# CSC3104 Computer Vision

## Lab 2: Image Segmentation: Edges

### A. 3D Surface Plot

So far, we have only tried to plot or display 2D images. If we would like to visualize the edges, we have to plot a 2D images in a 3D space, where the intensity value of each pixel will now serve as the height. As shown below is a sample code of plotting a 2D image in a 3D space.

```
1   import cv2                                          #import OpenCV
2   import numpy as np                                  #import numpy package
3   from matplotlib import pyplot as pt                 #import pyplot for plotting figure
4   from mpl_toolkits import mplot3d                    #import functions for 3D plotting
5
6   img = cv2.imread("cameraman.png", 0)                #Read cameraman
7   img = cv2.resize(img, (128,128))                    #Reduce the resolution to 128x128
8
9   [nrow,ncol] = img.shape                             #Get the dimension (row and col)
10  [xCoor, yCoor] = np.mgrid[0:nrow, 0:ncol]           #Generate the coordinates
11
12  pt.figure()                                         #Create a figure window for plotting
13  ax = pt.axes(projection='3d')                       #Create a 3D plot
14  ax.plot_surface(xCoor, yCoor, img, cmap=pt.cm.jet)  #Plot the image in the 3D plot
15
16  pt.show()                                           #Display the plot
```

The sample code is straight-forward. We first import the set of packages or functions that we need. All of us should already been quite familiar with those stated in line 1-3. The mpl_toolkits in line 4 is something new to us. It is basically a collection of functions that extends the functionality of matplotlib. For example, mplot3d gives us a set of functions that can be used to create a 3D plot.

After we have imported all the packages or functions, we then proceed to read the test image, our favorite cameraman. In line 7, the spatial resolution of the image is reduced to 128x128. This line is only necessary if the system takes too long to produce the surface plot. Depends on the specification of your computer, it might take a while to see the plot.

In line 9, the dimension (number of rows and columns) of the image is obtained. The dimension is needed so that we can use it to generate a coordinate map in line 10. This function is very useful and is commonly used to simplify coding required to process a set of pixels. Hence, try to take some time to understand how it works. In fact, the idea is quite simple. Given the dimension of an array, it will produce two different arrays based on the dimension. The first array is used to save all the x coordinates, whereas the second array is used to save all the y coordinates.

Let us look at the example shown in Figure 1. Assuming that we are now trying to do this for a 3x3 array. As mentioned earlier, the function will give us two different arrays, one for the x coordinates as shown in Figure 1(a), and one for the y coordinates as shown in Figure 1(b). When we combined the two (in realworld, we do not have to combine the two, this is just to show you the meaning of all those values), we can see from Figure 1(c) that they actually forms a set of coordinates.

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 2 | 2 |

(a) xCoor

| | | |
|---|---|---|
| 0 | 1 | 2 |
| 0 | 1 | 2 |
| 0 | 1 | 2 |

(b) yCoor

| | | |
|---|---|---|
| [0,0] | [0,1] | [0,2] |
| [1,0] | [1,1] | [1,2] |
| [2,0] | [2,1] | [2,2] |

(c) Combined

Figure 1

In this case, we are incrementing the coordinate by 1 each time (when move from one pixel to another). However, one can also choose to increase the coordinate by 0.5 each time instead of 1. In addition to this, one can also choose to start from, for example, [5,5] instead of [0,0]. Depends on what you are trying to code, this function can be handy in certain situations when you need to produce a coordinate map. If you need some examples on this, you may refer to the low-pass or high-pass filter function.

**Exercise**

Create the sample images shown in Figure 2 and try to visualize the edges by using a 3D surface plot.

| | | | | |
|---|---|---|---|---|
| 255 | 255 | 0 | 0 | 0 |
| 255 | 255 | 0 | 0 | 0 |
| 255 | 255 | 0 | 0 | 0 |
| 255 | 255 | 0 | 0 | 0 |
| 255 | 255 | 0 | 0 | 0 |

(a)

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 255 | 255 | 255 | 0 |
| 0 | 255 | 255 | 255 | 0 |
| 0 | 255 | 255 | 255 | 0 |
| 0 | 0 | 0 | 0 | 0 |

(b)

| | | | | |
|---|---|---|---|---|
| 255 | 0 | 0 | 0 | 0 |
| 0 | 255 | 0 | 0 | 0 |
| 0 | 0 | 255 | 0 | 0 |
| 0 | 0 | 0 | 255 | 0 |
| 0 | 0 | 0 | 0 | 255 |

(c)

Figure 2

## B. Image Gradient Calculation

There are two criteria that we can look at then calculating the gradient of a point, (i) the magnitude which tells us the steepness of the slope, and (ii) orientation that tells us the direction of the slope. Please take note that the orientation is always pointing from the lowest to highest direction. In the sample code below, what we are trying to do is to validate the calculations that we have gone through.

```
img = np.array([[100,100,100],[100,0,100],[100,100,200]],dtype=np.float32)   #Create sample image
kernel_hori = np.array([[-1,0,1],[-1,0,1],[-1,0,1]],dtype=np.float32)/2       #Create kernel (horizontal)
kernel_vert = np.array([[1,1,1],[0,0,0],[-1,-1,-1]],dtype=np.float32)/2       #Create kernel (vertical)

grad_hori = cv2.filter2D(img,-1,kernel_hori)                                  #Apply 2D convolution
grad_vert = cv2.filter2D(img,-1,kernel_vert)                                  #Apply 2D convolution

grad_mag = np.sqrt(np.power(grad_hori,2)+np.power(grad_vert,2))               #Calculate the magnitude

orientation_rad = np.arctan2(grad_vert,grad_hori)                             #Calculate the orientation
orientation_deg = (orientation_rad*180)/np.pi                                 #Convert radians to degrees
```

In line 1-3, we first create a 3x3 sample image and then follows by the kernels to be used for the horizontal direction and vertical direction respectively. As you can see from the sample code, the image and the kernels are all saved in the form of np.float32, instead of np.uint8 or np.int.

It could be your first time using np.float32. It stands for 32 bits floating point format. Generally there are two reasons to use this. Firstly, the cv2.filter2D function only accepts arrays stored in certain format and np.float32 is one of them. Although it also accepts np.uint8, we cannot use it because we need to store negative numbers in those arrays. Secondly, np.float32 can be used to store floating point numbers.

Next, we will move on to apply 2D convolution to the sample image in both the horizontal and vertical directions in line 5-6. It is not hard to see that the first parameter for the function is the input image and the third parameter is the kernel. The second parameter is to specify the depth (or precision). It controls how many bits should be used to store the output. In this case, the value of '1' means the depth should just follow the input image.

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 100 | 100 | 100 |
| 1 | 100 | 0 | 100 |
| 2 | 100 | 100 | 200 |

(a) Sample Image

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0.00 | 0.00 | 0.00 |
| 1 | 0.00 | 70.71 | 50.00 |
| 2 | 0.00 | 50.00 | 0.00 |

(b) Gradient Magnitude

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | -45 | -90 |
| 2 | 0 | 0 | 0 |

(c) Gradient Orientation (Deg)

Figure 3

As shown in Figure 3(b) and (c) are the magnitude and orientation that you would obtain by running the sample code. Overall, the results match with what we have discussed previously, where the magnitude is 70.71 and oriented at -45$^{O}$. If we compared the magnitude of point [1,1] with point [1,2], we can see that the magnitude of the former is higher, because the intensity changes from the center pixel [1,1] to the southwest pixel [2,2] is higher than going from pixel [1,2] to point [0,2]. Hence, the magnitude is in fact telling us the steepness from one point to another. Of course, you will need to first check the orientation to see which direction the gradient is pointing towards. It is very important to remember that the orientation is always telling us the direction from the lowest point to the highest point.

**Exercise**

Create the sample images shown in Figure 4 and then calculate the gradient magnitude and orientation for each of them. Try to check and understand the meaning behind those values that you obtained.

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 100 | 100 | 100 |
| 1 | 100 | 200 | 100 |
| 2 | 100 | 100 | 50 |

(a)

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 100 | 100 | 50 |
| 1 | 50 | 200 | 100 |
| 2 | 100 | 100 | 20 |

(b)

Figure 4

## C. Sobel Edge Detection

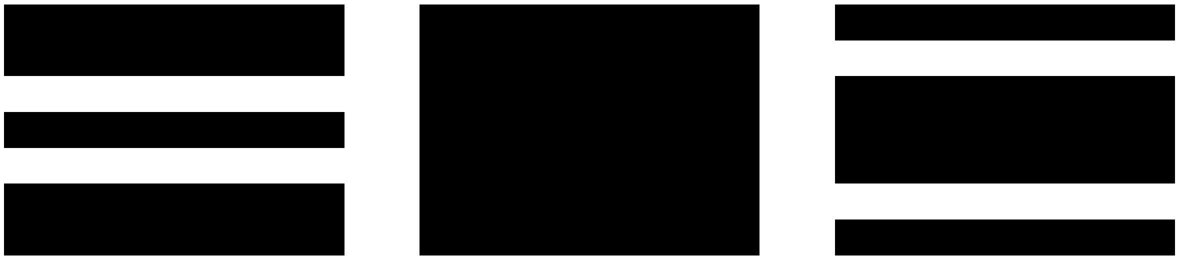
In this section, we will try to perform Sobel edge detection. In addition to finding the gradient in both horizontal and vertical directions, we also need to combine the two to get the final output that shows us all the edges. As shown below is the sample code to perform Sobel edge detection on a sample image shown in Figure 5(a).

```
img = np.zeros((7,7))                                          #Create an array with zeros.
img[2,:] = 255                                                 #Set row 2 to 255.
img[4,:] = 255                                                 #Set row 4 to 255.

sobel_hori = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=3)             #Sobel kernel (horizontal)
sobel_vert = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=3)             #Sobel kernel (vertical)

sobel_hori_mag = np.abs(sobel_hori)                            #Calculate the magnitude
sobel_vert_mag = np.abs(sobel_vert)                            #Calculate the magnitude

combined_mag = (0.5*sobel_hori_mag) + (0.5*sobel_vert_mag)     #Combined the two magnitude
```



(a) Sample Image (b) Sobel Output (Horizontal) (c) Sobel Output (Vertical)

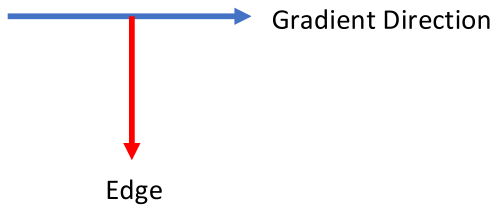
Figure 5

Instead of defining the Sobel kernels by ourselves, we can also use the cv2.Sobel() function from OpenCV. It can accept up to five parameters. The first two parameters are the input image and depth (or precision) respectively. In this case, we are using CV_64F that stands for 64 bits floating point format. It provides higher precision when compared to the 32 bits floating point format. We might not need such a higher precision but to be safe we will just use this to ensure the loss of precision is minimal. The following two parameters are used to set which Sobel kernel should be use (horizontal or vertical). If we put “1” and “0”, then the kernel used to find changes in the horizontal direction will be use. If we swap the “1” and “0”, then the kernel used to find changes in the vertical direction will be use. The last parameter is used to determine the size of the kernel. You may choose to use a kernel that is larger than 3x3 if needed.

The outputs after applying the two Sobel kernels are shown in Figure 5(b) and Figure 5(c) respectively. By looking at the two outputs, do you think they are the correct outputs? The answer to the previous question is yes. There is no issue with the output. We will not be able to see anything in the output when we applied the kernel to find the changes in the horizontal direction. If we look at the sample image again, we can see that there are no changes in the horizontal direction. In other words, we will not be able to find any slope when moving from left to right (or right to left). It is different when compared to the vertical direction, where we can find two slopes when moving from bottom to top (or top to bottom).

**Important Note**

Based on the previous experiment, we can see that the gradient direction is always normal to the edge.



**Exercise**

i. Use image "lena.bmp" as the sample image, try to get the gradient magnitude in both the horizontal and vertical directions by using the two Sobel kernels. Furthermore, try to combine the two outputs. You may also try to adjust the kernel size to see what you would get.
ii. Repeat (i) using image "noisy_lena.bmp" as the sample image. Compare the outputs with the those that you obtained from (i). Also, try to resolve any issue(s) you encountered when trying to find the edges.

## D. Canny Edge Detection

Different from Sobel edge detection, the Canny edge detection has multiple stages. We will be using the cv2.Canny() function from OpenCV to perform Canny edge detection. It first filters an image with a 5x5 Gaussian filter (based on the description given by OpenCV) to remove the effects of noise, before applying its kernel to find the derivatives. It also includes additional techniques such as non-maximum suppression and hysteresis thresholding to improve the performance in getting clear edges.

Using the cv2.Canny() function is straight forward. As shown below is the sample code of using the function. Please take note that we might also need to first filter the image on our own if we think that the 5x5 Gaussian filter is insufficient in reducing the noise presents in the image. The only challenge here is basically to determine the low and high thresholds that should be used. General, we may first find the mean intensity value of the image, $img_m$, then set the low threshold to be $0.5*img_m$ or $0.66*img_m$ and the high threshold to be $1.33*img_m$ or $1.5*img_m$, depends on whether you want to have a wider range or tighter range. A wider range would give you more edges when compared to a tighter range. Of course, you may also choose adjust the two thresholds manually to get the results that you want.

```
img = cv2.imread('lena.bmp',0)                  #Read the sample image

edges = cv2.Canny(img, 50,150)                  #Apply Canny edge detection

cv2.imshow("Edges",img)                         #Display the edges
cv2.waitKey()                                   #Wait for user to press any key
cv2.destroyAllWindows()                         #Close the display window
```

**<u>Exercise</u>**

i. Use image “lena.bmp” as the sample image, try to get the edges by using Canny edge detection. Compared the output with the version that you obtained using Sobel edge detection.
ii. Repeat (i) using image “noisy_lena.bmp” as the sample image. Compare the outputs with the those that you obtained from (i). Also, try to resolve any issue(s) you encountered when trying to find the edges.