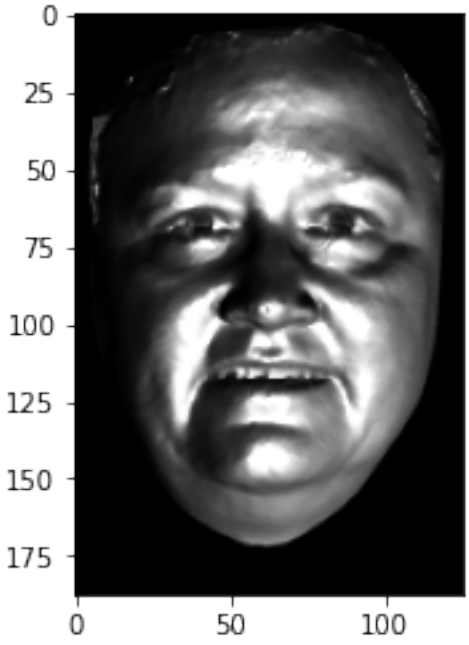
[13]:
```
# Render Images

def lambertian(normals, light, albedo, intensity, mask):
    """ ==========
    YOUR CODE HERE
    ========== """

    temp1 = np.zeros(albedo.shape)
    for i in range(normals.shape[1]):
        for j in range(normals.shape[2]):
            temp1[i,j] = normals[:,i,j].dot(light/np.linalg.norm(light))
            temp1[i,j] = np.max([0.0, temp1[i,j]])
```

```
    temp2 = temp1 * albedo * intensity
    image  = temp2 * mask
    return image
```

[15]:
```
# For each of the two albedos, render three images.
# One for each of the two light sources, and one for both light-sources␣
 ↪combined.
""" ==========
YOUR CODE HERE
========== """

mask = np.ones(heightmap.shape)
image1 = lambertian(normals, light_source[0,:], albedo, 1, mask)
plt.imshow(image1, cmap=plt.get_cmap('gray'))
plt.show()
image2 = lambertian(normals, light_source[1,:], albedo, 1, mask)
plt.imshow(image2, cmap=plt.get_cmap('gray'))
plt.show()
image3 = image1 + image2
image3[image3 > 1] = 1.0
plt.imshow(image3, cmap=plt.get_cmap('gray'))
plt.show()
```

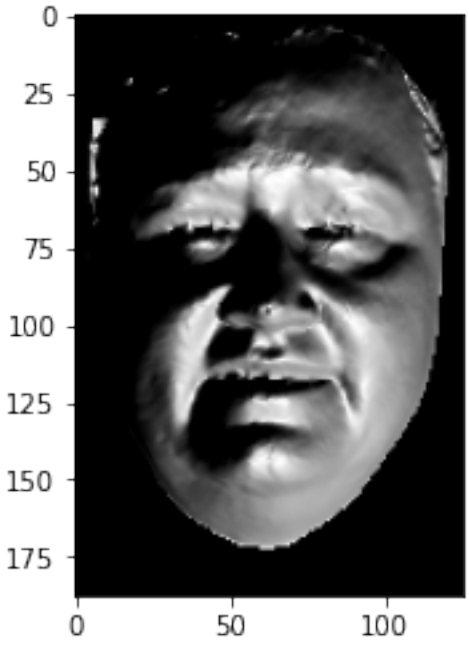


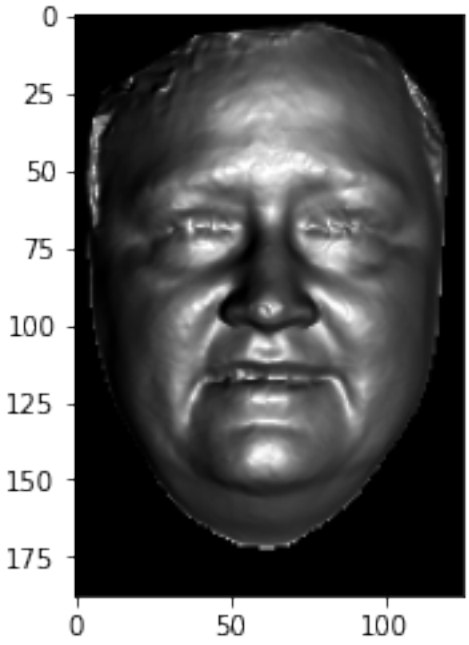


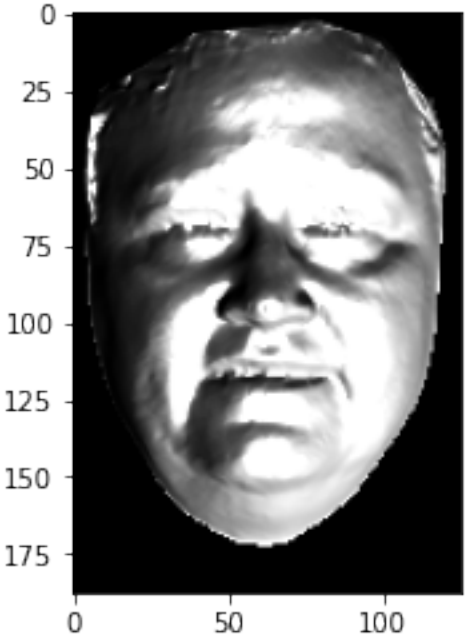


[16]:
```
image1 = lambertian(normals, light_source[0,:], uniform_albedo, 1, mask)
plt.imshow(image1, cmap=plt.get_cmap('gray'))
plt.show()
image2 = lambertian(normals, light_source[1,:], uniform_albedo, 1, mask)
```

```
plt.imshow(image2, cmap=plt.get_cmap('gray'))
plt.show()
image3 = image1 + image2
image3[image3 > 1] = 1.0
plt.imshow(image3, cmap=plt.get_cmap('gray'))
plt.show()
```






0
25
50
75
100
125
150
175
0
50
100

CSE 152A Assignment 1: Solutions
Introduction to Computer Vision
Fall 2022

Problem 3: Homogeneous Coordinates and Vanishing Points

(a) Say, choose $w = 1$ for $(3, 5, 1)^\top$ and $w = -1$ for $(-3, -5, -1)^\top$.

(b) Equation of line in usual Cartesian coordinates: $x + y - 2 = 0$.
Homogeneous representation: $(1, 1, -2)^\top$.

(c) The point must lie on line $\mathbf{l}_1$ and the point must also lie on line $\mathbf{l}_2$. Let the point be x. Then, from **Fact 1**, $\mathbf{l}_1^\top \mathbf{x} = 0$ and $\mathbf{l}_2^\top \mathbf{x} = 0$ is a necessary and sufficient condition for the point of intersection of lines $\mathbf{l}_1$ and $\mathbf{l}_2$. Since there must be a unique point of intersection and $\mathbf{l}_1 \times \mathbf{l}_2$ is the vector orthogonal to both $\mathbf{l}_1$ and $\mathbf{l}_2$, choosing $\mathbf{x} = \mathbf{l}_1 \times \mathbf{l}_2$ gives the point of intersection.

(d) To find intersection point in homogeneous coordinates, first compute cross product of homogeneous lines $(1, 1, -5)^\top$ and $(4, -5, 7)^\top$, which is $(-18, -27, -9)^\top$. To convert back to Cartesian coordinates, we divide by the third coordinate and drop it, to obtain $(2, 3)^\top$.
The coordinate geometry approach would be to solve the system of equations:

$$x + y = 5$$
$$4x - 5y = -7$$

The solution is $x = 2, y = 3$, so coordinate geometry gives the same point of intersection as projective geometry.

(e) The two lines are parallel, both have slope $\frac{-1}{2}$.
They do not intersect at a finite point on the Euclidean plane.

(f) Homogeneous representation: $(1, 2, 1)^\top$ and $(3, 6, -2)^\top$.
Intersection point in homogeneous coordinates: $(1, 2, 1) \times (3, 6, -2)^\top = (-10, -5, 0)^\top$.
This is called the vanishing point, which is the image of a point at infinity (or an ideal point).

(g) Yes, homogeneous coordinates provide a uniform treatment of line intersection. The point of intersection can be computed in the same manner for parallel lines as for non-parallel ones in homogeneous coordinates, one does not have to deal with special cases for parallelism.

(h) For a line $\mathbf{l}$ to pass through $\mathbf{x}_1$ and $\mathbf{x}_2$, both $\mathbf{x}_1$ and $\mathbf{x}_2$ must lie on the line $\mathbf{l}$. Thus, from **Fact 1**, $\mathbf{x}_1^\top \mathbf{l} = 0$ and $\mathbf{x}_2^\top \mathbf{l} = 0$. The vector $\mathbf{l}$ must be orthogonal to both $\mathbf{x}_1$ and $\mathbf{x}_2$, such a vector $\mathbf{x}_1 \times \mathbf{x}_2$. Since a unique line must pass through the two points $\mathbf{x}_1$ and $\mathbf{x}_2$, the line is given by $\mathbf{l} = \mathbf{x}_1 \times \mathbf{x}_2$.

Problem 4: Camera Matrices and Rigid-Body Transformations

(a) There are two vectors orthogonal to $\widehat{\mathbf{i}}'$ and $\widehat{\mathbf{j}}'$. One is $\mathbf{a} = \widehat{\mathbf{i}}' \times \widehat{\mathbf{j}}'$, the other is $\mathbf{b} = \widehat{\mathbf{i}}' \times \widehat{\mathbf{j}}' = -\mathbf{a}$. Computing the cross-product, we have $\mathbf{a} = (-0.1225, -0.1493, 0.9812)^\top$. Let θ_a be the angle between $\mathbf{a}$ and $\widehat{\mathbf{k}}'$. Then, we have

$$\cos\theta_a = \mathbf{a} \cdot \widehat{\mathbf{k}} = (-0.1225, -0.1493, 0.9812)^\top \cdot (0, 0, 1)^\top = 0.9812.$$

Since $\cos\theta_a > 0$, we have $|\theta_a| < 90°$, so we choose $\widehat{\mathbf{k}}' = \mathbf{a}$.

(b) From the lecture notes, such a rotation matrix is

$$\mathbf{R} = \begin{bmatrix} \widehat{\mathbf{i}}'^\top \\ \widehat{\mathbf{j}}'^\top \\ \widehat{\mathbf{k}}'^\top \end{bmatrix} = \begin{bmatrix} 0.9 & 0.4 & 0.1732 \\ -0.41833 & 0.90427 & 0.08539 \\ -0.1225 & -0.1493 & 0.9812 \end{bmatrix}.$$

(c) The camera center is given by $\mathbf{C} = (-1, -2, -3)^\top$ in world coordinates. From the lecture notes, the extrinsic parameter matrix is given by

$$\begin{bmatrix} \mathbf{R} & -\mathbf{RC} \\ \mathbf{0}^\top & 1 \end{bmatrix} = \begin{bmatrix} 0.9 & 0.4 & 0.1732 & 2.2196 \\ -0.41833 & 0.90427 & 0.08539 & 1.6464 \\ -0.1225 & -0.1493 & 0.9812 & 2.5224 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

(d) From lecture notes, the intrinsic parameter matrix is given by

$$\begin{bmatrix} -1050 & 0 & 10 \\ 0 & -1050 & -5 \\ 0 & 0 & 1 \end{bmatrix}.$$

(e) From lecture notes, the projection matrix is given by

$$\mathbf{P} = \mathbf{K}[\mathbf{R}| - \mathbf{RC}] = \begin{bmatrix} -946.225 & -421.493 & -172.053 & -2305.371 \\ 439.85 & -948.737 & -94.565 & -1741.31 \\ -0.1225 & -0.1493 & 0.9812 & 2.5225 \end{bmatrix}.$$

(f) The obtained image of the circle should be an *ellipse*.