In [2]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

In [3]:

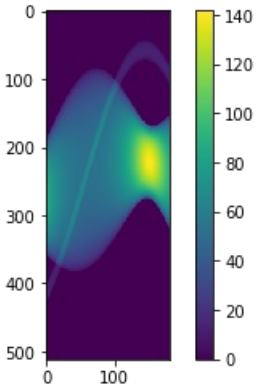
```
def load_data(filename):
    f = open(filename, 'r').read()
    return [list(map(float, s.split('\t')))
            for s in f.replace(' ', '').strip().split('\n')]
```

In [816]:

```
attach_2 = load_data('attachment2.txt')
#plt.figure(figsize=(15, 15))
plt.imshow(attach_2)
plt.colorbar()
```

Out[816]:

```
<matplotlib.colorbar.Colorbar at 0x7fa7a8a1a438>
```

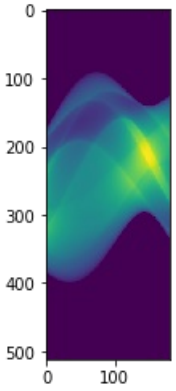


In [5]:

```
attach_3 = load_data('attachment3.txt')
plt.imshow(attach_3)
```

Out[5]:

```
<matplotlib.image.AxesImage at 0x7fa7a875d8d0>
```

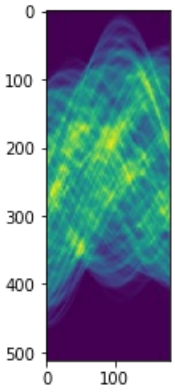


In [6]:

```
attach_5 = load_data('attachment5.txt')
plt.imshow(attach_5)
```

Out[6]:

```
<matplotlib.image.AxesImage at 0x7fa7a841b978>
```

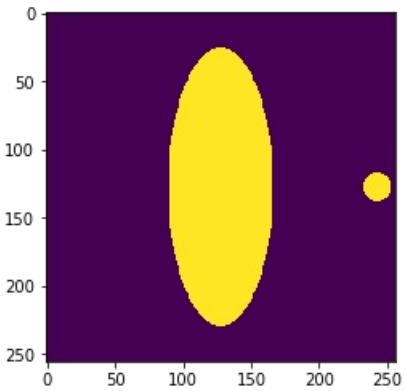


In [7]:

```
attach_1 = load_data('attachment1.txt')
plt.imshow(attach_1)
```

Out[7]:

```
<matplotlib.image.AxesImage at 0x7fa7a81b52e8>
```



Question 1

In [8]:

```
import copy
import numpy as np
```

In [9]:

```
data1 = np.array(attach_2)
data1 = np.transpose(data1)
```

In [10]:

```
# find the position of the circle in the first scanline
for i in range(511, 0, -1):
    if data1[0][i] != 0:
        circle_bottom = i
        break

for i in range(circle_bottom, 0, -1):
    if data1[0][i] == 0:
        circle_top = i + 1
        break
```

In [11]:

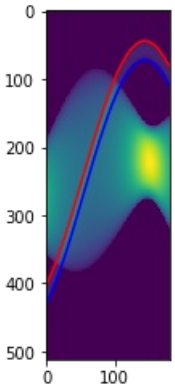
```
circle = [(circle_top, circle_bottom)]
circle_shift = 5

for i in range(1, 120):
    ct, cb = circle[-1]
    # find circle top
    max_now, max_id = -1, 999
    for shift in range(1, circle_shift):
        val = data1[i][ct - shift] - data1[i][ct - shift - 1]
        if val > max_now:
            max_id, max_now = ct - shift, val
    ct = min(max_id, ct)

    # find circle bottom
    max_now, max_id = -1, 999
    for shift in range(1, circle_shift):
        val = data1[i][cb - shift - 1] - data1[i][cb - shift]
        if val > max_now:
            max_id, max_now = cb - shift, val
    cb = min(max_id, cb)

    circle.append((ct, cb))

for i in range(120, 180):
    for j in range(0, 512):
        if data1[i][j] != 0:
            ct = j
            break
    for j in range(ct, 512):
        if data1[i][j] == 0:
            cb = j - 1
            break
    circle.append((ct, cb))
```

In [12]:

```
#plt.figure(figsize=(20, 20))
plt.imshow(attach_2)
plt.plot(range(180), [c[0] for c in circle], 'r',
         range(180), [c[1] for c in circle], 'b')
```

Out[12]:

```
[<matplotlib.lines.Line2D at 0x7fa7a851e0b8>,
 <matplotlib.lines.Line2D at 0x7fa7a8160240>]
```



In [13]:

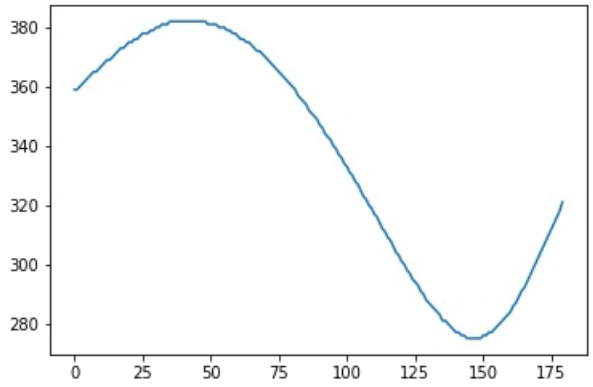
```
def find_bottom(data, start):
    for i in range(start, 0, -1):
        if data[i] != 0:
            return i

ellipse_shift = 5
ellipse_bottom = [find_bottom(data1[0], 390)]

for i in range(1, 30):
    eb = ellipse_bottom[-1]
    # find circle bottom
    max_now, max_id = -1, 0
    for shift in range(1, circle_shift):
        val = data1[i][eb + shift - 1] - data1[i][eb + shift]
        if val > max_now:
            max_id, max_now = eb + shift - 1, val
    ellipse_bottom.append(max(max_id, eb))

for i in range(30, 180):
    ellipse_bottom.append(find_bottom(data1[i], 511))
```

In [14]:

```
plt.plot(ellipse_bottom)
```

Out[14]:

```
[<matplotlib.lines.Line2D at 0x7fa7a80e4f98>]
```



In [15]:

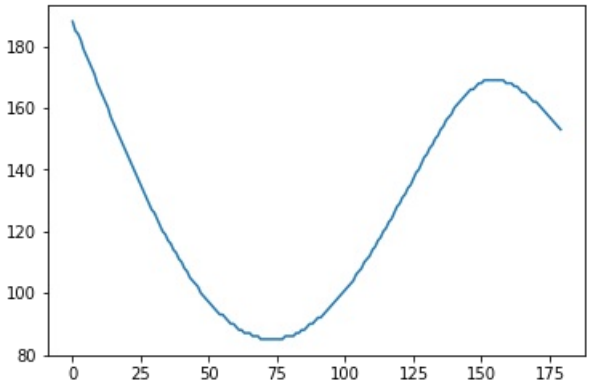
```
def find_top(data, start):
    for i in range(start, 512):
        if data[i] != 0:
            return i

ellipse_top = [find_top(data1[i], 0) for i in range(85)]

for i in range(85, 120):
    et = ellipse_top[-1]
    # find circle top
    max_now, max_id = -1, 0
    for shift in range(0, circle_shift):
        val = data1[i][et + shift] - data1[i][et + shift - 1]
        if val > max_now:
            max_id, max_now = et + shift, val
    ellipse_top.append(max(max_id, et))

ellipse_top.extend([find_top(data1[i], 120) for i in range(120, 180)])
```

In [16]:

```
plt.plot(ellipse_top)
```

Out[16]:

```
[<matplotlib.lines.Line2D at 0x7fa7a27d9e80>]
```

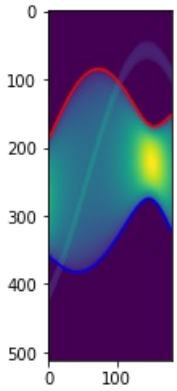


In [17]:

```
plt.imshow(attach_2)
plt.plot(range(180), ellipse_top, 'r',
         range(180), ellipse_bottom, 'b')
```

Out[17]:

```
[<matplotlib.lines.Line2D at 0x7fa7a277f518>,
 <matplotlib.lines.Line2D at 0x7fa7a277a320>]
```



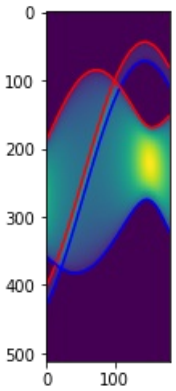
Bound

In [18]:

```
#plt.figure(figsize=(20, 20))
plt.imshow(attach_2)
plt.plot(range(180), [c[0] for c in circle], 'r',
         range(180), [c[1] for c in circle], 'b')
plt.plot(range(180), ellipse_top, 'r',
         range(180), ellipse_bottom, 'b')
```

Out[18]:

```
[<matplotlib.lines.Line2D at 0x7fa7a268ef28>,
 <matplotlib.lines.Line2D at 0x7fa7a269f208>]
```



In [19]:

```
# output to file
open('bound-circle.txt', 'w').write('\n'.join(
    [str(c[0]) + '\t' + str(c[1]) for c in circle]))
open('bound-ellipse.txt', 'w').write('\n'.join(
    [str(ellipse_top[i]) + '\t' + str(ellipse_bottom[i]) for i in range(180)]))
```

Out[19]:

```
1388
```

Concat

In [20]:

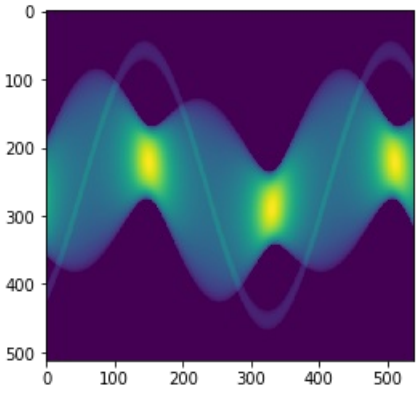
```
reverse_attach_2 = np.transpose(np.array([[data1[i][511 - j] for j in range(512)] for i in range(180)]))
```

In [817]:

```
#plt.figure(figsize=(15, 15))
plt.imshow(np.concatenate([attach_2, reverse_attach_2, attach_2], axis=1))
```

Out[817]:

```
<matplotlib.image.AxesImage at 0x7fa7a8c000b8>
```



Scanline Width

In [22]:

```
import scipy.stats
from sklearn import linear_model
import math
```

In [23]:

```
def get_circle_scanline(i):
    return data1[i][circle[i][0]:circle[i][1] + 1]
```

In [24]:

```
def scanline_width(line):
    n = len(line)
    y = [0.25 * (line[0] * line[0] - line[i] * line[i]) for i in range(1, n)]
    X = [(2 * i, i * i) for i in range(1, n)]
    clf = linear_model.LinearRegression()
    clf.fit(X, y)
    _, LL = clf.coef_
    return math.sqrt(LL)
```

In [25]:

```
l_i_many = [scanline_width(get_circle_scanline(i)) for i in range(110, 180)]
```

l_i is the scanline width in scale of absorbation intensity

In [26]:

```
l_i = np.mean(l_i_many)
```

In [96]:

```
np.std(l_i_many)
```

Out[96]:

```
0.00087448679772017621
```

In [28]:

```
l_i
```

Out[28]:

```
0.49029825753407719
```

Circle Radius in Scale of Intensity

In [29]:

```
d_i_many = [max(get_circle_scanline(i)) for i in range(110, 180)] \
         + [max(get_circle_scanline(i)) for i in range(0, 14)]
d_i = max(d_i_many)
```

In [30]:

```
d_i       # circle radius in intensity
```

Out[30]:

```
14.179600000000001
```

In [95]:

```
np.std(d_i_many)
```

Out[95]:

```
0.0027440130149663398
```

In [32]:

```
d_s = d_i / l_i  # circle radius in scanline
```

In [33]:

```
d_s
```

Out[33]:

```
28.920355685772503
```

Exact Bound

In [177]:

```
def find_bound_s(index, ignore=2):
    cb, ct = circle[index]
    line = data1[index]
    circle_line = line[cb : ct + 1]
    peak = np.argmax(circle_line)

    d_center_s = [0.5 * math.sqrt(d_i * d_i - l * l) / l_i for l in circle_line]

    center_many = []
    for i, dc_s in enumerate(d_center_s[: peak - ignore]):
        center_many.append(cb + i + dc_s)

    for i, dc_s in enumerate(d_center_s[peak + ignore + 1: ]):
        base = cb + i + peak + ignore + 1
        center_many.append(base - dc_s)

    center_s = np.mean(center_many)
 #   print(np.var(center_many))

    return center_s - 0.5 * d_s, center_s + 0.5 * d_s
```

In [178]:

```
bound_0_13 = [find_bound_s(i) for i in range(13)]
bound_110_180 = [find_bound_s(i) for i in range(110, 180)]
```

In [179]:

```
def get_angle(width):
    D = d * width
    s = 2880/11 * (1/225 - 4/(D**2))
    t = math.asin(s)
    n2 = math.atan(math.sqrt((4*40*40-D**2)/(D**2-4*15*15)))
    return n2 * 180 / math.pi
```

In [180]:

```
open('middle_0_13.txt', 'w').write('\n'.join([str(0.5 * (y + x)) for x, y in bound_0_13]))
open('middle_110_180.txt', 'w').write('\n'.join([str(0.5 * (y + x)) for x, y in bound_110_180]))
```

Out[180]:

```
972
```

In [181]:

```
circle[0], circle[164]
```

Out[181]:

```
((401, 429), (56, 84))
```

In [182]:

```
bound_110_180[164-110]
```

Out[182]:

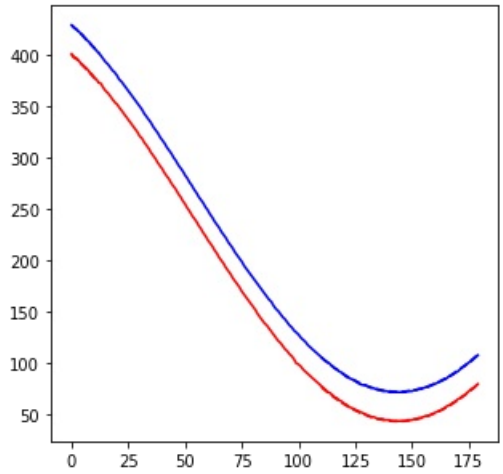
```
(55.841241313946909, 84.761596999719416)
```

In [183]:

```
plt.figure(figsize=(5,5))
plt.plot(range(180), [c[0] for c in circle], 'r',
         range(180), [c[1] for c in circle], 'b')
plt.plot(range(13), [c[0] for c in bound_0_13], 'r--',
         range(13), [c[1] for c in bound_0_13], 'b--')
plt.plot(range(110, 180), [c[0] for c in bound_110_180], 'r--',
         range(110, 180), [c[1] for c in bound_110_180], 'b--')
```

Out[183]:

```
[<matplotlib.lines.Line2D at 0x7fa78e8c0940>,
 <matplotlib.lines.Line2D at 0x7fa78e8cc1d0>]
```



Stupid Regression

In [184]:

```
from calc_coef import calc_coef
```

In [185]:

```
a, b = d_i / 8 * 15, d_i / 8 * 40
```

In [186]:

```
line = data1[0][ellipse_top[0] : ellipse_bottom[0]]
```

In [187]:

```
X_raw = [ calc_coef(a, b, n, l_i, line[n]) for n in range(len(line)) ]
y = [ -x[0] for x in X_raw ]
X = [ x[1:] for x in X_raw ]
```

In [188]:

```
clf = linear_model.LinearRegression()
clf.fit(X, y)
```

Out[188]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [189]:

```
clf.coef_
```

Out[189]:

```
array([  7.52024978e-10,  -4.72863902e-02,  -8.33273515e-04,
         1.02739149e-04,  -9.96910326e-04,   1.32317189e-03,
         2.11497869e-10,   1.48708622e-11,  -4.03513143e-05,
         5.35571895e-05,  -1.76950732e-08,   1.76950730e-08,
         1.20283339e-05,  -1.09348490e-05,   4.77808125e-06,
         8.74005713e-06,  -5.06284261e-08,   8.88440383e-09,
        -4.00446130e-10,  -2.06897167e-09])
```

In [190]:

```
math.asin(clf.coef_[7] ** 0.25) * 180 / 3.1415926535
```

Out[190]:

```
0.11251409946126277
```

In [191]:

```
line[:5]
```

Out[191]:

```
array([  6.5414,  15.1468,  20.3436,  24.4151,  27.8592])
```

In [192]:

```
a, b, l_i
```

Out[192]:

```
(26.586750000000002, 70.897999999999996, 0.49029825753407719)
```

Exact Angle

In [849]:

```
def extract_ellipse(index, ignore=3):
    et, eb = ellipse_top[index] + ignore, ellipse_bottom[index] - ignore
    ct, cb = circle[index]
    ct, cb = ct - ignore, cb + ignore

    line = data1[index]
    scan_lines = []
    for e in range(et, eb):
        if ct <= e <= cb or not line[e]:
            continue
        scan_lines.append((e - et, line[e]))
    return scan_lines
```

In [850]:

```
ellipse_slines = [extract_ellipse(i) for i in range(180)]
```

In [195]:

```
def write_ellipse_to_file(filename, line_id):
    f = open(filename, 'w')
    f.write('\n'.join(str(n) + ' ' + str(L) for n, L in ellipse_slines[line_id]))
    f.close()
```

In [196]:

```
for i in range(180):
    write_ellipse_to_file('ellipse_scanline/%d.txt' % i, i)
```

Waiting for Mathematica's solution. $n \in [138, 161]$

In [851]:

```
import re
import fileinput
```

In [852]:

```
def load_angle_data(filename):
    slope_many = []
    for line in open(filename, 'r').read().split('\n'):
        if line.strip():
            slope_one_many = re.findall(r'\{-?\d*\.?\d*, (-?\d*\.?\d*)\}', line)
            slope_many.append(abs(float(slope_one_many[0])))
    return slope_many
```

$n \in [0, 60], \theta = \pi/2 - \omega$

$n \in [61, 150], \theta = \pi/2 + \omega$

$n \in [151, 179], \theta = 3\pi/2 - \omega$

In [862]:

```
def calc_angle(index):
    if 138 <= index <= 161:
        is_inverted = True
        filename = 'ellipse_scanline/ans_inverted_step_1/%d.txt'
    else:
        is_inverted = False
        filename = 'ellipse_scanline/ans_step_1/%d.txt'
    slope_many = load_angle_data(filename % index)
    if not slope_many: return 0
    slope = np.mean(slope_many)
    omega = math.pi / 2 - math.atan(slope) if is_inverted else math.atan(slope)
    if index <= 60: theta = math.pi / 2 - omega
    elif index <= 150: theta = math.pi / 2 + omega
    else: theta = 3 * math.pi / 2 - omega
    return theta, np.std(np.arctan(slope_many))
```

In [863]:

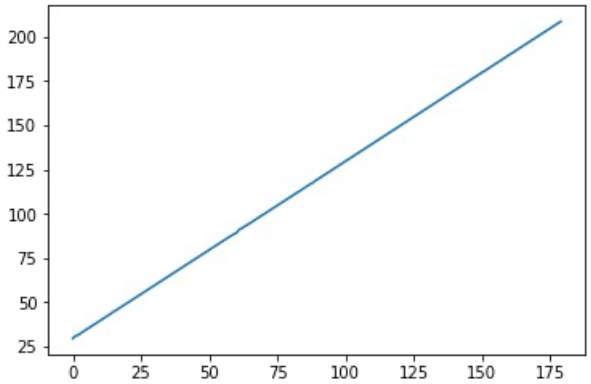
```
angles = [ calc_angle(i)[0] * 180 / math.pi for i in range(180) ]
```

In [855]:

```
plt.plot(range(180), angles)
```

Out[855]:

```
[<matplotlib.lines.Line2D at 0x7fa77f8375f8>]
```



In [202]:

```
scipy.stats.linregress(range(180), angles[:180])
```

Out[202]:

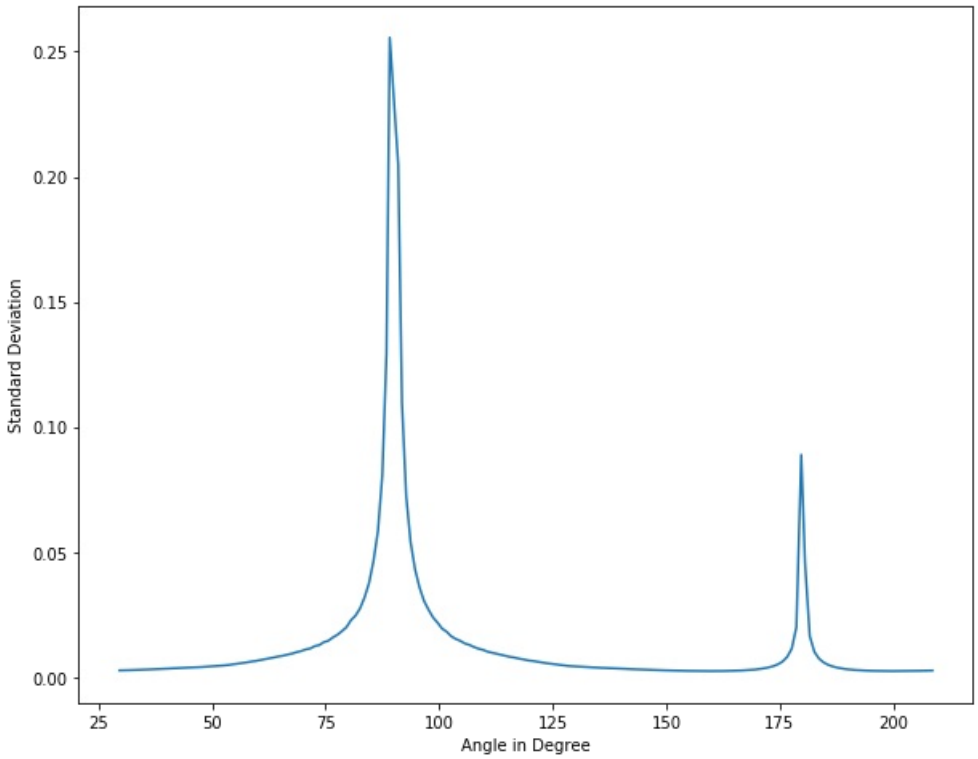
```
LinregressResult(slope=1.0000157321033163, intercept=29.645265082606542, rvalue=0.9999991820210
5853, pvalue=0.0, stderr=9.5870094830890972e-05)
```

In [877]:

```
plt.figure(figsize=(10, 8))
plt.plot(angles, [ calc_angle(i)[1] * 180 / math.pi for i in range(180) ])
plt.xlabel('Angle in Degree')
plt.ylabel('Standard Deviation')
```

Out[877]:

```
<matplotlib.text.Text at 0x7fa78214ce48>
```



In [203]:

```
angle_f = open('angle_middle_0_13.txt', 'w')
angle_f.write('\n'.join([str(angles[i]) + ' ' + str((bound_0_13[i][1] + bound_0_13[i][0] - 511) / 2) for i i
n range(13)]))
angle_f.close()
angle_f = open('angle_middle_110_180.txt', 'w')
angle_f.write('\n'.join([str(angles[i]) + ' ' + str((bound_110_180[i - 110][1] + bound_110_180[i - 110][0] -
511) / 2) for i in range(110, 180)]))
angle_f.close()
```

Roatation Center

In [204]:

```
def load_center_data(filename):
    data = open(filename, 'r').read()
    center_many = re.findall(r'\{(-?\d*\.?\d*), (-?\d*\.?\d*)\}', data)
    center_many = [ list(map(float, line)) for line in center_many ]
    return center_many
```

In [209]:

```
center_offset_many_s = load_center_data('center_coords.txt')
```

In [210]:

```
center_offset_s = np.mean(center_offset_many_s, axis=0)
```

In [220]:

```
np.std(center_offset_many_s * 8 / d_s, axis=0)  # stddev in mm
```

Out[220]:

```
array([ 0.00067592,  0.00130586])
```

In [221]:

```
center_mm = [45, 0] - 8 / d_s * center_offset_s
```

In [222]:

```
center_mm
```

Out[222]:

```
array([-9.23831002,  6.26628192])
```

Question 2

In [224]:

```
d_mm = 8      # diameter of circle in mm
scale_i2mm = d_mm / d_i
scale_mm2p = 256 / 100.
scale_i2p  = scale_i2mm * scale_mm2p
```

In [228]:

```
proj_axis_s = np.array(range(512)) - 0.5 * 511
proj_axis_p = proj_axis_s * l_i * scale_i2p
```

In [234]:

```
center_p = center_mm * scale_mm2p
```

In [261]:

```
thetas = np.array(angles) * math.pi / 180
```

In [315]:

```
def proj_pixel_dist(pixel, theta, center_p):
    p_x = pixel[0] - center_p[0]
    p_y = pixel[1] - center_p[1]
    return p_x * np.cos(theta) + p_y * np.sin(theta)
```

In [331]:

```
def proj_direct(g, axis_p, theta, center_p):
    size = 256
    mu = np.zeros([size, size])
    for n in range(len(theta)):
        R_flatten = [ proj_pixel_dist([x - (size - 1) / 2, y - (size - 1) / 2], theta[n], center_p)
                     for x in range(size) for y in range(size) ]
        g_theta = np.reshape(np.interp(R_flatten, axis_p, g[n]), [size, size])
        mu = mu + g_theta
    return np.flip(np.transpose(mu), axis=0)
```

Directly Inverse Projection

In [332]:

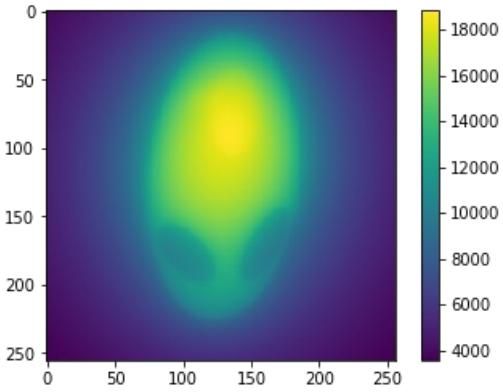
```
mu_data = proj_direct(np.transpose(attach_3), proj_axis_p, thetas, center_p)
```

In [333]:

```
plt.imshow(mu_data)
plt.colorbar()
```

Out[333]:

<matplotlib.colorbar.Colorbar at 0x7fa78d32da58>



In [319]:

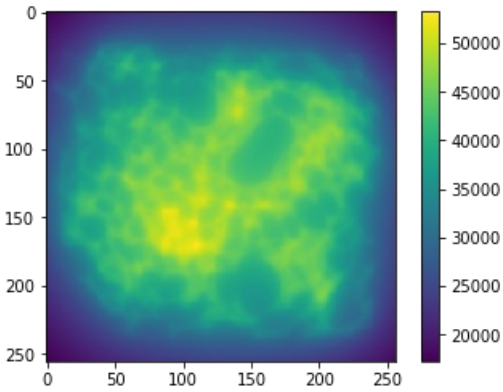
```
mu_data = proj_direct(np.transpose(attach_5), proj_axis_p, thetas, center_p)
```

In [320]:

```
plt.imshow(mu_data)
plt.colorbar()
```

Out[320]:

<matplotlib.colorbar.Colorbar at 0x7fa78e44eac8>



In [321]:

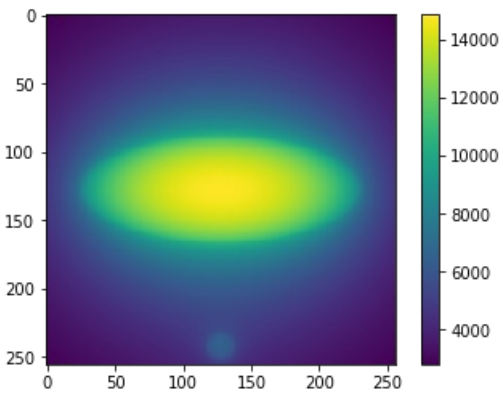
```
mu_data = proj_direct(data1, proj_axis_p, thetas, center_p)
```

In [322]:

```
plt.imshow(np.transpose(mu_data))
plt.colorbar()
```

Out[322]:

<matplotlib.colorbar.Colorbar at 0x7fa78d5cb0b8>



Filtering

In [520]:

```
def filter_projection(g, H):
    return np.real(np.fft.ifft(np.fft.fft(g, axis=1) * H, axis=1))
```

In [645]:

```
def RL_conv(rho_0, length):
    T = 0.5 / rho_0
    h = []
    for i in range(0, length // 2 + 1):
        if i == 0:
            h.append(0.25 / T ** 2)
        elif i % 2 == 0:
            h.append(0)
        else:
            h.append(-1. / (math.pi * i * T) ** 2)
    h += list(reversed(h[1:-1]))
    return np.real(np.fft.fft(h))
```

In [792]:

```
def MRL3_conv(rho_0, length):
    T = 0.5 / rho_0
    h = []
    for i in range(0, length // 2 + 1):
        if i == 0:
            h.append(0.25 / T ** 2)
        elif i % 2 == 0:
            h.append(0)
        else:
            h.append(-1. / (math.pi * i * T) ** 2)
    h += list(reversed(h[1:-1]))
    t = [ h[i - 1] / 4 + h[(i + 1) % length] / 4 + h[i] / 2 for i in range(length) ]
    return np.real(np.fft.fft(t))
```

In [560]:

```
def SL_conv(rho_0, length):
    T = 0.5 / rho_0
    h = []
    for i in range(0, length // 2 + 1):
        h.append(-2 / (math.pi * T) ** 2 /  (4 * i ** 2 - 1))
    h += list(reversed(h[1:-1]))
    return np.real(np.fft.fft(h))
```

In [584]:

```
def PL_conv(rho_0, length):
    T = 0.5 / rho_0
    h = []
    for i in range(0, length // 2 + 1):
        if i == 0:
            h.append(math.pi / (3 * T**2))
        else:
            h.append(-1 / (i * T) ** 2)
    h += list(reversed(h[1:-1]))
    return np.real(np.fft.fft(h))
```

In [670]:

```
def get_kernel():
    return RL_conv(0.5 / (l_i * scale_i2p), 512)
```

In [799]:

```
def inv_proj(data, kernel=None):
    if kernel is None: kernel = get_kernel()
    return proj_direct(filter_projection(np.transpose(data), kernel), proj_axis_p, thetas, center_p)
```

In [818]:

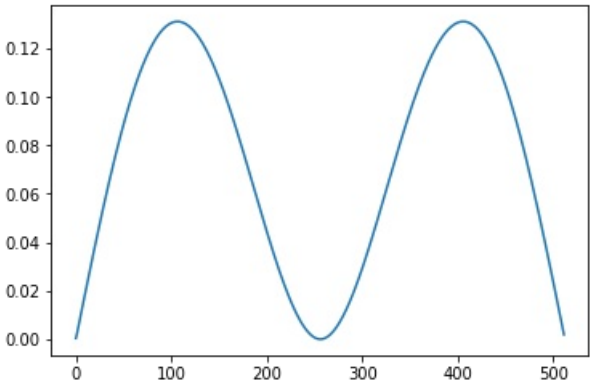
```
def filter_min0(data):
    for i in range(256):
        for j in range(256):
            data[i][j] = max(data[i][j], 0)
    return data
```

In [796]:

```
plt.plot(range(512), MRL3_conv(0.5, 512))
```

Out[796]:

```
[<matplotlib.lines.Line2D at 0x7fa77decd400>]
```



In [803]:

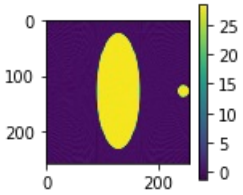
```
test_recons = inv_proj(attach_2, MRL3_conv(0.5, 512))
```

In [805]:

```
plt.figure(figsize=(2, 2))
plt.imshow(test_recons)
plt.colorbar()
```

Out[805]:

```
<matplotlib.colorbar.Colorbar at 0x7fa77dc42278>
```

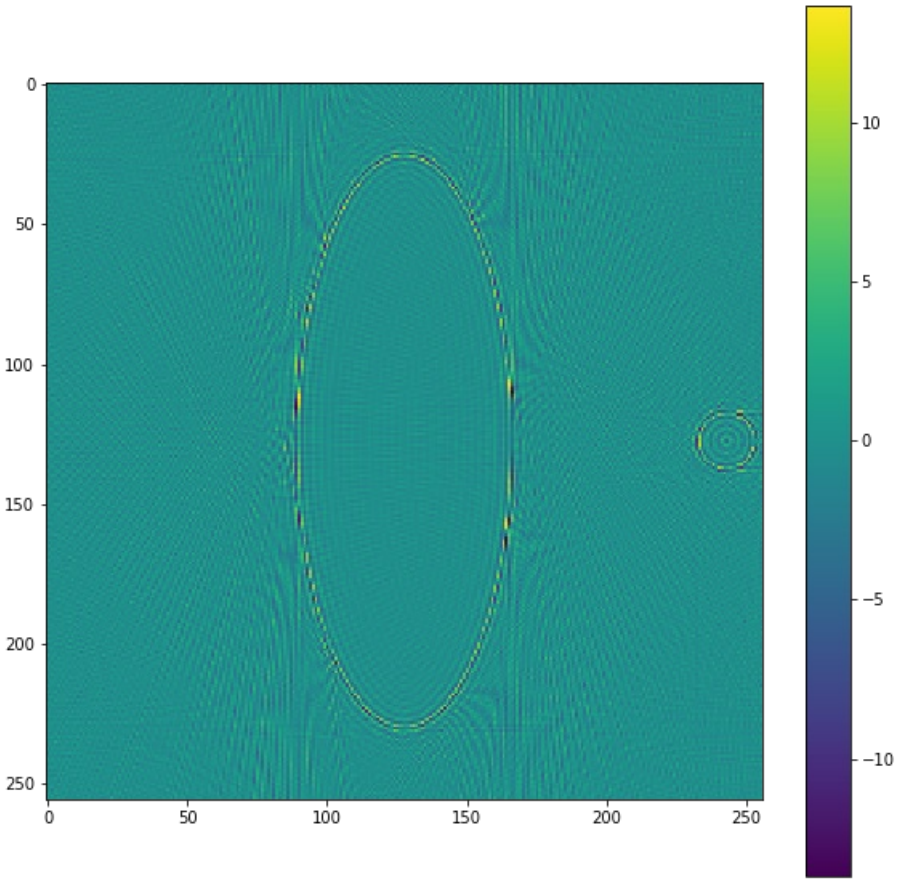


In [635]:

```
mu_attach2_RL = proj_direct(filter_projection(np.transpose(attach_2), RL_conv(0.5 / (l_i * scale_i2p), 512,
128)),
                            proj_axis_p, thetas, center_p)
plt.figure(figsize=(10, 10))
plt.imshow(mu_attach2_RL)
plt.colorbar()
```

Out[635]:

```
<matplotlib.colorbar.Colorbar at 0x7fa780890748>
```



In [594]:

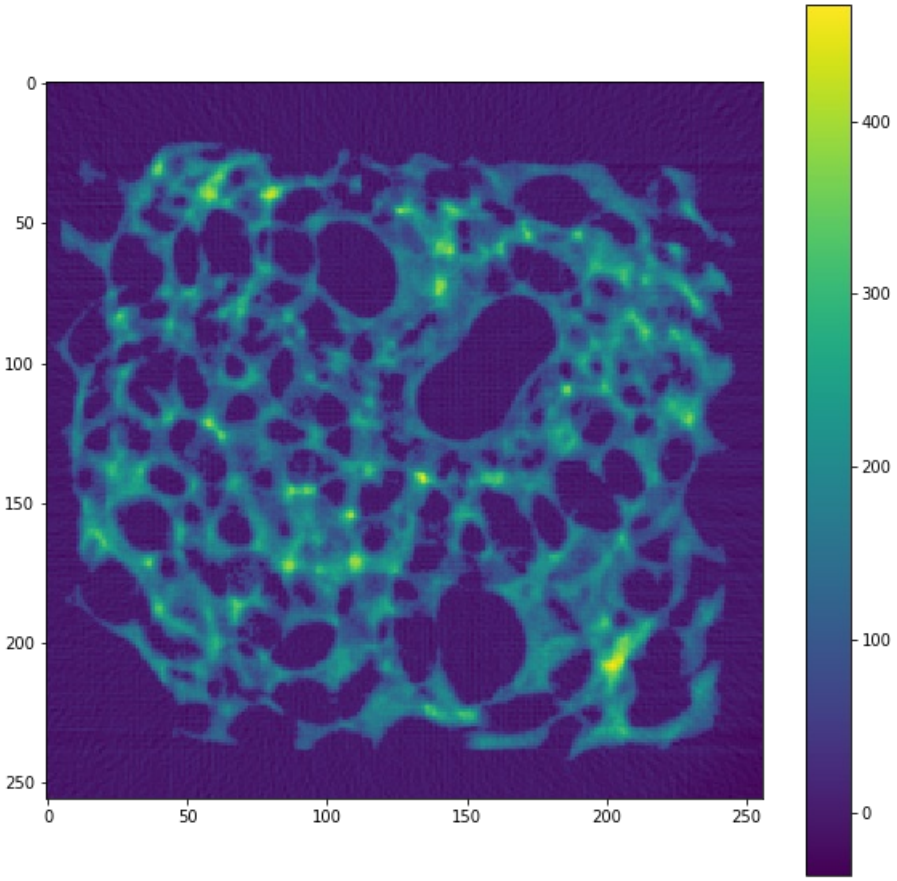
```
mu_attach5_RL = proj_direct(filter_projection(np.transpose(attach_5), RL_conv(0.5 / (l_i * scale_i2p), 512))
,
                            proj_axis_p, thetas, center_p)
```

In [608]:

```
plt.figure(figsize=(10, 10))
plt.imshow(mu_attach5_RL)
plt.colorbar()
```

Out[608]:

```
<matplotlib.colorbar.Colorbar at 0x7fa787a837f0>
```



In [624]:

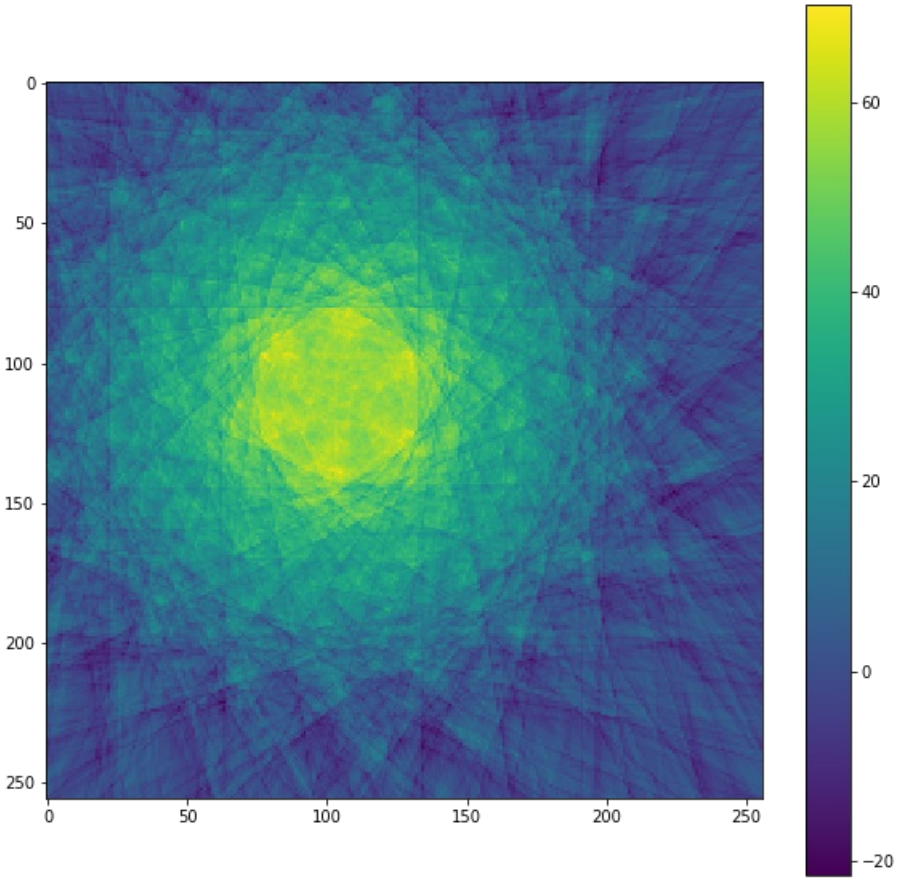
```
mu_attach3_RL = proj_direct(filter_projection(np.transpose(attach_3), RL_conv(0.5 / (l_i * scale_i2p), 512))
,
                            proj_axis_p, thetas[0] + np.array([i for i in range(180)]), center_p)
```

In [625]:

```
plt.figure(figsize=(10, 10))
plt.imshow(mu_attach3_RL)
plt.colorbar()
```

Out[625]:

```
<matplotlib.colorbar.Colorbar at 0x7fa780e1cac8>
```

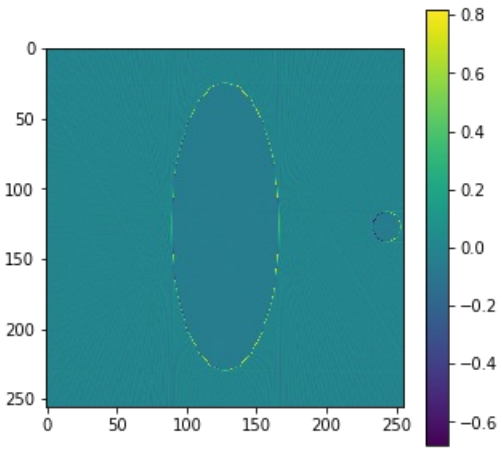


In [626]:

```
plt.figure(figsize=(5, 5))
plt.imshow(mu_attach2_RL / np.max(np.abs(mu_attach2_RL)) - attach_1)
plt.colorbar()
```

Out[626]:

```
<matplotlib.colorbar.Colorbar at 0x7fa780d1f7f0>
```

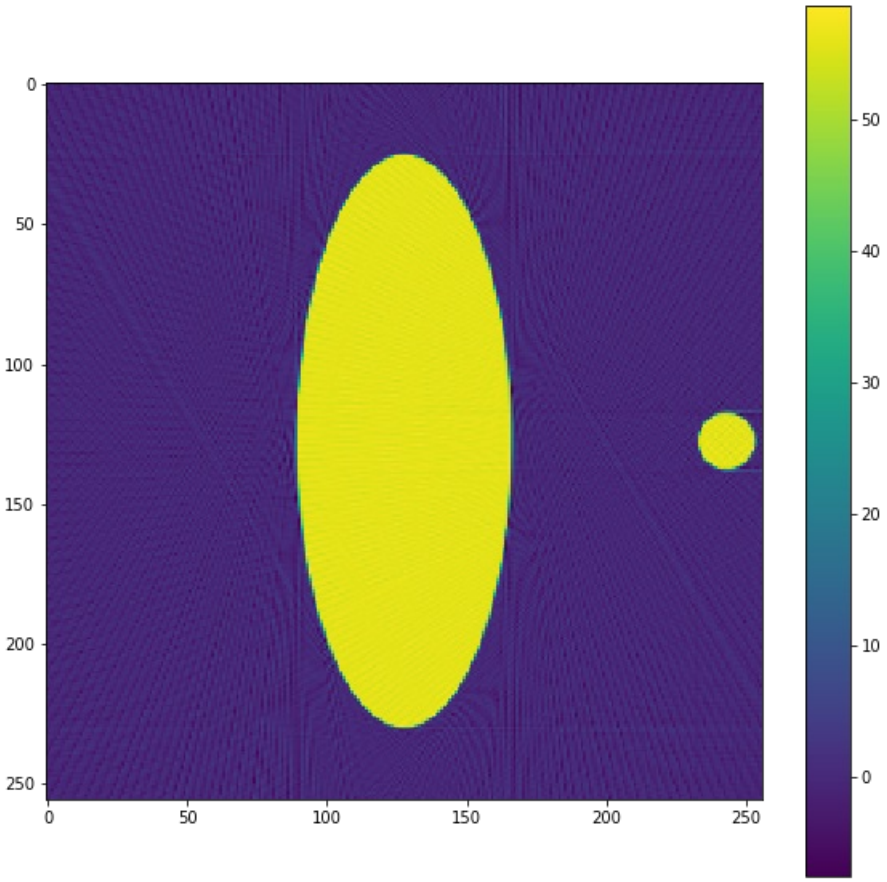


In [646]:

```
mu_attach2_RL = proj_direct(filter_projection(np.transpose(attach_2), RL_conv(0.5 / (l_i * scale_i2p), 512))
,
                            proj_axis_p, thetas, center_p)
plt.figure(figsize=(10, 10))
plt.imshow(mu_attach2_RL)
plt.colorbar()
```

Out[646]:

```
<matplotlib.colorbar.Colorbar at 0x7fa7801579e8>
```



Generate Image

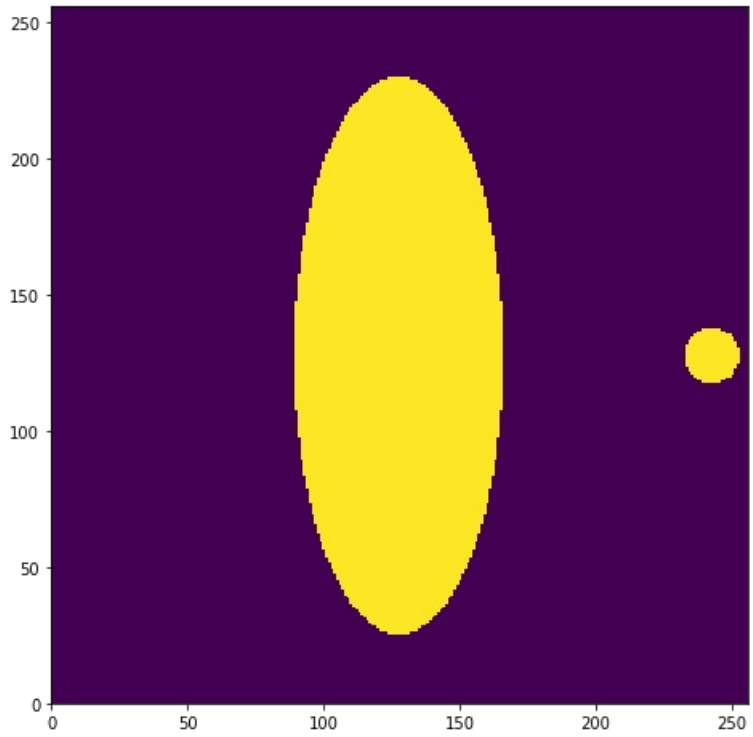
Attachment 1

In [772]:

```
plt.figure(figsize=(8, 8))
plt.xlim((0, 256))
plt.ylim((0, 256))
plt.imshow(np.flip(attach_1, axis=0))
```

Out[772]:

```
<matplotlib.image.AxesImage at 0x7fa77e390ef0>
```



Attachment 2

In [821]:

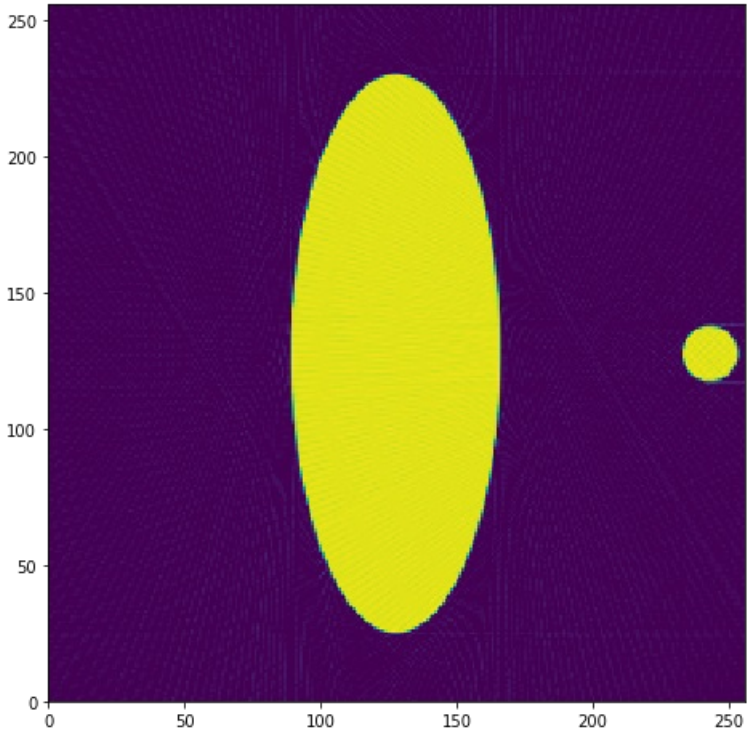
```
attach2_recons = filter_min0(inv_proj(attach_2))
```

In [822]:

```
plt.figure(figsize=(8, 8))
plt.xlim((0, 256))
plt.ylim((0, 256))
plt.imshow(np.flip(attach2_recons, axis=0))
```

Out[822]:

```
<matplotlib.image.AxesImage at 0x7fa77d70ccc0>
```



Attachment 3

In [823]:

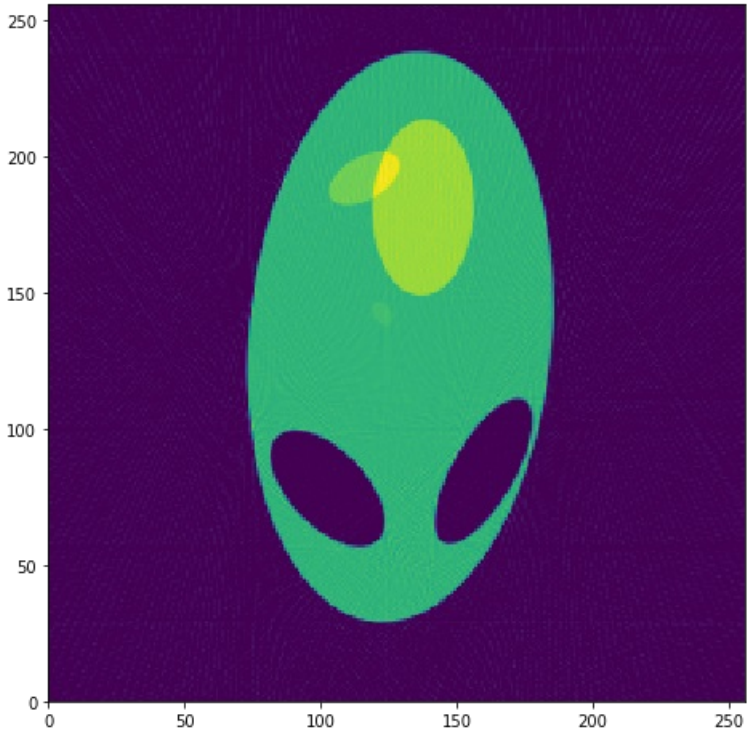
```
attach3_recons = filter_min0(inv_proj(attach_3))
```

In [824]:

```
plt.figure(figsize=(8, 8))
plt.xlim((0, 256))
plt.ylim((0, 256))
plt.imshow(np.flip(attach3_recons, axis=0))
```

Out[824]:

```
<matplotlib.image.AxesImage at 0x7fa77d69ab70>
```



Attachment 5

In [825]:

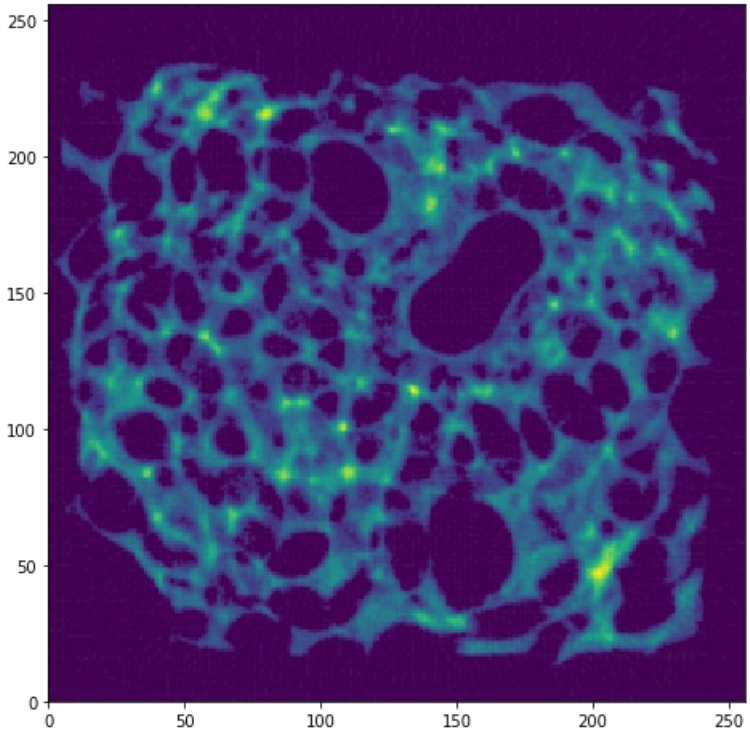
```
attach5_recons = filter_min0(inv_proj(attach_5))
```

In [837]:

```
plt.figure(figsize=(8, 8))
plt.xlim((0, 256))
plt.ylim((0, 256))
plt.imshow(np.flip(attach5_recons, axis=0))
```

Out[837]:

```
<matplotlib.image.AxesImage at 0x7fa77d2932b0>
```



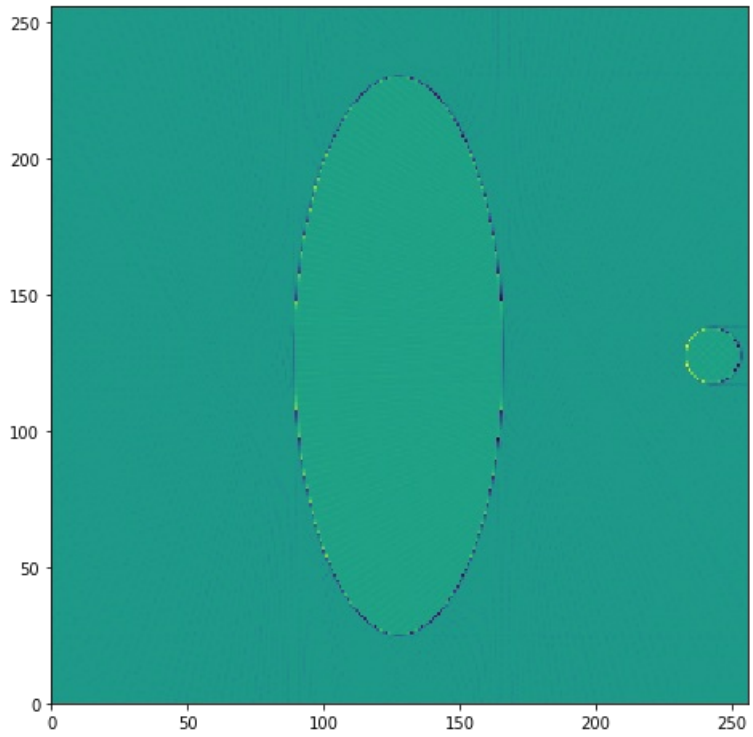
Attachment 2 - Attachment 1

In [827]:

```
plt.figure(figsize=(8, 8))
plt.xlim((0, 256))
plt.ylim((0, 256))
plt.imshow(np.flip(attach_1 - attach2_recons / np.max(np.abs(attach2_recons)), axis=0))
```

Out[827]:

```
<matplotlib.image.AxesImage at 0x7fa7a8d3f748>
```



Attachment 2 Directly

In [661]:

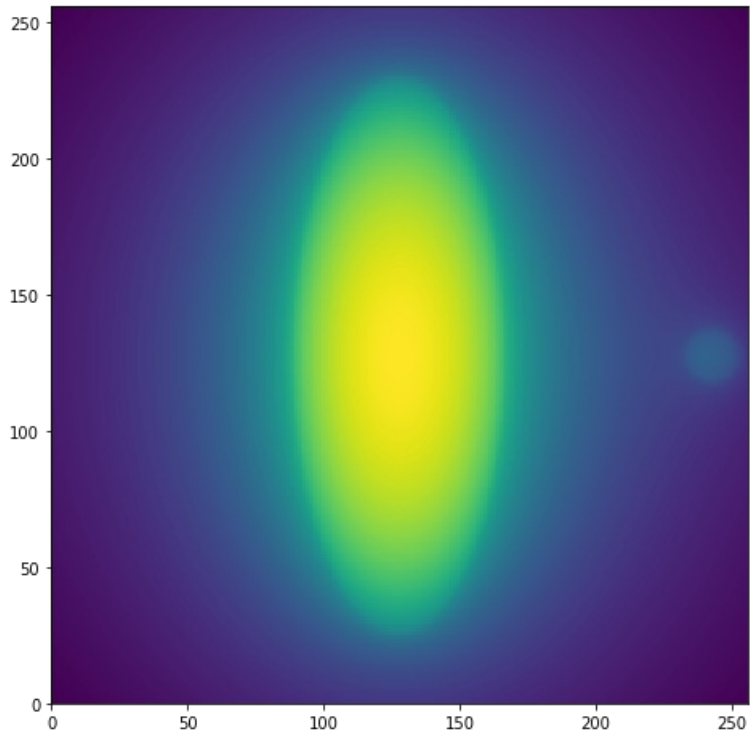
```
attach2_direct = proj_direct(np.transpose(attach_2), proj_axis_p, thetas, center_p)
```

In [767]:

```
plt.figure(figsize=(8, 8))
plt.xlim((0, 256))
plt.ylim((0, 256))
plt.imshow(np.flip(attach2_direct, axis=0))
```

Out[767]:

```
<matplotlib.image.AxesImage at 0x7fa77e64b780>
```



Attachment 3 Directly

In [663]:

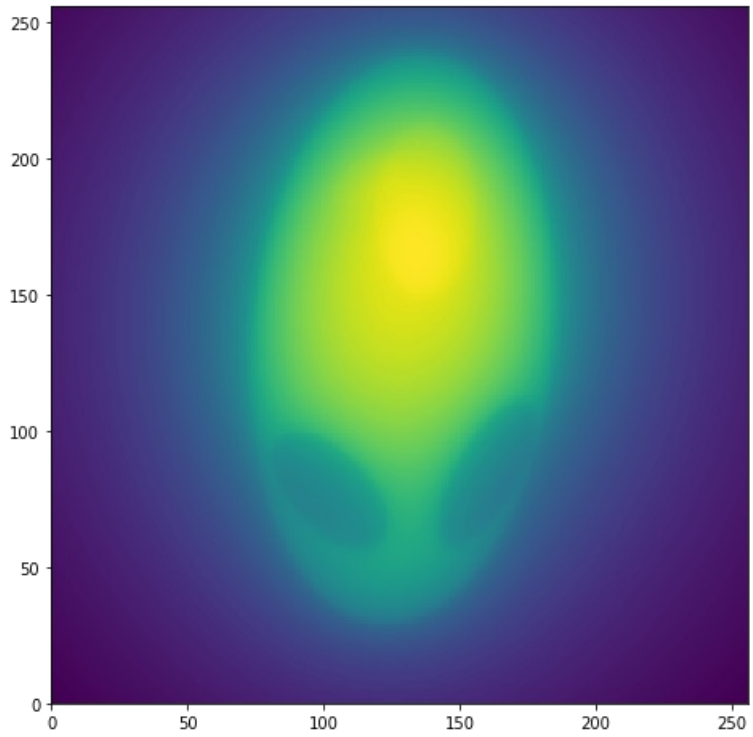
```
attach3_direct = proj_direct(np.transpose(attach_3), proj_axis_p, thetas, center_p)
```

In [766]:

```
plt.figure(figsize=(8, 8))
plt.xlim((0, 256))
plt.ylim((0, 256))
plt.imshow(np.flip(attach3_direct, axis=0))
```

Out[766]:

```
<matplotlib.image.AxesImage at 0x7fa77e6c8320>
```



Attachment 5 Directly

In [665]:

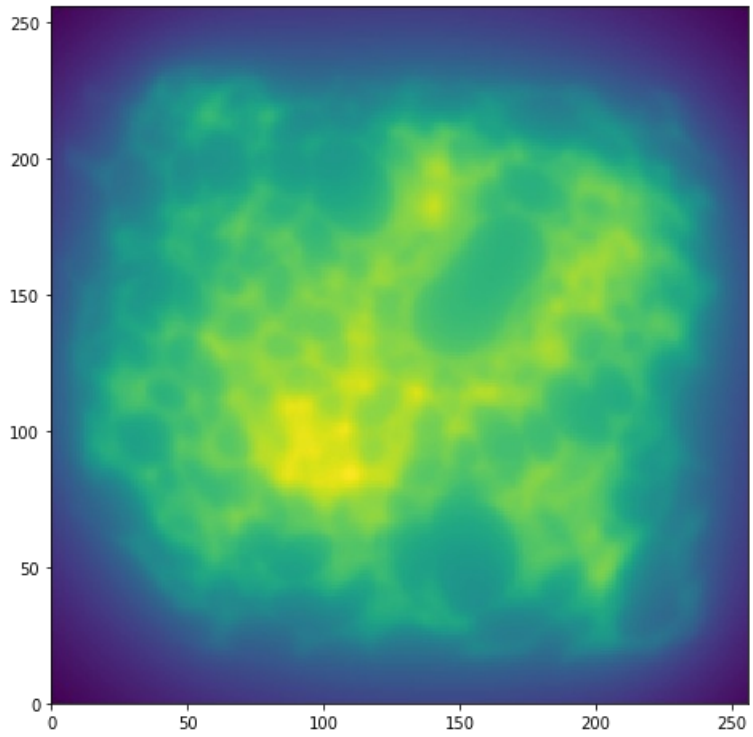
```
attach5_direct = proj_direct(np.transpose(attach_5), proj_axis_p, thetas, center_p)
```

In [765]:

```
plt.figure(figsize=(8, 8))
plt.xlim((0, 256))
plt.ylim((0, 256))
plt.imshow(np.flip(attach5_direct, axis=0))
```

Out[765]:

```
<matplotlib.image.AxesImage at 0x7fa77e7acb70>
```



Compute Recons Intensity

In [776]:

```
max_attach2_recons = np.max(np.abs(attach2_recons))
scale_recons = 1.0 / np.mean([ x for x in np.reshape(np.abs(attach2_recons), [256 * 256]) if x > max_attach2
_recons * 0.5 ])
```

In [777]:

```
attach_4 = load_data('attachment4.txt')
pixels = np.array(attach_4) * scale_mm2p
pixels_x, pixels_y = zip(*pixels)
```

In [778]:

```
pixels
```

Out[778]:

```
array([[  25.6 ,   46.08],
       [  88.32,   64.  ],
       [ 111.36,   84.48],
       [ 115.2 ,  193.28],
       [ 124.16,  142.08],
       [ 128.  ,  193.28],
       [ 143.36,  195.84],
       [ 167.68,   94.72],
       [ 203.52,   46.08],
       [ 252.16,  111.36]])
```

In [828]:

```
def compute_recons_intensity(data, pixels):
    g = filter_projection(np.transpose(data), get_kernel())
    mu = np.zeros(len(pixels))
    for n in range(len(thetas)):
        R = [ proj_pixel_dist(pixel, thetas[n], center_p) for pixel in pixels - (256 - 1) / 2 ]
        g_theta = np.interp(R, proj_axis_p, g[n])
        mu = mu + g_theta
    for i in range(len(pixels)):
        mu[i] = max(mu[i], 0)
    return mu * scale_recons
```

In [829]:

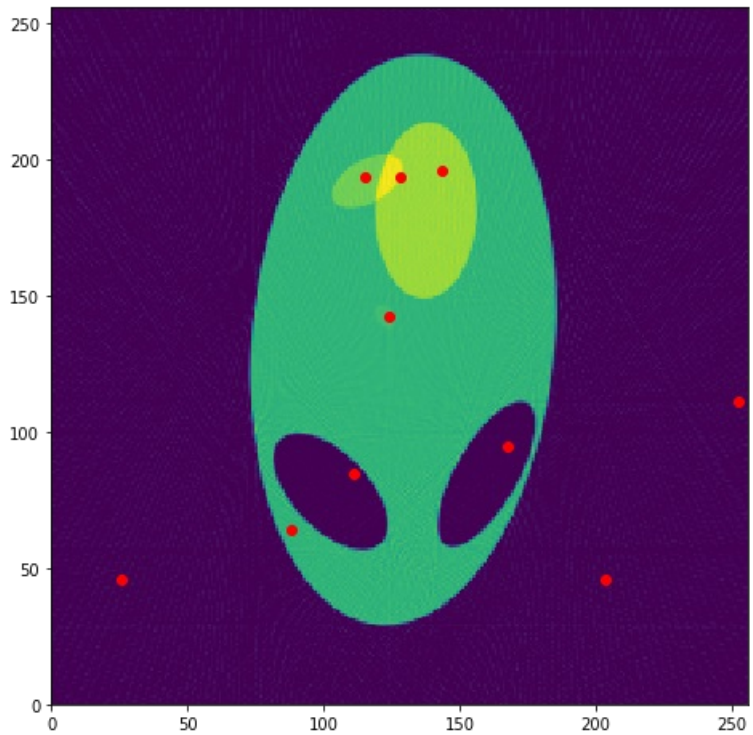
```
def compute_approx_intensity(data, pixels):
    return np.array([ data[255 - int(p[1])][int(p[0])] for p in pixels ]) * scale_recons
```

In [830]:

```
plt.figure(figsize=(8, 8))
plt.xlim((0, 256))
plt.ylim((0, 256))
plt.imshow(np.flip(attach3_recons, axis=0))
plt.scatter(pixels_x, pixels_y, color='red')
```

Out[830]:

```
<matplotlib.collections.PathCollection at 0x7fa77d4cb128>
```



In [831]:

```
compute_recons_intensity(attach_3, pixels)
```

Out[831]:

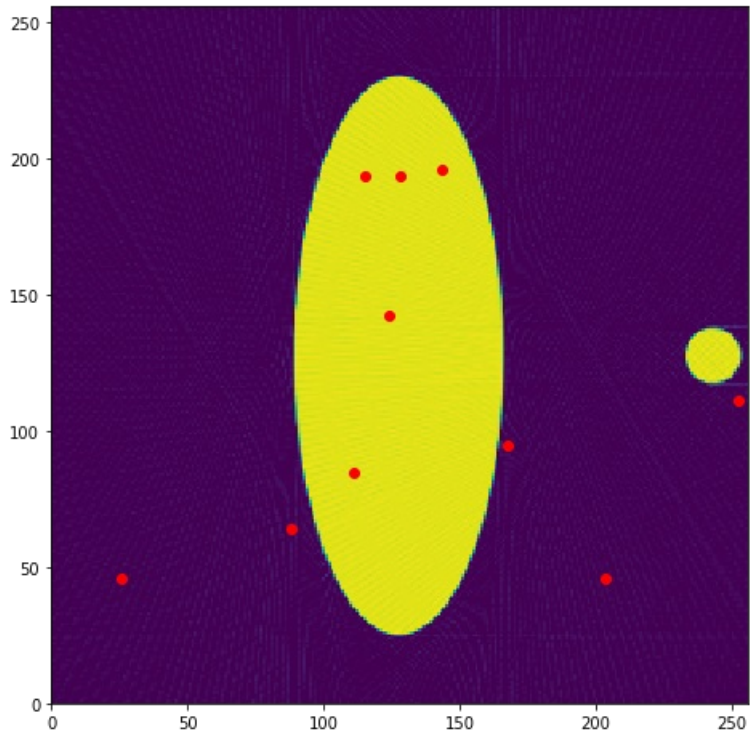
```
array([ 0.        ,  0.99785355,  0.        ,  1.20498938,  1.08658072,
        1.4174558 ,  1.29149607,  0.00640526,  0.0285707 ,  0.        ])
```

In [832]:

```
plt.figure(figsize=(8, 8))
plt.xlim((0, 256))
plt.ylim((0, 256))
plt.imshow(np.flip(attach2_recons, axis=0))
plt.scatter(pixels_x, pixels_y, color='red')
```

Out[832]:

```
<matplotlib.collections.PathCollection at 0x7fa77d462278>
```



In [833]:

```
compute_recons_intensity(attach_2, pixels)
```

Out[833]:

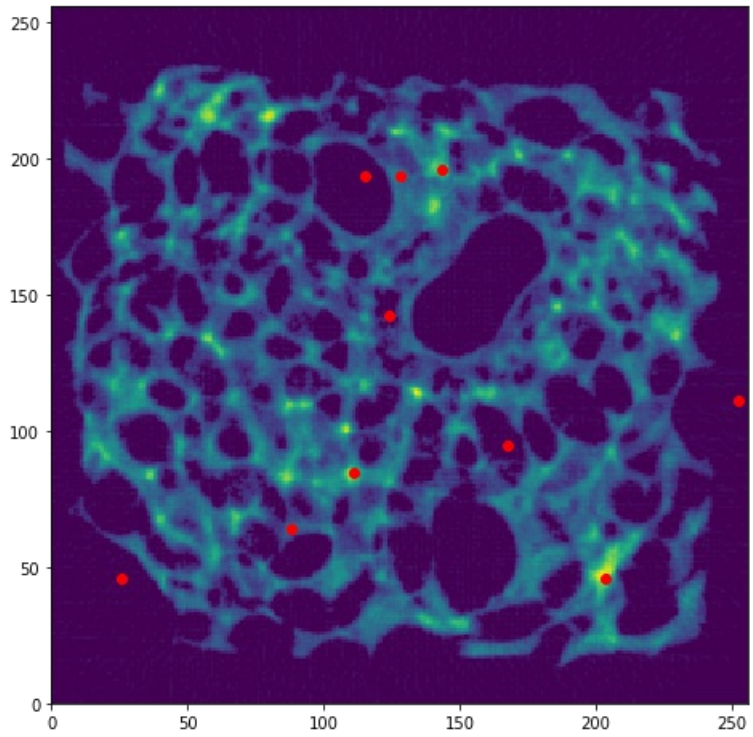
```
array([ 0.01816869,  0.03120544,  0.99190348,  1.00581171,  1.00905103,
        0.99070229,  0.99278848,  0.        ,  0.        ,  0.        ])
```

In [834]:

```
plt.figure(figsize=(8, 8))
plt.xlim((0, 256))
plt.ylim((0, 256))
plt.imshow(np.flip(attach5_recons, axis=0))
plt.scatter(pixels_x, pixels_y, color='red')
```

Out[834]:

```
<matplotlib.collections.PathCollection at 0x7fa77d3e6f98>
```



In [835]:

```
compute_recons_intensity(attach_5, pixels)
```

Out[835]:

```
array([ 0.07803615,  2.82267903,  6.79651207,  0.19936046,  0.1626274 ,
        3.1407299 ,  6.46755403,  0.        ,  7.31355592,  0.        ])
```

In [808]:

```
256*1.41/l_i / scale_i2p
```

Out[808]:

```
509.72126896174029
```

Save

In [841]:

```
open('attach5_recons.txt', 'w').write('\n'.join([','.join(['%.4f' % attach5_recons[i][j] for j in range(256)
]) for i in range(256)]))
```

Out[841]:

```
512968
```

In [842]:

```
open('attach3_recons.txt', 'w').write('\n'.join([','.join(['%.4f' % attach3_recons[i][j] for j in range(256)
]) for i in range(256)]))
```

Out[842]:

```
475022
```

In [843]:

```
open('attach2_recons.txt', 'w').write('\n'.join([','.join(['%.4f' % attach2_recons[i][j] for j in range(256)
]) for i in range(256)]))
```

Out[843]:

```
471598
```

In [876]:

```
a, b, l_i
```

Out[876]:

```
(26.586750000000002, 70.897999999999996, 0.49029825753407719)
```

In [888]:

```
open('angles_tex.txt', 'w').write('\\\\ \\hline\n'.join([' & '.join([ '$%.4f^\circ$' % angle for angle in an
gles[i * 6: i * 6 + 6]]) for i in range(30)]))
```

Out[888]:

```
3549
```

In [887]:

```
open('angles.txt', 'w').write('\n' .join(['%.4f' % angle for angle in angles]))
```

Out[887]:

```
1548
```

In [889]:

```
center_mm
```

Out[889]:

```
array([-9.23831002,  6.26628192])
```