

## Introduction

The following manual is an attempt at documenting the in and outs of the code written by Jai Samarth (<https://github.com/JaiSamarth>) during his internship at Soft Matter Biophysics Lab at IISER Mohali in the summer of 2022. This document details the instructions to use the code as well as the algorithmic approach used in it.

The implementation is divided into two parts. Part one deals with extraction of meaningful data from each image slice in the confocal data and part two deals with the process of visualization of this data.

## Part 1.1 : Extracting information from individual slices

### Analyze confocal data

1. Start by placing the python script "analyse\_confocal.py" and the acquired confocal data .czi file in the same directory. Open up a terminal with a python environment set up and the prerequisite packages (mentioned in the beginning of the script) installed. Navigate to the same directory.

```
\Softmatterlab> cd .\Confocal\  
\Softmatterlab\Confocal> python analyse_confocal.py
```

Figure 1: Run the script

2. Run the script by typing "python analyse\_confocal.py" in the terminal. A text prompt will be visible requesting to input the name of the file to be analyzed. Type the same and press enter.

```
Please enter the name of the file to be analysed below ...  
Z-STACK 6.czi
```

Figure 2: Enter file name

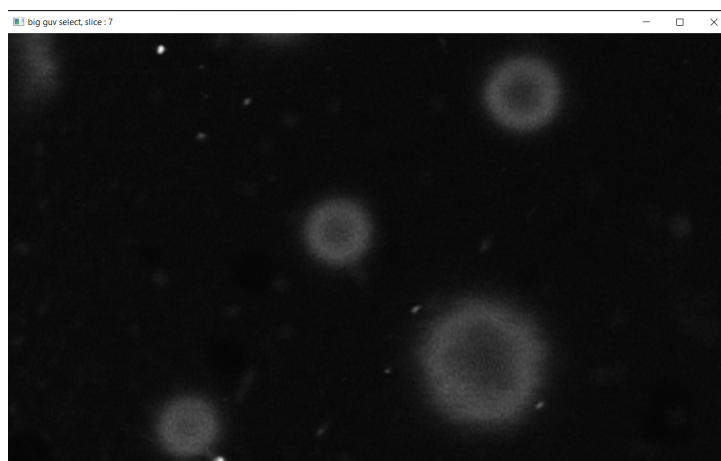


Figure 3: Image pop up window

```
Please enter the name of the file to be analysed below ...  
Z-STACK 6.czi  
  
Analysing big guv  
█
```

Figure 4: Capturing data for big GUV

3. First, data for the big GUV will be captured. Once the image window pops up, look for the big GUV you want to analyze. Next left click and drag over the interest area to draw a rectangle making sure to not include any unwanted structures for accurate results. Make sure a clear halo is visible. If the program is unable to find a halo, it will skip to the next image and produce an error message in the terminal.

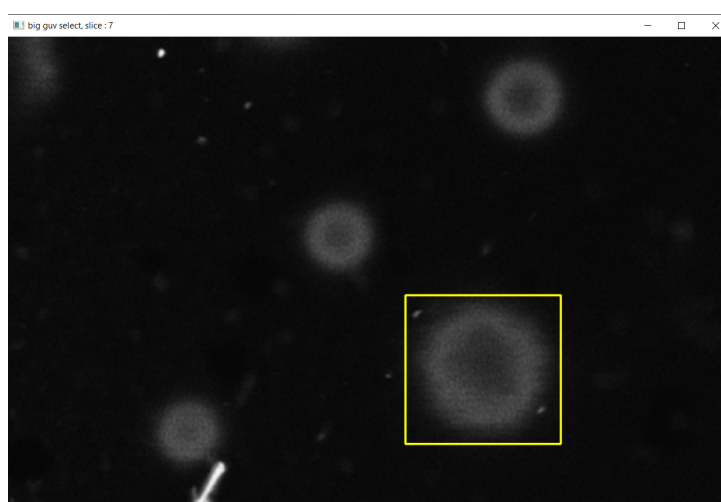


Figure 5: Draw rectangle over the area of interest

4. After the interest area is selected press the middle mouse button to preview the fitted ellipse. If not satisfied with the result step 3 may be repeated.

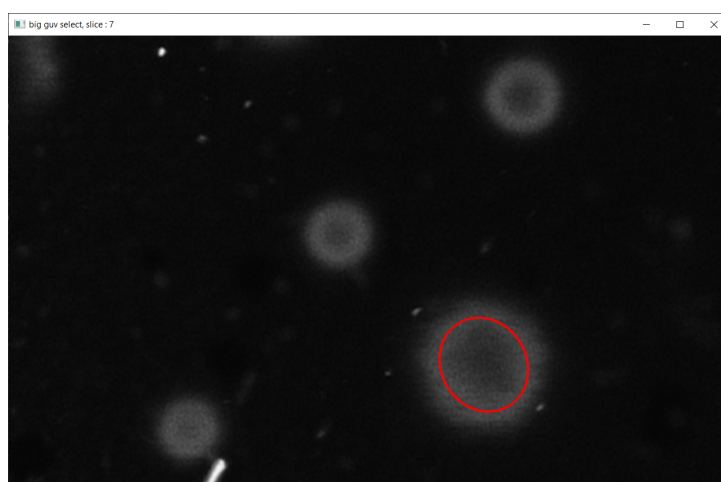
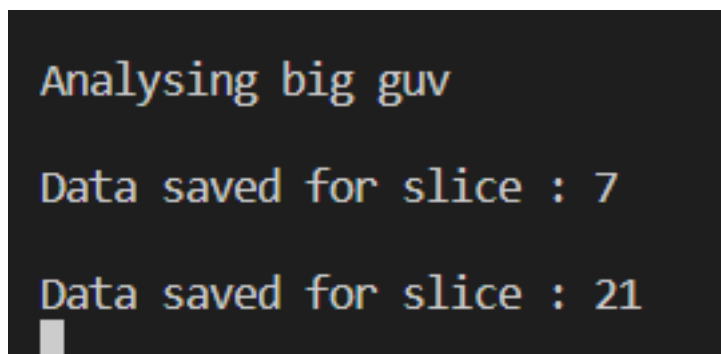


Figure 6: Fitted ellipse preview

5. If satisfied with the ellipse fit press the right mouse button and the data for that slice will be saved to a csv file for later viewing. A message in the terminal will also be visible along with the data saved in green text on the image. Make sure to press the right mouse button for saving data otherwise data for that slice will not be saved.



```
Analysing big guv
Data saved for slice : 7
Data saved for slice : 21
```

Figure 7: Messages in the terminal

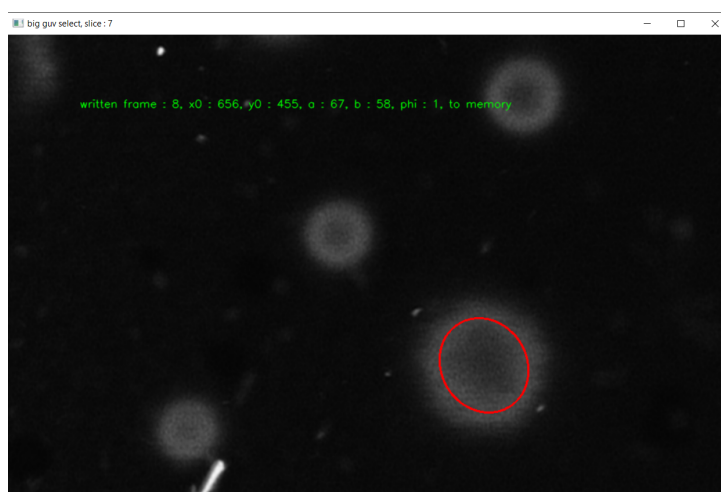
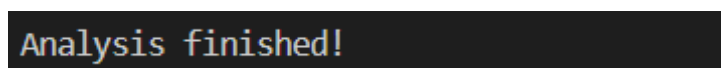


Figure 8: Fitted parameters in green text

6. Press the escape key to move onto the next image. After the program goes through all the images it will repeat once more to capture data for the small bud GUV. Upon completion, it will terminate and an appropriately named csv file can be accessed from the same directory to view the recorded data.



```
Analysis finished!
```

Figure 9: Program terminated

## Part 1.2 : Inside the workings of the program

### Reading in data

- Image data needs to be read from memory into arrays. For this we can either use ImageJ to convert .czi data into .bmp images and then read them using OpenCV's built-in imread function, or we can use a module known as aicspylibczi which is a python wrapper of the libCZI C/C++ library used for reading Zeiss CZI files. This code does the latter, but former is also easily implementable by using a *for* loop to iterate over all .bmp images.
- Each image is normalized to have pixel values between 0 and 255 and converted into uint8 format before further processing.

### Use of image filters

- After the area of interest has been selected, it goes through a series of image filters to facilitate peak detection.
- First we use CLAHE to enhance the contrast of the image. Then we use simple blur to smooth out any noise in the data. After that we apply Gaussian blur which smooths out intensity peaks in the image giving a much better estimate of the GUV membrane position. Finally, we use a threshold filter to set the values of all dim pixels to zero.

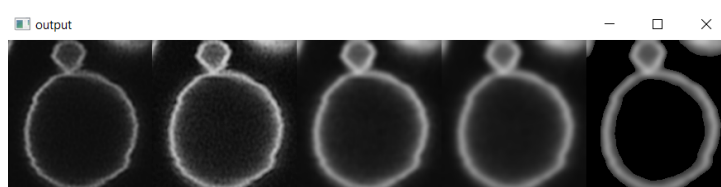


Figure 10: Successive image filters in order : original, CLAHE, blur, Gaussian & threshold

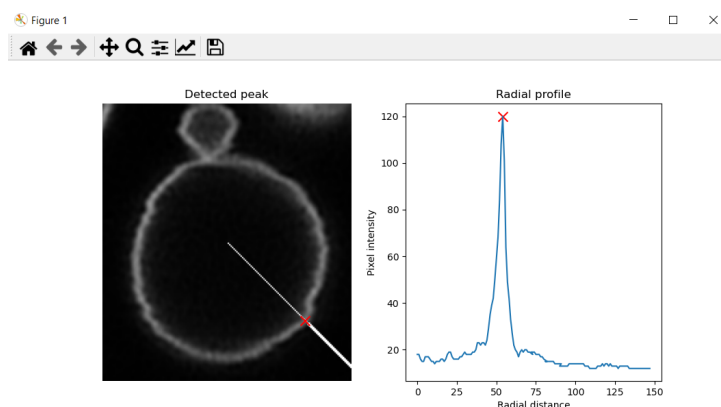


Figure 11: Visualization of peak detection algorithm

## Detection of peaks

The algorithm for detection of peaks can be described by the following steps in order.

1. Post-processed image is read pixel-by-pixel and angular coordinate of each pixel with non-zero intensity is stored in a list.
2. All pixels with a particular angle are sorted with respect to their radial distance from the center.
3. Now our problem is reduced to finding data peaks in this one dimensional data which can be done using scipy's *find\_peaks* function. Here it is important to specify suitable minimum width parameter for the peaks for accurate results since this one dimensional data will be littered with small peaks and we're only interested in the peak generated by the GUV membrane. This parameter can be found through trial & error.
4. Once the program scans through all the different angles, peak coordinates are stored in a list from where we can fit an ellipse over them.

## Fitting ellipse on data

- A python implementation of an algorithm proposed [1] and then improved in [2] is used to fit ellipses to radial peaks. It is highly recommended reading it for the explanation and proof of the algorithm.

[1] A. Fitzgibbon, M. Pilu and R. B. Fisher, "Direct least square fitting of ellipses," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 21, no. 5, pp. 476-480, May 1999, doi: 10.1109/34.765658.

[2] <http://autotrace.sourceforge.net/WSCG98.pdf>

- Also credits to Nikolai Chernov for publishing a Matlab implementation of this algorithm upon which this python implementation is based.

(<https://www.mathworks.com/matlabcentral/fileexchange/22684-ellipse-fit-direct-method>)

## Part 2.1 : Visualizing GUV data

### Visualize data

1. Start by opening the .czi file of confocal data in ZEN. The output from the previous script is in pixel units & we need to convert that into microns. Go to "Info" tab and look under "Image Dimensions", take note of "Scaling (per pixel)".

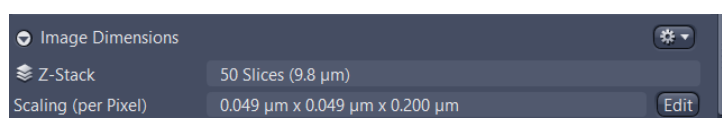


Figure 12: For this case, 1 pixel = 0.049  $\mu\text{m}$  and Z scaling is 0.2  $\mu\text{m}$

2. Edit the csv file and make two new columns labelled 'a (um)' and 'b (um)' with formula 'a/b (um)' = 'a/b' \* x/y scaling.

	A	B	C	D	E	F	G
1	Z-STACK 19.czi						
2	big guv						
3	frame	x0	y0	a	b	phi	
4	2	337	422	70.85784	51.12256	0.653158	
5	3	330	424	71.1613	55.95098	0.340895	

Figure 13: .csv file opened in Excel

	A	B	C	D	E	F	G	H	I
1	Z-STACK 19.czi								
2	big guv								
3	frame	x0	y0	a	b	phi	a (um)	b (um)	
4	2	337	422	70.85784	51.12256	0.653158	3.472034	2.505005	
5	3	330	424	71.1613	55.95098	0.340895	3.486903	2.741598	

Figure 14: Additional columns created

- Now the file is ready to be read for visualization. Start by placing the now updated .csv file and "guv\_stack.py" in the same directory. Open up a terminal with a python environment set up and prerequisite packages installed as mentioned in the beginning of the script. Navigate to that directory.



Figure 15: Run the script

- Run the script by typing "python guv\_stack.py" in the terminal. A text prompt will be visible requesting name of the .csv file to be visualized. Type the same and press enter.

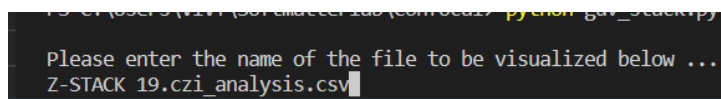


Figure 16: Enter file name

- A new window will open with an interactive 3d plot of the ellipses generated by the previous program. Close the window after viewing/saving the plots.
- Another prompt will appear in the terminal asking if the small GUV needs to be fitted or not. Here the decision is up to the user but due to lesser number of data points results will be inaccurate. Enter True or False as desired.
- A new window will open with an interactive 3d plot of the fitted ellipsoids. Close the window after viewing/saving the plots and the program will be terminated.

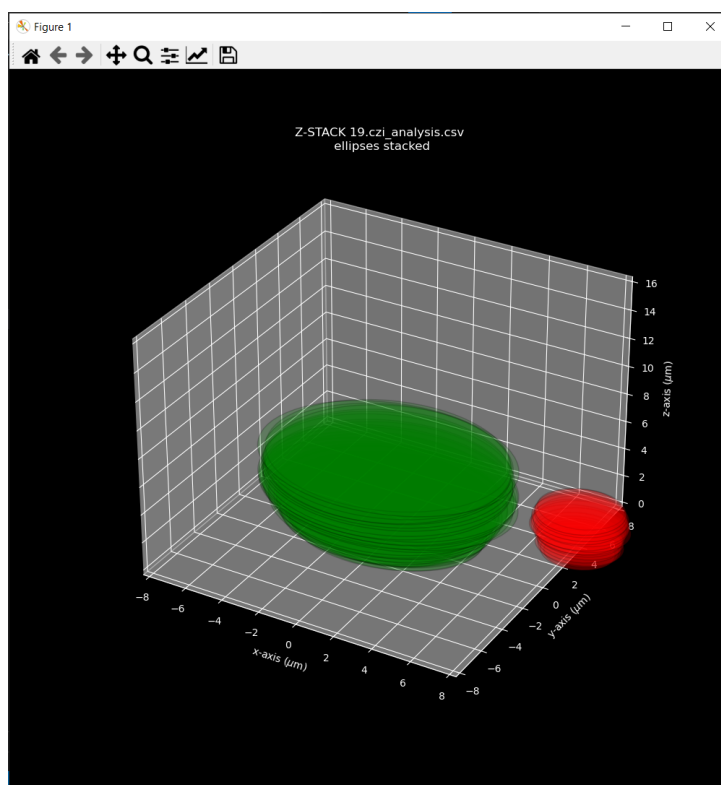


Figure 17: Preview of fitted ellipses stacked upon each other

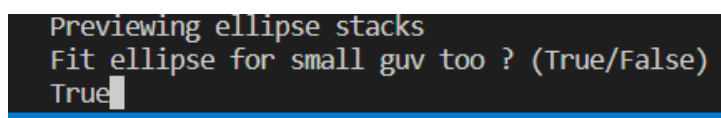


Figure 18: Choose if ellipsoid should be fitted for small GUV or not

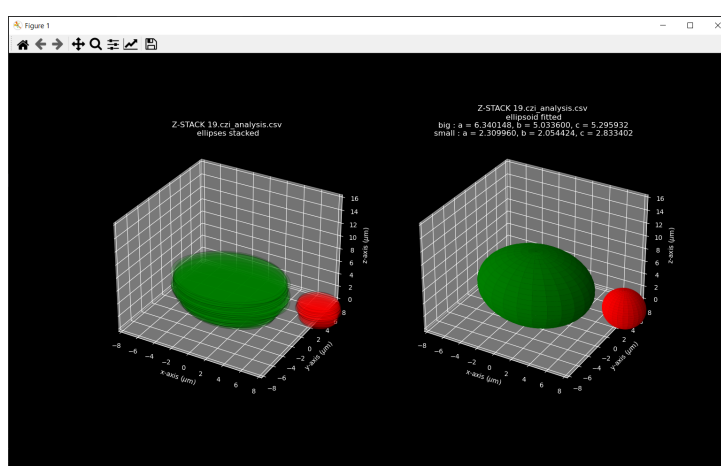


Figure 19: Fitted ellipsoids

## Part 2.2 : Inside the workings of the program

### Fitting ellipsoid on data

- A python implementation of the algorithm proposed in this paper is used for fitting ellipsoids to acquired data. It is highly recommended reading it for the explanation and proof of the algorithm.

Qingde Li and J. G. Griffiths, "Least squares ellipsoid specific fitting," Geometric Modeling and Processing, 2004. Proceedings, 2004, pp. 335-340, doi: 10.1109/GMAP.2004.1290055.

- Here we take in the major and minor axes fits obtained in the previous program and generate new sets of points to obtain a 3d cloud upon which the ellipsoid is fitted. Rotation of individual ellipses in their planes is not accounted for and all principal axes are aligned with each other.

— \* \* \* \* \*



## Appendix

### Code : analyse\_confocal.py

```
# python script for analysing guvs from confocal microscopy data
# this script takes in czi files in 'TCZYX' format and fits an
# ellipse on guvs in a semi-automatic fashion using user input
# to create a bounding box around the interest area. make sure
# necessary modules are installed. data will be saved in a csv
# file with appropriate name. -jsama

#importing modules
import numpy as np
import matplotlib.pyplot as plt
import cv2
from aicspylibczi import CziFile
from scipy.signal import find_peaks
from scipy import linalg
#input file name
file_name = input('\nPlease enter the name of the file to be \
analysed below ...\n')
#read czi file
czi = CziFile(file_name)
#reject if wrong format (must be 'TCZYX' and monochromatic)
if czi.dims != 'TCZYX' or czi.get_dims_shape()[0]['C'][1] != 1:
    print('\nFile format incorrect!\nformat must be TCZYX')
    exit()
#file to save data
data_file = open(file_name + '_analysis.csv', 'w')
data_file.write(file_name)
#define functions
#clahe object for filter
clahe = cv2.createCLAHE(clipLimit=2, tileGridSize=(3,3))
def apply_filters(array):
    #clahe to enhance contrast
    array = clahe.apply(array)
    #blur to smooth noise
    array = cv2.blur(array, ksize=(7,7))
    #gaussian for smoother peaks
    array = cv2.GaussianBlur(array, ksize=(7,7), sigmaX=5, sigmaY=5)
    #threshold to reject dim pixels
    _, array = cv2.threshold(array,55,255, cv2.THRESH_TOZERO)
    return array
def detect_peaks(image_array):
    #image center
    xi = image_array.shape[1]//2
    yi = image_array.shape[0]//2
    pixel_id = []
    #id each pixel
    for j in range(image_array.shape[0]): #scan down
        for i in range(image_array.shape[1]): #scan accross
```

```

        if image_array[j,i] != 0: #intensity not zero
            theta = np.angle((i - xi) + 1j*(j - yi)) #get angle
            #store coords, angle, intensity
            pixel_id.append(np.array([i, j,\
                                     int(round(np.rad2deg(theta),0)), image_array[j,i])))
#convert to array
pixel_id = np.array(pixel_id)
peak_vl = [] #store peaks
for i in np.arange(-180,181,1): #scan angles
    pixel_angle = pixel_id[pixel_id[:,2] == i,:] #at this particular angle
    #sort data with radial distance
    pixel_angle = pixel_angle[np.argsort(\
        np.sqrt((pixel_angle[:,1] - yi)**2 + (pixel_angle[:,0] - xi)**2))]
    #find signal peak
    peak_id, _ = find_peaks(pixel_angle[:,3], width=4)
    #store first peak coords, intensity
    if peak_id.size != 0 :
        peak = pixel_angle[peak_id[0]]
        peak_vl.append(np.array([peak[0],peak[1],peak[3]]))
#output peak coords and intensity
return np.array(peak_vl, dtype=int)
def direct_ellipse(xy):
    #direct implimentation of algorithm
    centroid = np.mean(xy, axis = 0)
    D1 = np.vstack([(xy[:,0]-centroid[0])**2,
                    (xy[:,0]-centroid[0])*(xy[:,1]-centroid[1]),
                    (xy[:,1]-centroid[1])**2]).T
    D2 = np.vstack([(xy[:,0]-centroid[0]),
                    (xy[:,1]-centroid[1]),
                    np.ones(xy.shape[0])]).T
    S1 = D1.T @ D1
    S2 = D1.T @ D2
    S3 = D2.T @ D2
    T = -np.linalg.inv(S3) @ S2.T
    M = S1 + S2 @ T
    C = np.array(((0, 0, 2), (0, -1, 0), (2, 0, 0)), dtype=float)
    M = np.linalg.inv(C) @ M
    _, eigvec = np.linalg.eig(M)
    con = 4 * eigvec[0]* eigvec[2] - eigvec[1]**2
    a1 = eigvec[:, np.nonzero(con > 0)[0]]
    A = np.concatenate((a1, T @ a1)).ravel()
    A3 = A[3]-2*A[0]*centroid[0]-A[1]*centroid[1]
    A4 = A[4]-2*A[2]*centroid[1]-A[1]*centroid[0]
    A5 = A[5]+A[0]*centroid[0]**2+A[2]*centroid[1]**2\
        +A[1]*centroid[0]*centroid[1]-A[3]*centroid[0]\
        -A[4]*centroid[1]
    A[3] = A3
    A[4] = A4
    A[5] = A5
    A = A/np.linalg.norm(A)
    #convert algebric parameters to polar

```

```

a = A[0]
b = A[1] / 2
c = A[2]
d = A[3] / 2
f = A[4] / 2
g = A[5]
den = b**2 - a*c
x0, y0 = (c*d - b*f) / den, (a*f - b*d) / den
num = 2 * (a*f**2 + c*d**2 + g*b**2 - 2*b*d*f - a*c*g)
fac = np.sqrt((a - c)**2 + 4*b**2)
ap = np.sqrt(num / den / (fac - a - c))
bp = np.sqrt(num / den / (-fac - a - c))
width_gt_height = True
if ap < bp:
    width_gt_height = False
    ap, bp = bp, ap
if b == 0:
    phi = 0 if a < c else np.pi/2
else:
    phi = np.arctan((2.*b) / (a - c)) / 2
    if a > c:
        phi += np.pi/2
if not width_gt_height:
    phi += np.pi/2
phi = phi % np.pi
return x0, y0, ap, bp, phi
#main callback function
def main(event, x, y, flags, params):
    #global variables
    global ix, iy, jx, jy, draw, imgv, imgs, z, x0, y0, a, b, phi
    #draw rectangle
    if event == cv2.EVENT_LBUTTONDOWN:
        draw = True
        ix = x
        iy = y
    elif event == cv2.EVENT_MOUSEMOVE:
        if draw == True:
            imgv = imgc.copy()
            cv2.rectangle(imgv, pt1=(ix,iy), pt2=(x,y), color=(0,255,255),
                           thickness=2)
    elif event == cv2.EVENT_LBUTTONUP:
        draw = False
        imgv = imgc.copy()
        cv2.rectangle(imgv, pt1=(ix,iy), pt2=(x,y), color=(0,255,255),
                       thickness=2)
        jx, jy = x, y
    #draw elliptical fit
    elif event == cv2.EVENT_MBUTTONDOWN:
        imgv = imgc.copy()
        ix,jx = sorted((ix,jx))
        iy,jy = sorted((iy,jy))

```

```

peak_vl = detect_peaks(apply_filters(imgs[iy:jy,ix:jx]))
if peak_vl.shape[0] != 0:
    x = peak_vl[:,0]
    y = peak_vl[:,1]
    x0, y0, a, b, phi = direct_ellipse(np.vstack((x,y)).T)
    x0 += ix
    y0 += iy
    imgv = cv2.ellipse(imgv, (int(round(x0,0)),
                                int(round(y0,0))),
                        (int(round(a,0)),
                         int(round(b,0))),
                        np.rad2deg(phi),
                        0,
                        360,
                        (0,0,255), 2)

#save data
elif event == cv2.EVENT_RBUTTONDOWN:
    imgv = cv2.putText(imgv,
        "written frame : %i, x0 : %i, y0 : %i, a : %i, b : %i, phi : %i,\
        to memory"%(z+1, x0, y0, a, b, phi),
        (X//10,Y//10),
        cv2.FONT_HERSHEY_SIMPLEX,
        0.5,
        (0,255,0),
        1,
        cv2.LINE_AA)
    print('\nData saved for slice : ' + str(z))
    data_file.write("\n%i,%i,%i,%f,%f,%f"%(z+1, x0, y0, a, b, phi))

#parameters
shape = czi.get_dims_shape()[0]
Z = shape['Z'][1]
X = shape['X'][1]
Y = shape['Y'][1]
#start analysis
#write big guv data
print('\nAnalysing big guv')
data_file.write('\nbig guv')
data_file.write('\nframe,x0,y0,a,b,phi')
#iterate different frames
for z in range(Z):
    #syntax to read files
    img, _ = czi.read_image(Z = z)
    img = img[0,0,0,:,:]
    #make bitdepth 8bit
    imgs = np.uint8(cv2.normalize(img, dst=None, alpha=0, beta=255,
                                norm_type=cv2.NORM_MINMAX))
    imgc = cv2.cvtColor(imgs, cv2.COLOR_GRAY2BGR)
    #initialize rectangle variables
    ix = -1
    iy = -1
    jx = -1

```

```
jy = -1
#initialize draw variable
draw = False
#copy image to display
imgv = imgc.copy()
#name window
window_name = 'big guv select, slice : ' + str(z)
cv2.namedWindow(window_name)
#set callback
cv2.setMouseCallback(window_name, main)
#wait for escape key the kill windows
while True:
    cv2.imshow(window_name, imgv)
    if cv2.waitKey(10) == 27:
        break
    cv2.destroyAllWindows()
#write small guv data
#comment out if not required
print('\nAnalysing small guv')
data_file.write('\nsmall guv')
data_file.write('\nframe,x0,y0,a,b,phi')
#iterate different frames
for z in range(Z):
    #syntax to read files
    img, _ = czi.read_image(Z = z)
    img = img[0,0,0,:,:]
    #make bitdepth 8bit
    imgs = np.uint8(cv2.normalize(img, dst=None, alpha=0, beta=255,
                                norm_type=cv2.NORM_MINMAX))
    imgc = cv2.cvtColor(imgs, cv2.COLOR_GRAY2BGR)
    #initialize rectangle variables
    ix = -1
    iy = -1
    jx = -1
    jy = -1
    #initialize draw variable
    draw = False
    #copy image to display
    imgv = imgc.copy()
    #name window
    window_name = 'small guv select, slice : ' + str(z)
    cv2.namedWindow(window_name)
    #set callback
    cv2.setMouseCallback(window_name, main)
    #wait for escape key the kill windows
    while True:
        cv2.imshow(window_name, imgv)
        if cv2.waitKey(10) == 27:
            break
        cv2.destroyAllWindows()
#end program
```

```
data_file.close()
print('\nAnalysis finished!')
```

## Code : guv\_stack.py

```
# python script for visualizing data generated by
# analyse_confocal.py. make sure necessary modules
# are installed. -jsama

#importing modules
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.patches import Ellipse
import mpl_toolkits.mplot3d.art3d as art3d
from soupsieve import match

#input file name
file_name = input('\nPlease enter the name of the file to be \
visualized below ... \n')

#read csv file into dataframe
df = pd.read_csv(file_name, header=None)
split_at = df[df[0] == 'small guv'].index[0]
big = df.iloc[3:split_at].reset_index(drop = True)
small = df.iloc[split_at+2:].reset_index(drop = True)
big.rename(columns=df.iloc[2], inplace = True)
small.rename(columns=df.iloc[2], inplace = True)
#convert datatype
big = big.astype(np.float64)
small = small.astype(np.float64)

#defining functions
def direct_ellipsoid(x_val, y_val, z_val):
    #direct implementation of algorithm
    D = np.vstack([x_val*x_val,
                    y_val*y_val,
                    z_val*z_val,
                    2*y_val*z_val,
                    2*x_val*z_val,
                    2*x_val*y_val,
                    2*x_val,
                    2*y_val,
                    2*z_val,
                    np.ones_like(x_val)])
    S = D @ D.T
    C = np.zeros((6,6))
    C[0,0] = -1
    C[1,1] = -1
    C[2,2] = -1
```

```

C[3,3] = -4
C[4,4] = -4
C[5,5] = -4
C[0,1] = 1
C[1,0] = 1
C[0,2] = 1
C[2,0] = 1
C[1,2] = 1
C[2,1] = 1
S11 = S[0:6,0:6]
S12 = S[0:6,6:10]
S22 = S[6:10,6:10]
A = S11 - S12 @ np.linalg.pinv(S22) @ S12.T
CA = np.linalg.inv(C) @ A
[eval, evec] = np.linalg.eig(CA)
v1=evec[:, np.argmax(eval)]
v2=-np.linalg.pinv(S22) @ S12.T @ v1
#approximating ellipsoid to be aligned
#along cartesian axis
a,b,c,_,_,_,p,q,r,d = np.hstack([v1, v2])
x0 = -p/a
y0 = -q/b
z0 = -r/c
ABCSQ = p**2/a + q**2/b + r**2/c - d
A0 = (ABCSQ/a)**0.5
B0 = (ABCSQ/b)**0.5
C0 = (ABCSQ/c)**0.5
return x0, y0, z0, A0, B0, C0
#this function takes in ellipse parameters and
#returns x array, y array of ellipse
def ellipse_cart(x0,y0,a,b):
    theta = np.linspace(0,2*np.pi,360)
    x = x0 + a*np.cos(theta)
    y = y0 + b*np.sin(theta)
    return x,y
#this function takes in ellipsoid parameters
#and returns x array, y array, z array of ellipsoid
def ellipsoid_cart(x0, y0, z0, rx, ry, rz):
    u = np.linspace(0, 2 * np.pi, 100)
    v = np.linspace(0, np.pi, 100)
    x = x0 + rx * np.outer(np.cos(u), np.sin(v))
    y = y0 + ry * np.outer(np.sin(u), np.sin(v))
    z = z0 + rz * np.outer(np.ones_like(u), np.cos(v))
    return x,y,z
#coordinates of small guv
def small_coord(df1, df2):
    conv = df2.iloc[0]['a (um)']/df2.iloc[0]['a']
    xs = []
    ys = []
    for _, row in df2.iterrows():
        frame, x0, y0 = row[0:3]

```

```

        x1, y1= df1.iloc[np.argmin(np.abs(df1['frame'] - frame))][1:3]
        xs.append((x0-x1)*conv)
        ys.append((y0-y1)*conv)
    return np.mean(xs), np.mean(ys)
#set axis
#change these parameters
#to set bounding box of figure
def set_axis(axis):
    axis.set_xlabel(r'x-axis ( $\mu\text{m}$ )')
    axis.set_ylabel(r'y-axis ( $\mu\text{m}$ )')
    axis.set_zlabel(r'z-axis ( $\mu\text{m}$ )')
    axis.set_xlim(-8,8)
    axis.set_ylim(-8,8)
    axis.set_zlim(0,16)
    return
#plot big guv
def plot_big(axis, df1):
    for _, row in df1.iterrows():
        frame, _, _, _, _, a, b = row
        e = Ellipse((0,0), 2*a, 2*b, 0, alpha = 0.2, facecolor = 'green',
                    edgecolor = 'k')
        axis.add_patch(e)
        art3d.pathpatch_2d_to_3d(e,
                                z=(frame-min(df1['frame']))*0.2, # <- scaling in Z
                                zdir="z")
    return
#plot small guv
def plot_small(axis, df1, df2):
    xs, ys = small_coord(df1,df2)
    for _, row in df2.iterrows():
        frame, _, _, _, _, a, b = row
        e = Ellipse((xs, ys), 2*a, 2*b, 0, alpha = 0.2, facecolor = 'red',
                    edgecolor = 'k')
        axis.add_patch(e)
        art3d.pathpatch_2d_to_3d(e,
                                z=(frame-min(df1['frame']))*0.2, # <- scaling in Z
                                zdir="z")
    return
#ellipsoid sample points
def get_3d_sample_points_big(df1):
    x_val = np.array([])
    y_val = np.array([])
    z_val = np.array([])
    for _, row in df1.iterrows():
        frame, _, _, _, _, a, b = row
        x, y = ellipse_cart(0,0,a,b)
        x_val = np.hstack([x_val,x])
        y_val = np.hstack([y_val,y])
        z_val = np.hstack([z_val,
                           np.ones_like(x)*(frame - min(df1['frame']))*0.2 # <- scaling in Z
                           ])

```



```

    return x_val, y_val, z_val
def get_3d_sample_points_small(df1, df2):
    xs, ys = small_coord(df1, df2)
    x_val = np.array([])
    y_val = np.array([])
    z_val = np.array([])
    for _, row in df2.iterrows():
        frame, _, _, _, _, a, b = row
        x, y = ellipse_cart(xs,ys,a,b)
        x_val = np.hstack([x_val,x])
        y_val = np.hstack([y_val,y])
        z_val = np.hstack([z_val,
            np.ones_like(x)*(frame - min(df1['frame']))*0.2 # <- scaling in Z
        ])
    return x_val, y_val, z_val
#plot fitted ellipsoids
def plot_big_ellipsoid(axis, df1):
    x_val, y_val, z_val = get_3d_sample_points_big(df1)
    x0, y0, z0, a, b, c = direct_ellipsoid(x_val, y_val, z_val)
    x, y, z = ellipsoid_cart(x0, y0, z0, a, b, c)
    axis.plot_surface(x, y, z, cstride = 4, rstride = 4, color = 'green',
        alpha = 1)
    axis.set_title(axis.get_title() + '\n big : a = %f, b = %f, c = %f'%(a,
        b,c))
    return
def plot_small_ellipsoid(axis, df1, df2):
    x_val, y_val, z_val = get_3d_sample_points_small(df1, df2)
    x0, y0, z0, a, b, c = direct_ellipsoid(x_val, y_val, z_val)
    x, y, z = ellipsoid_cart(x0, y0, z0, a, b, c)
    axis.plot_surface(x, y, z, cstride = 4, rstride = 4, color = 'red',
        alpha = 1)
    axis.set_title(axis.get_title() + '\n small : a = %f, b = %f, c = %f'%(a,
        b,c))
    return
#main
#first preview 3d slices
#dark background
print('Previewing ellipse stacks')
plt.style.use('dark_background')
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection = '3d')
ax.set_title(file_name + '\n ellipses stacked')
plot_big(ax, big)
plot_small(ax, big, small)
set_axis(ax)
plt.show()
#prompt for small guv
if(input('Fit ellipse for small guv too ? (True/False)\n') == 'True'):
    #fit ellipsoid for small guv too
    print('\n Previewing fitted ellipsoid')
    fig = plt.figure(figsize=(10,10))

```

```
ax1 = fig.add_subplot(121, projection = '3d')
ax1.set_title(file_name + '\n ellipses stacked')
ax2 = fig.add_subplot(122, projection = '3d')
ax2.set_title(file_name + '\n ellipsoid fitted')
plot_big(ax1, big)
plot_small(ax1, big, small)
plot_big_ellipsoid(ax2, big)
plot_small_ellipsoid(ax2, big, small)
set_axis(ax1)
set_axis(ax2)
plt.show()
else:
    #fit ellipsoid for only big guv
    print('Previewing fitted ellipsoid')
    fig = plt.figure(figsize=(10,10))
    ax1 = fig.add_subplot(121, projection = '3d')
    ax1.set_title(file_name + '\n ellipses stacked')
    ax2 = fig.add_subplot(122, projection = '3d')
    ax2.set_title(file_name + '\n ellipsoid fitted')
    plot_big(ax1, big)
    plot_small(ax1, big, small)
    plot_small(ax2, big, small)
    plot_big_ellipsoid(ax2, big)
    set_axis(ax1)
    set_axis(ax2)
    plt.show()
```