PNAS
www.pnas.org

Supplementary Information for

4D Electron Microscopy of T-Cell Activation

Yue Lu[1,2,†], Byung-Kuk Yoo[1,†,*], Alphonsus H.C. Ng[1,2], Jungwoo Kim[1], Sinchul Yeom[1,3], Jau Tang[4], Milo M. Lin[5], Ahmed H. Zewail[1,‡], and James R. Heath[1,2,*]

[1] The Arthur Amos Noyes Laboratory of Chemical Physics, Division of Chemistry and Chemical Engineering, California Institute of Technology, Pasadena, CA 91125, USA.

[2] Institute for Systems Biology, Seattle, WA 98109, USA.

[3] Mechanical, Aerospace and Biomedical Engineering Department, University of Tennessee, Knoxville, TN 37996, USA.

[4] The Institute for Technological Sciences, Wuhan University, China.

[5] Green Center for Systems Biology, U T Southwestern Medical Center, Dallas, TX 75390, USA.

† These authors contributed equally to this work.

‡ Deceased August 2nd, 2016.

* Corresponding Authors
E-mail: jheath@systemsbiology.org (J.R.H.); bkyoo@caltech.edu (B.Y.).

This PDF file includes:

Figures S1 to S6
Supplementary text

www.pnas.org/cgi/doi/10.1073/pnas.1914078116

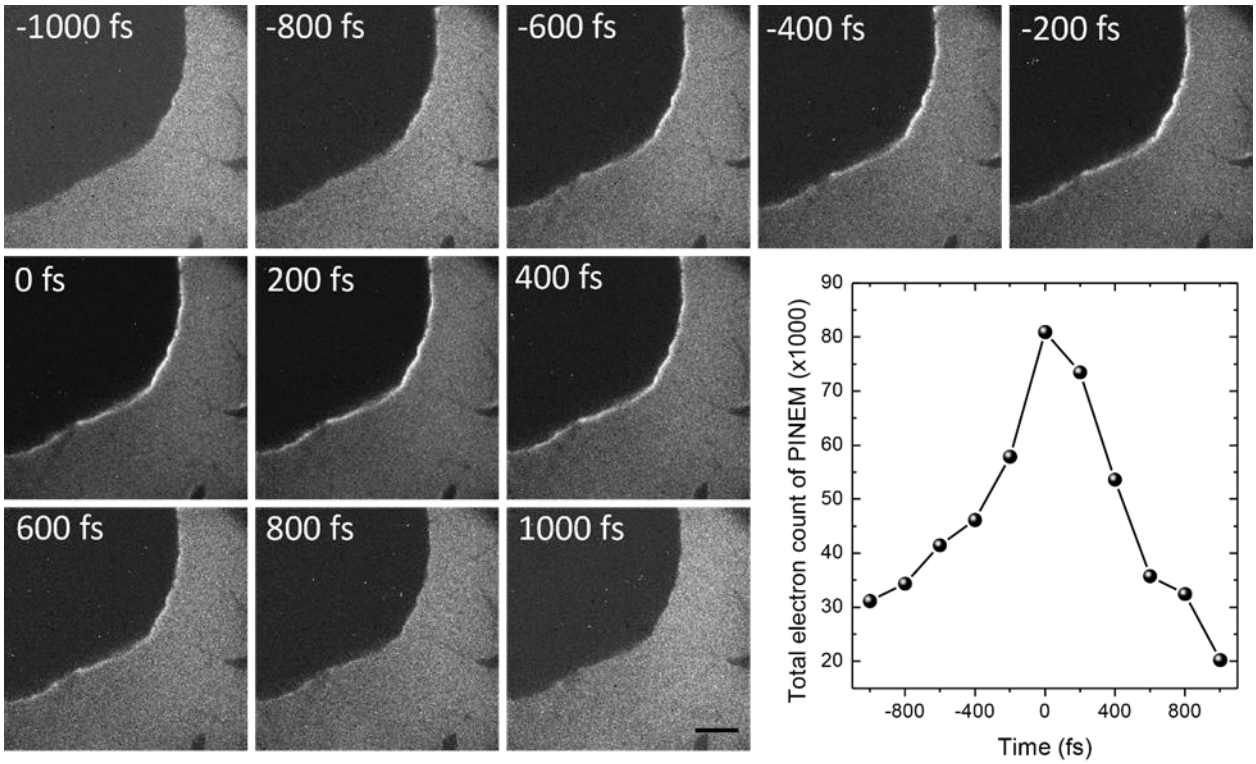

Fig. S1. Time-dependency of PINEM as a function of the overlap between the laser fs pulse and the electron pulse on the cell surface at given time delays. At time zero (0 fs), when there is a complete overlap between the laser and the electron pulse, PINEM intensity is the strongest. The intensity of PINEM decreases as the time difference between the two pulses increases. A plot represents the changes of PINEM intensity calculated from total electron counts of bright-field image data at given time delays.

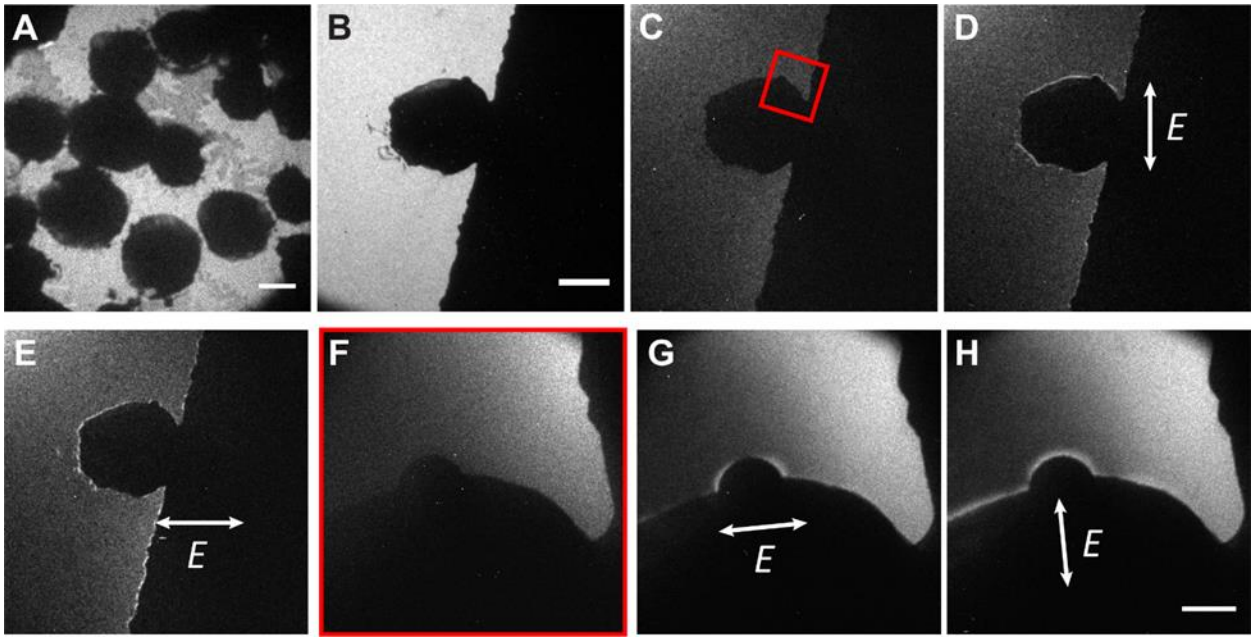

Fig. S2. (A) Bright-field image of cells at the resting state without activators at low magnification. Scale bar: 5 μm. (B) Closer look at one chosen cell at higher magnification, Scale bar: 5 μm. (C and F) Same cell but in UEM mode of detection with fs laser. (F) is a higher magnification image of the red square region in (C). (D-E) PINEM images of a cell at low magnification, generated with the fs laser linearly polarized in a plane indicated by the double-headed arrows. Scale bar: 5 μm. (G and H) PINEM images of a cell at higher magnification. Scale bar: 1 μm.

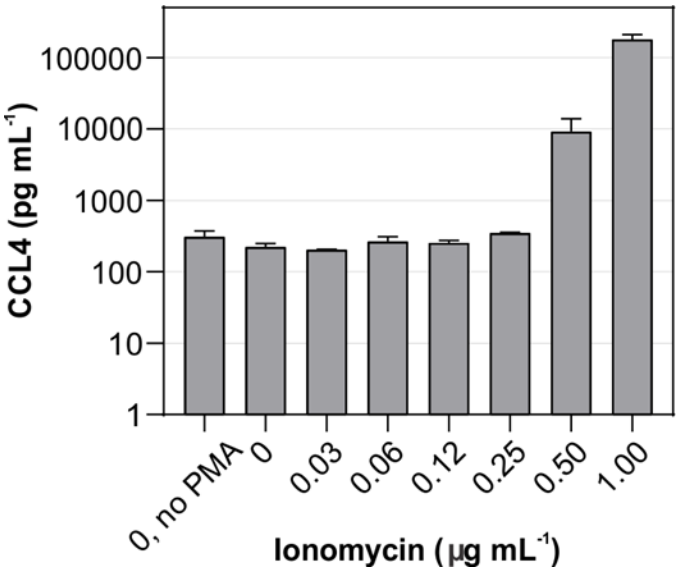

Fig. S3. CCL4 secretion of cells treated with Ionomycin and PMA measured by ELISA.

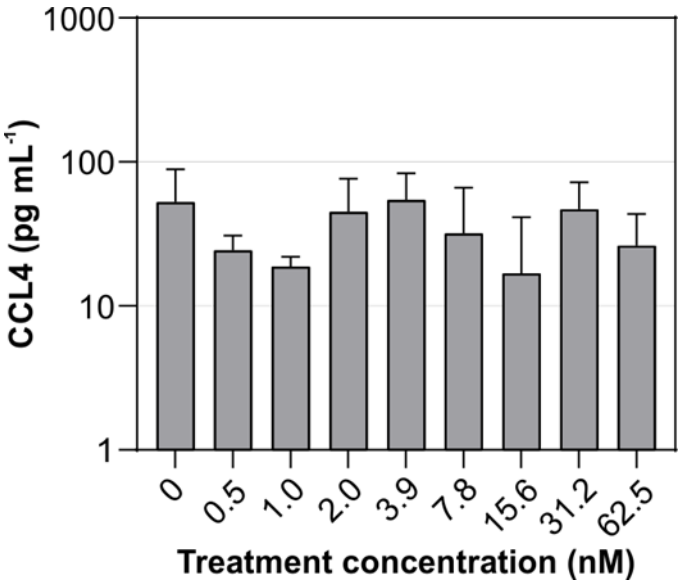

Fig. S4. CCL4 secretion of cells treated with MART-1 tetramer measured by ELISA.

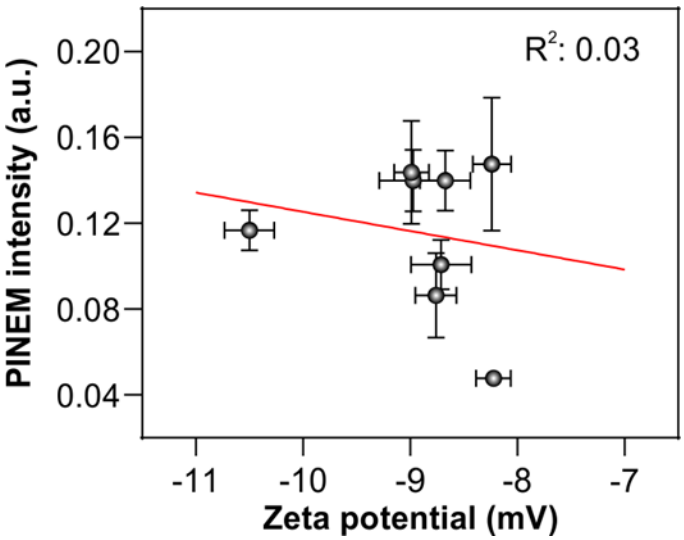

Fig. S5. No correlation between PINEM intensity and membrane potential was found in cells treated with MART-1 tetramer.

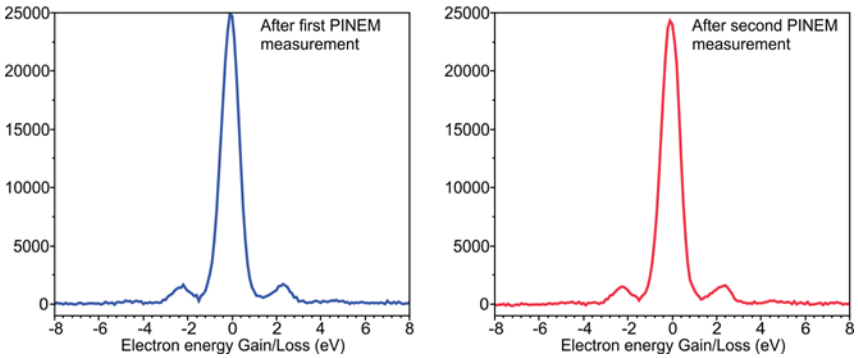

Fig. S6. Electron energy spectra of a Jurkat T cell after one (blue) or two (red) PINEM measurements.

Supplementary Information Text

Extended methods for PINEM image analysis. To investigate the temporal and polarization dependence of PINEM, PINEM images were analyzed with a pixel-by-pixel approach. For the cell contour analysis shown in Fig. 4A and 4B, PINEM intensity is obtained along a trace of the edge of a cell with respect to an angle. The angle was measured between two lines: a line connecting a pixel in the trace and a fixed cell center and the horizontal line of the image. To quantify the heterogeneity of intensity profile, PINEM intensity profile data were Fourier-transformed.
Python code for PINEM intensity analysis:

```
#!/usr/bin/env python3

'''
  PINEM data process script

 * Written by
   Sinchul Yeom (syeom@vols.utk.edu)
   Mechanical, Aerospace and Biomedical Engineering Department
   University of Tennessee, Knoxville

  last update : 6/10/2019

  Tested under following module verions.
  (Maybe working with other versions, too)
    image==1.5.16
    matplotlib==2.1.0
    numpy==1.13.3
    scipy==1.0.0
    Shapely==1.6.2.post1
'''

import argparse
import os, sys, glob
import numpy as np
import matplotlib as mpl
import statistics
from collections import OrderedDict
mpl.use('Agg')
import matplotlib.pyplot as plt
from matplotlib.ticker import (MultipleLocator,
FormatStrFormatter,
                               AutoMinorLocator)
import re, math

import warnings
warnings.simplefilter('ignore', np.RankWarning)
np.seterr(divide='ignore', invalid='ignore')

from PIL import Image
from pathlib import Path
from shapely.geometry import Point
from shapely.geometry import Polygon

from timeit import default_timer as timer

header = OrderedDict(Dist='Distance(um)',
```

```
            Angle='Angle(deg)',
            NormBr='NormBr(a.u.)',
            NormCosBr='NormCos.Br(a.u.)',
            FFreq='FFT freq(1/deg)',
            FFT='|FFT(NormBr)|',
            b0='',
            Br='Brightness(raw)',
            NCos='NCosine(rad)',
            CosBr='Cos.Br',
            NTan='NTangent(rad)',
            )

Num_normal_pixels=10
Num_norm_fit_pixels=3

mode_to_bpp = {'1':1, 'L':8, 'P':8, 'RGB':24, 'RGBA':32,
'CMYK':32, 'YCbCr':24, 'I':32,  'F':32, 'I;16B':16}

def printf(format, *args):
    print(format % args)

def dist_calc(p0,p1, res=None):
    diff=np.array(p0)-np.array(p1)
    if (res): diff=np.divide(diff,res)
    ret=np.linalg.norm(diff)
    return ret

def adj_color(x):
    ret = 0
    if (x > 255): ret = 1
    elif (x > 0): ret=x/255
    return (ret,)*3

def update_dic(arr, key, val):
    lst=arr.get(key, None)
    if (not lst):
        lst=[]
        arr.update( { key : lst })
    lst.append(val)
    return lst


# FFT plotting
def fft_process(title, fpre, darr):
    cut_a=args.fa[0]
    cut_b=args.fb[0]

    signal=np.array(darr.get('NormBr'))
    detrend_array=signal-signal.mean()
    time_angle=darr.get('Angle')

    # assumes equally spaced angles
    n_signal=len(signal)
    n_pos=(n_signal//2)
    freq=np.fft.fftfreq(n_signal, (time_angle[-1]-
time_angle[0])/(n_signal-1))
    freq_pos=freq[:n_pos]
```

```
    unit=freq[1]
    fft_detrend=abs(np.fft.fft(detrend_array))/n_signal

    icut_a = np.abs(freq_pos - cut_a).argmin()
    icut_b = np.abs(freq_pos - cut_b).argmin()

    sum=fft_detrend[:icut_b].sum()
    fft_detrend=fft_detrend/(sum*unit)

    sum_up=(fft_detrend[:icut_a].sum())*unit
    sum_down=fft_detrend[icut_a:icut_b].sum()*unit
    #print(sum_up, sum_down, sum_up+sum_down)
    darr.update( { 'FFreq' : freq, 'FFT' : fft_detrend })
    #plt.plot(freq,fft, label='Raw')
    fig, ax = plt.subplots()
    annot="{:.2f}:{:.2f}".format(sum_up,sum_down)
    plt.plot(freq[:n_pos],fft_detrend[:n_pos])
    plt.axvline(cut_a, linestyle='--', linewidth='1',
color='black')
    plt.axvline(cut_b, linestyle='-', linewidth='1.5',
color='black')
    #plt.legend()

    ofile=Path('.').joinpath('fft_'+fpre+"-cut{:.2f}-
{:.2f}.png".format(cut_a,cut_b))
    ax.set_title(ofile.stem)
    #ax.set_ylim([0,0.03])
    ax.minorticks_on()
    ax.grid(which='major', linestyle='-', linewidth='0.5',
color='red')
    ax.grid(which='minor', linestyle=':', linewidth='0.5',
color='black')
    ax.tick_params(which='both', # Options for both major and
minor ticks
                   top='off', # turn off top ticks
                   left='off', # turn off left ticks
                   right='off',  # turn off right ticks
                   bottom='off') # turn off bottom ticks

    t = ax.text(cut_a, max(fft_detrend)//2, annot, ha="center",
va="center", size=15)
    ax.set_xlabel('1/Angle (1/deg)')
    ax.set_ylabel('|FFT(Normalized Intensity)|')

    fig.savefig(ofile, dpi=300)
    plt.cla()
    plt.close(fig)

debug=False
startt=timer()
def timecheck(str):
    global debug, startt
    if (not debug): return
    print("[T]{:.3f}:{}".format(timer() - startt,
str));startt=timer();

def onefile(file):
```

```
    timecheck("Onefile")

    p = Path(file)
    fsplit=p.stem.split('@')
    forg=fsplit[0]
    fopt=fsplit[1]
    fsrc=str(p.parent.joinpath(forg+'.tif'))
    if (os.path.isfile(fsrc) == False): print('   [ERROR] No
source image file...'); return;

    m=re.search('N(\d+)_*', fopt)
    if (m):
        nextend=int(m.groups()[0])
        print('   -- # of normal pixels : '+str(nextend))
    else: nextend=Num_normal_pixels

    fpre=str(p.parent)+'_'+p.stem

    match = re.search('[sp][0-9]+', fsrc)
    polar=''
    if (match):
        polar=fsrc[match.start()]

    ofile1=str(Path('.').joinpath('map_'+fpre+'.png'))
    ofile2=str(Path('.').joinpath('br_'+fpre+'.png'))
    ofile3=str(Path('.').joinpath('out_'+fpre+'.csv'))
    ofile4=str(Path('.').joinpath('fft_'+fpre+'.png'))

    cont=np.loadtxt(file,skiprows=1,usecols=[1,2])
    XS=cont[:,0]; YS=cont[:,1]
    timecheck("LoadTxt")

    img=Image.open(fsrc)
    bpp=mode_to_bpp.get(img.mode)
    bpp_factor=(1<<bpp)//256
    if (bpp_factor <= 0): bpp_factor=1

    (xres,yres)=img.info['resolution']
    xc=(max(XS)+min(XS))/2
    yc=(max(YS)+min(YS))/2
    poly = Polygon(np.transpose([XS,YS]).tolist())

    # Map plotting
    fig, ax = plt.subplots()

    nds=Num_norm_fit_pixels//2
    xpre=ypre=-1
    dist=0
    darr={}

    timecheck("ImageOpen")
    for i in range(nds,len(XS)-nds,1):
        x=XS[i-nds:i+nds+1]; y=YS[i-nds:i+nds+1]
        x0 = XS[i]; y0 = YS[i]
        if (xpre < 0): xpre=x0;ypre=y0
        if (i==nds):
```

```
            ax.annotate('start', xy=(x0, y0), xytext=(x0-30,
y0+10),
                arrowprops=dict(width=4,headwidth=8,
facecolor='red', shrink=0.05))
            ax.annotate('center', xy=(xc, yc), xytext=(xc-30,
yc+10),
                arrowprops=dict(width=4,headwidth=8,
facecolor='red', shrink=0.05))
        try:
            tck = np.polyfit(x,y, deg=2)
        except:
            continue

        der = np.polyder(tck)
        yder = np.polyval(der, x0)
        ynor = -1/yder

        fl = lambda x: ynor*(x-x0)+y0
        fx = lambda r: r/(ynor**2+1)**0.5

        xp=x0+fx(1);yp=fl(xp)
        if (poly.contains(Point(xp,yp))):   sign=-1
        else: sign = 1

        xl = np.arange(0,nextend*sign,sign)
        xx = fx(xl)+x0; yy = fl(xx); xys=zip(xx,yy)
        try:
            pix=list(map(img.getpixel,xys))
        except:
            continue

        br=max(pix[:nextend])
        dist+=dist_calc([x0,y0],[xpre,ypre],[xres,yres])
        angle=math.degrees(math.atan2(-y0+yc,x0-xc))
        if angle < 0: angle+=360
        arr_angle=darr.get('Angle')
        if (arr_angle):
            if (arr_angle[-1] > (angle+300)): angle+=360

        if (polar == 's'): #angle=math.atan2(-1,-ynor)
            xcos=1/((1+1/(ynor**2))**0.5)
        else:           #angle=math.atan(-ynor)
            xcos=1/((ynor**2+1)**0.5)

        update_dic(darr, 'Dist', dist)
        update_dic(darr, 'Angle', angle)
        update_dic(darr, 'Br', br)
        update_dic(darr, 'NTan', -ynor)
        update_dic(darr, 'NCos', xcos)
        update_dic(darr, 'CosBr', br*xcos)

        xpre=x0;ypre=y0

        # Map plotting
        imax=pix.index(br)

pixc=np.array(list(map(adj_color,np.array(pix)/bpp_factor)))
```

```
        plt.scatter(xx,yy,s=1,c=pixc)#,c=list(pixc))
        plt.scatter(xx[imax],yy[imax],s=4,facecolors='none',
edgecolors='y',  linestyle="-", linewidth=0.3)

    timecheck("Preprocess")

    ax.set_title(file)
    ax.set_xlabel('x (pixel)')
    ax.set_ylabel('y (pixel)')
    plt.axis('equal')
    plt.gca().invert_yaxis()
    plt.plot(*poly.exterior.xy, zorder=0);
    fig.savefig(ofile1, dpi=300)
    plt.cla()
    plt.close(fig)

    timecheck("Save Map fig")

    # Frobenius normalization
    arr=darr.get('Br')
    darr.update( { 'NormBr' : arr/np.linalg.norm(arr) } )
    arr=darr.get('CosBr')
    darr.update( { 'NormCosBr' : arr/np.linalg.norm(arr) } )

    if (polar == 's'):
        costxt='(y-axis)'
    else:
        costxt='(x-axis)'

    # Main data plotting
    fig, ax = plt.subplots()
    plt.plot(darr.get('Angle'),darr.get('NormBr'))
    ax.set_title(file)
    #ax.set_xlabel('Distance (um)')
    ax.set_xlabel('Angle (deg)')
    ax.set_ylabel('Normalized PINEM Intensity (a.u.)')
    fig.savefig(ofile2, dpi=300)
    plt.cla()

    timecheck("Save BR fig")

    fft_process(file, fpre, darr)

    timecheck("FFT")

    hdr=""
    delm=','
    outdata=[]
    dlen=len(darr.get(list(header)[0]))
    for hd in header.keys():
        hhd=header[hd]
        hdr+=hhd+delm
        lst=darr.get(hd, [])
        if (len(lst)==0): lst=[ '' ]*dlen
        outdata.append(list(lst))

    np.savetxt(ofile3,
```

```
        np.transpose(np.array(outdata)),
        fmt='%s', #fmt='%.3f',
        delimiter=delm,
        newline='\r\n',
        header=hdr)

    timecheck("Save TXT")

parser = argparse.ArgumentParser(description=__doc__,
formatter_class=argparse.RawTextHelpFormatter)
parser.add_argument('-fa', nargs=1, type=float, default=[0.01],
help='FFT cutoff start(a), default:0.01')
parser.add_argument('-fb', nargs=1, type=float, default=[0.3],
help='FFT cutoff end(b), default:0.3')
parser.add_argument('File', type=str, nargs='+', help='Boundary
text file')

args = parser.parse_args()

for i in args.File:
    printf('>>> Globbing \'%s\' ...',i)
    f=glob.glob(i)
```