In [1]:

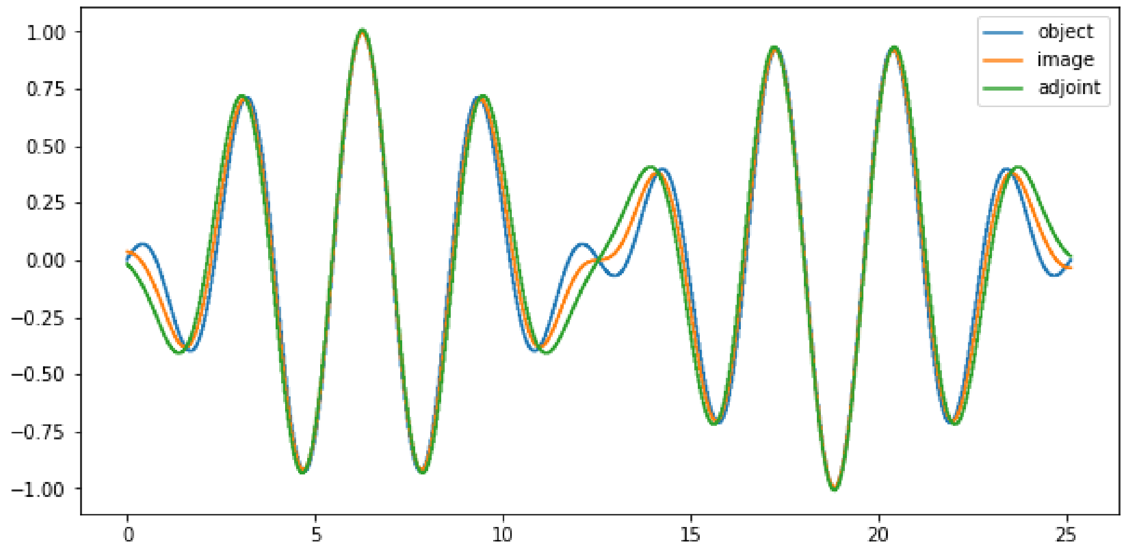
```python
import numpy as np
import matplotlib.pyplot as plt
import skimage.transform as st
import imageio
import scipy.signal as ss

```

In [2]:

```python
# Q.1
expanded_x = np.linspace(-0.25*np.pi,8.*np.pi+0.25*np.pi,851)
x = expanded_x[25:826]
obj = np.zeros_like(expanded_x)
obj[25:826] = np.cos(2.*x)*np.sin(x/4.)
kernel = np.ones(51)

dint= x[1]-x[0]

## Hf =
img = np.zeros_like(x)
for i in range(25,826):
    for j in range(51):
        img[i-25] += np.flip(kernel)[j] * obj[i-25+j] * dint

## H*g =
adj = np.zeros_like(x)
eimg = np.zeros_like(expanded_x)
eimg[25:826] = img
for i in range(25,826):
    for j in range(51):
        adj[i-25] += kernel[j] * eimg[i-25+j] *dint
```

In [3]:
```python
#Object
plt.figure(figsize=(10,5))
plt.step(x,obj[25:826])
plt.step(x,img)
plt.step(x,adj)
plt.legend(['object','image','adjoint'])
```

Out[3]:
```
<matplotlib.legend.Legend at 0x7f1c28df77d0>
```

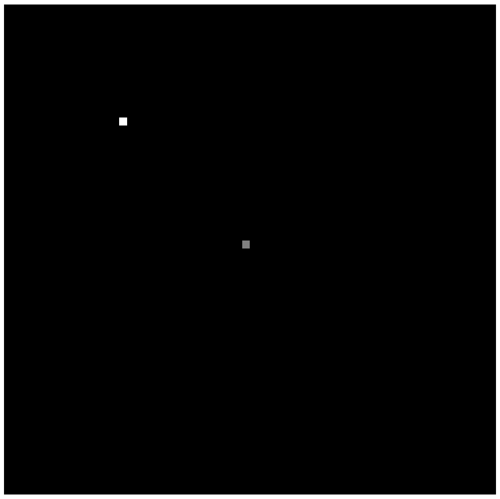


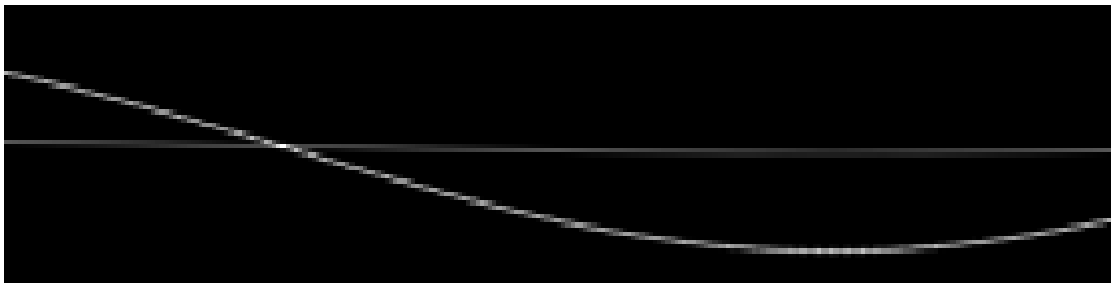
3

In [4]:
```python
#Q.3
obj = np.zeros([64,64])
obj[31,31] = 64
obj[15,15]  =128
theta = np.linspace(0,180,256)
img = st.radon(obj, theta)
```

In [5]:

```python
figure,ax = plt.subplots(2,1,figsize=(10,10))
shows = [obj, img]
for i in range(2):
    ax[i].imshow(shows[i],cmap='gray')
    ax[i].axis('off')
```





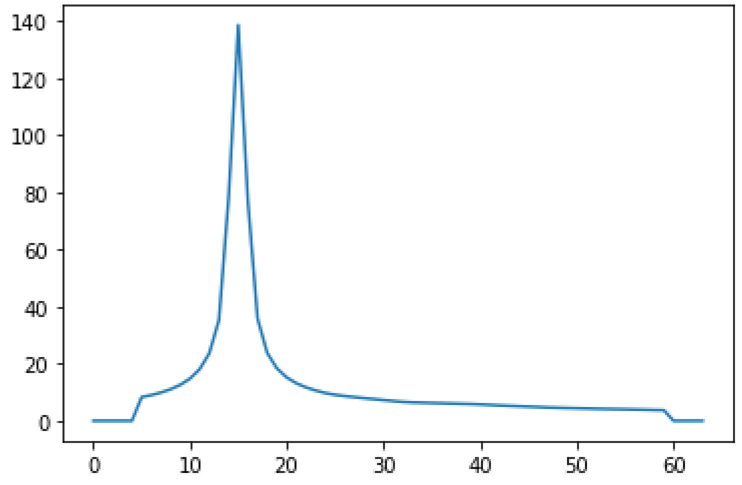
In [6]:

```
# b will be included in c, this is not full description for b
import scipy.signal as ssig

# def width(profile,point):
#     profile = profile[point-5:point+14]
#     print(len(profile))
#     return ssig.peak_widths(profile,[5])# +([point],[point],[p
def width(profile,point):
    return ssig.peak_widths(profile,[point])# +([point],[point],


iradon = st.iradon(img,theta,filter=None)
#plt.imshow(iradon,cmap='gray')
profile_center = iradon[:,31]
profile_corner = iradon[:,15]
width_center = width(profile_center,31)
width_corner = width(profile_corner,15)
print(width_corner)
plt.plot(profile_corner)
```

```
(array([2.38473877]), array([69.36492032]), array([13.796978]), arr
ay([16.18171677]))
```

Out[6]:

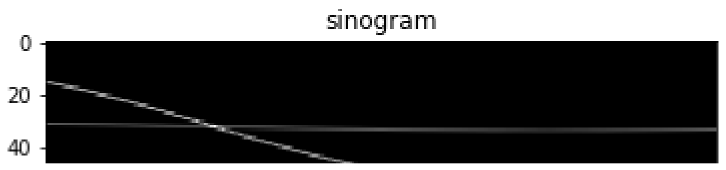
```
[<matplotlib.lines.Line2D at 0x7f1c28a7b550>]
```



In [56]:

```python
#c (including b)
N= [256,128,64,32,16,8]
wct=[]; wcn=[];
for n in N:
    theta = np.linspace(0,180,n)
    img = st.radon(obj, theta)
    plt.figure()
    plt.imshow(img,cmap='gray')
    plt.title('sinogram')
#     plt.show()

#     iradon = st.iradon(img,theta,filter=None)
    iradon = st.iradon(img,theta,filter=None)

    plt.figure()
    plt.imshow(iradon,cmap='gray')
    plt.title('BP')
#     plt.show()

    profile_center = iradon[:,31]
    width_center = width(profile_center,31)
    plt.figure()
    plt.plot(profile_center)
    plt.plot(31,profile_center[31])
    plt.hlines(width_center[1], width_center[2], width_center[3]

    profile_corner = iradon[:,15]
    width_corner = width(profile_corner,15)
    plt.figure()
    plt.plot(profile_corner)
    plt.plot(15,profile_corner[15])
    plt.hlines(width_corner[1], width_corner[2], width_corner[3]
    print('N: ',str(n),', peak value at corner:',iradon[15,15],'

```

```
e.max_open_warning`).
  from ipykernel import kernelapp as app
/home/gangwon/anaconda3/lib/python3.7/site-packages/ipykernel_lau
ncher.py:22: RuntimeWarning: More than 20 figures have been opene
d. Figures created through the pyplot interface (`matplotlib.pypl
ot.figure`) are retained until explicitly closed and may consume
too much memory. (To control this warning, see the rcParam `figur
e.max_open_warning`).
/home/gangwon/anaconda3/lib/python3.7/site-packages/ipykernel_lau
ncher.py:29: RuntimeWarning: More than 20 figures have been opene
d. Figures created through the pyplot interface (`matplotlib.pypl
ot.figure`) are retained until explicitly closed and may consume
too much memory. (To control this warning, see the rcParam `figur
e.max_open_warning`).
```



Generally, the width increases as the number of views decreases. By sampling enough, the point to-be-estimated becomes more localized. The values at the points become more deviated as the number of views decreases. Likewise, under-determined system with a few views may yield less accurate solution.

In [59]:

```
#d (including b)
N= [256,128,64,32,16,8]
wct=[]; wcn=[];
for n in N:
    theta = np.linspace(0,180,n)
    img = st.radon(obj, theta)
    plt.figure()
    plt.imshow(img,cmap='gray')
    plt.title('sinogram')
#     plt.show()

#     iradon = st.iradon(img,theta,filter=None)
    iradon = st.iradon(img,theta,filter='ramp')

    plt.figure()
    plt.imshow(iradon,cmap='gray')
    plt.title('BP')
#     plt.show()

    profile_center = iradon[:,31]
    width_center = width(profile_center,31)
    plt.figure()
    plt.plot(profile_center)
    plt.plot(31,profile_center[31])
    plt.hlines(width_center[1], width_center[2], width_center[3]

    profile_corner = iradon[:,15]
    width_corner = width(profile_corner,15)
    plt.figure()
    plt.plot(profile_corner)
    plt.plot(15,profile_corner[15])
    plt.hlines(width_corner[1], width_corner[2], width_corner[3]
    print('N: ',str(n),', peak value at corner:',iradon[15,15],'

```

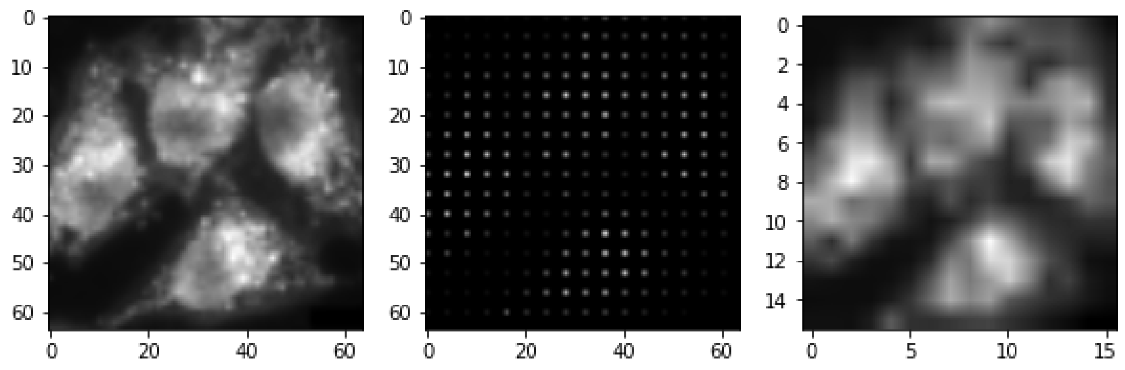
```
N:  256 , peak value at corner: 55.745188615899785 , peak width a
t corner: [1.38430638] , peak value at center:  25.93417626126006
, peak width at corner: [1.62446058]
N:  128 , peak value at corner: 56.11857138580631 , peak width at
corner: [1.38331912] , peak value at center:  26.000173489126098
, peak width at corner: [1.6212462]
N:  64 , peak value at corner: 55.50480358440926 , peak width at
corner: [1.48279811] , peak value at center:  26.16596540040996 ,
peak width at corner: [1.64747854]
N:  32 , peak value at corner: 60.004825303687475 , peak width at
corner: [1.46980989] , peak value at center:  26.377474105320307
, peak width at corner: [1.63426961]
N:  16 , peak value at corner: 59.85050229080789 , peak width at
corner: [1.69109141] , peak value at center:  25.825790384774788
, peak width at corner: [1.79947563]
N:  8 , peak value at corner: 65.11349556088027 , peak width at c
orner: [2.07854235] , peak value at center:  28.749467174723357 ,
peak width at corner: [1.51434154]
```

Width are sharper than the backprojection cases for the same number of views because of filtering-effects of ramp function in the freq-domain. The values at the points are diminished because of filtering effect. It should be noted that the point object has lots of low-frequency components. Even though the ramp filter is derived analytically in the inversion process, the filtering-effect may harm significantly the object components that contain low-frequency components.

5

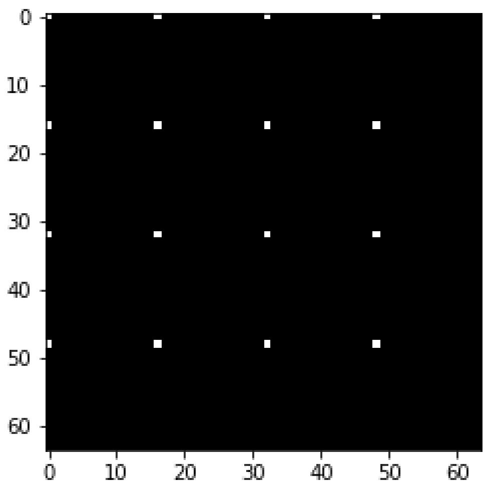
In [9]:
```
#Q. 5
f= imageio.imread('../BIOE580_hw01_data/BIOE580_hw01_data/hw01_e
# np.shape(f) 64 X 64
# a
comb = np.zeros([64,64])
comb[::4,::4] = 1
combf = f * comb
obj_d = combf[::4,::4]
figure,ax = plt.subplots(1,3,figsize=(10,10))
ax[0].imshow(f, cmap='gray', interpolation='bilinear')
ax[1].imshow(combf,cmap='gray',interpolation='bilinear')
ax[2].imshow(obj_d,cmap='gray',interpolation='bilinear')
```

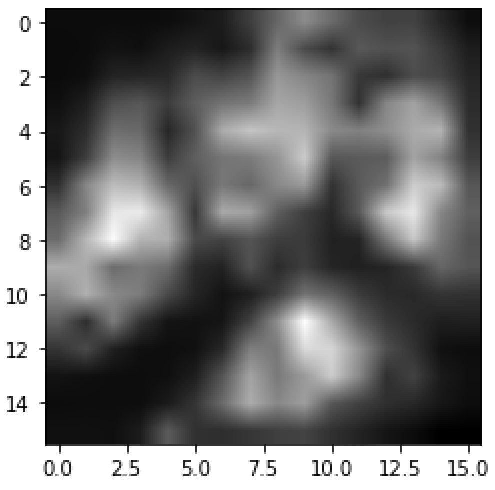
Out[9]:
```
<matplotlib.image.AxesImage at 0x7f1c28854ed0>
```



In [10]:
```
# b
img = np.fft.fft2(obj_d)
```

In [11]:
```
# c
fcomb = np.fft.fft2(comb)
```

In [12]:
```
plt.imshow(np.abs(np.fft.fftshift(fcomb)), cmap='gray')#, interp
```
Out[12]:
```
<matplotlib.image.AxesImage at 0x7f1c28be4090>
```


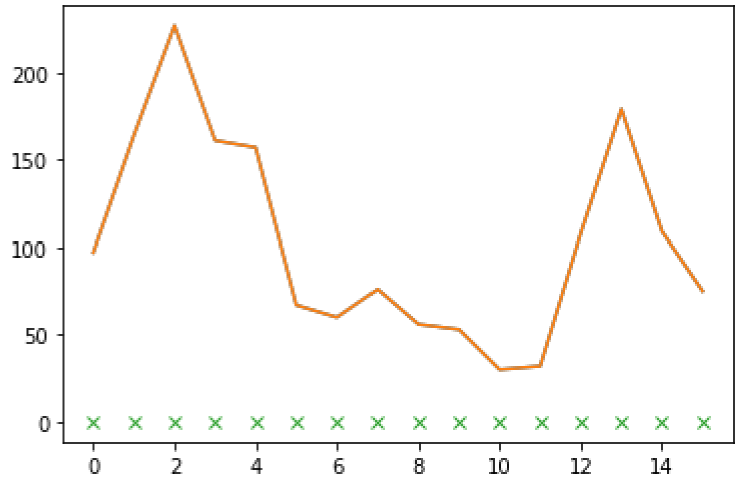
In [13]:
```
#d
img_diminished = ss.convolve2d(img,fcomb,'same')
```
In [14]:
```
#e
inv_diminished = np.fft.ifft2(img_diminished)/(16* 16)
```
In [15]:
```
plt.imshow(np.abs(inv_diminished),cmap='gray',interpolation='bil
```
Out[15]:
```
<matplotlib.image.AxesImage at 0x7f1c287fde90>
```


In [16]:

```
objv =f[::4,::4]
invv = np.abs(inv_diminished)
plt.plot(objv[8,:])
plt.plot(invv[8,:])
plt.plot(objv[8,:]-invv[8,:],'x')
```

Out[16]:

```
[<matplotlib.lines.Line2D at 0x7f1c2869c290>]
```



In this problem, we sampled the object at each 4th point using comb function. The FT of the comb function is sum of phase-shifted exponential functions with a period of 1/T in the freq domain. The FT of comb function does not involve aliasing because the envelop at each point in the freq domain does not overlapped with the adjacent points. So, the sampled object and the inverse FT of image (FT of sampled object) are the same

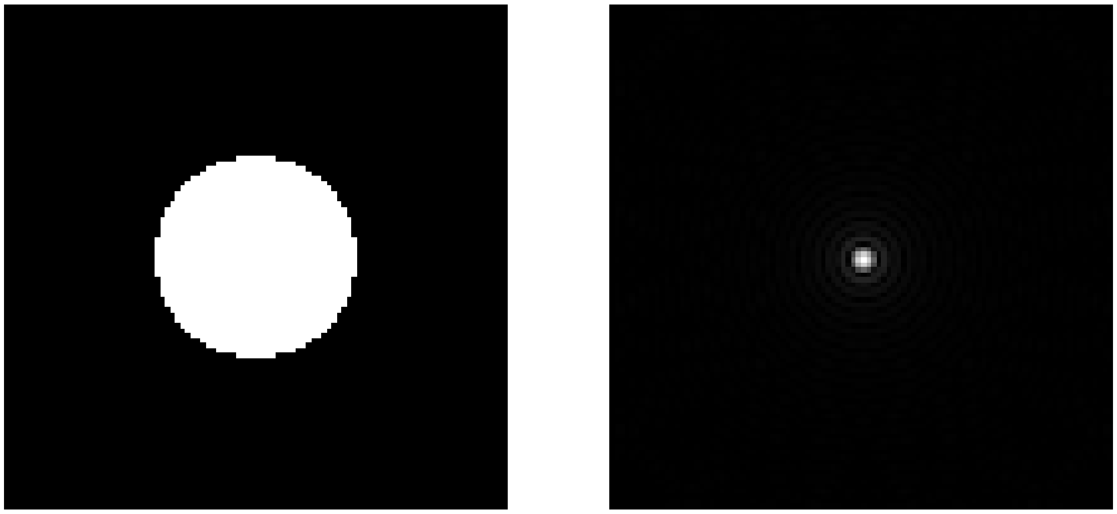
6.

In [43]:

```
def circ(r,N,dx):
    grid = np.linspace(-(N-1)/2*dx,(N-1)/2*dx,N)
#     print(grid)
    xx, yy = np.meshgrid(grid,grid)
    disk = np.sqrt(np.array(xx**2+yy**2))
    disk[np.where(disk>r)] = 0
    disk[np.where(disk!=0)] = 1
    return disk
```

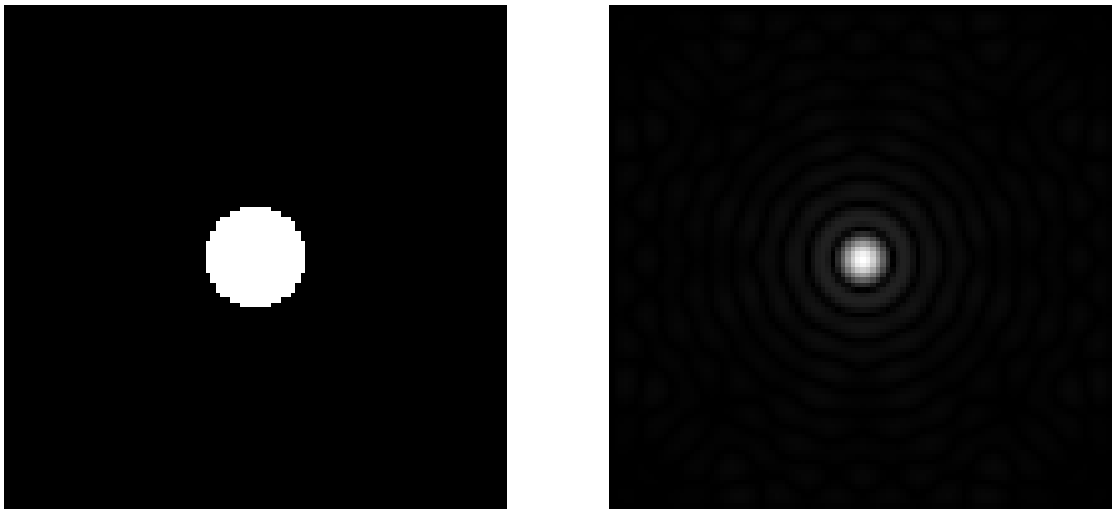
In [44]:

```python
figure,ax = plt.subplots(1,2,figsize=(10,10))
apert= circ(2,100,0.1)
img = np.abs(np.fft.fftshift(np.fft.fft2(apert)))
shows = [apert, img]
for i in range(2):
    ax[i].imshow(shows[i],cmap='gray')
    ax[i].axis('off')
```



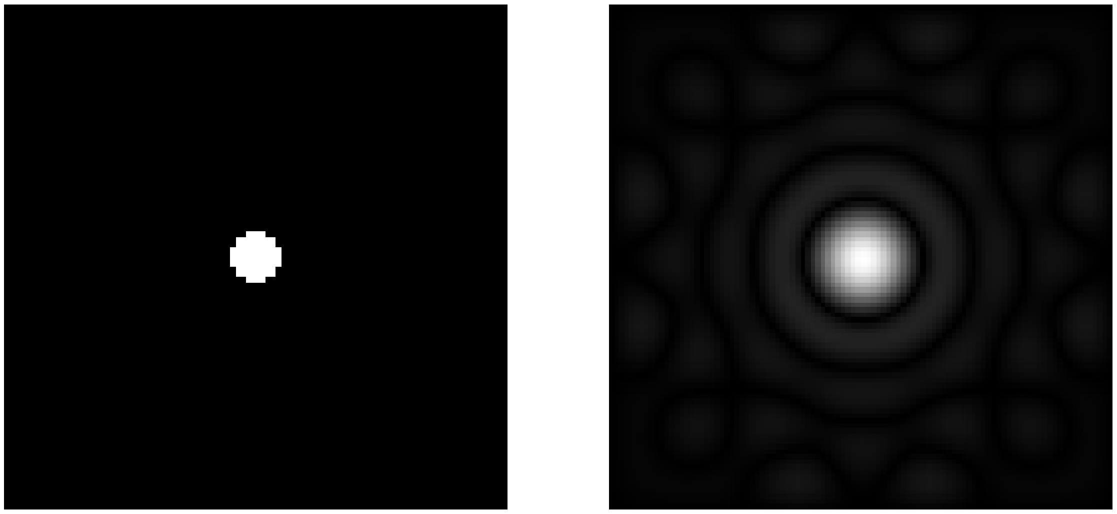
In [45]:

```python
figure,ax = plt.subplots(1,2,figsize=(10,10))
apert= circ(1,100,0.1)
img = np.abs(np.fft.fftshift(np.fft.fft2(apert)))
shows = [apert, img]
for i in range(2):
    ax[i].imshow(shows[i],cmap='gray')
    ax[i].axis('off')
```



In [46]:

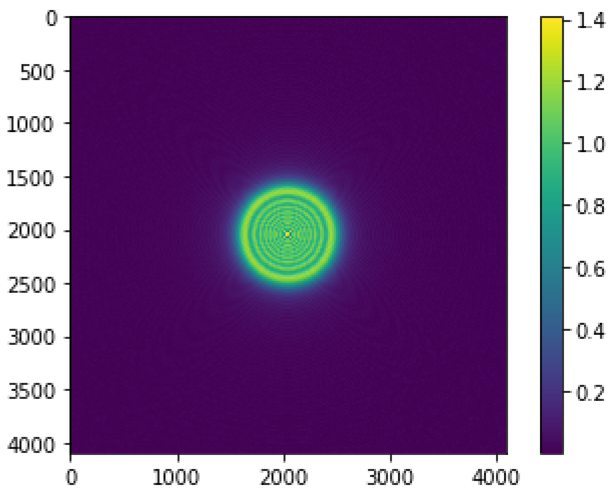
```python
figure,ax = plt.subplots(1,2,figsize=(10,10))
apert= circ(0.5,100,0.1)
img = np.abs(np.fft.fftshift(np.fft.fft2(apert)))
shows = [apert, img]
for i in range(2):
    ax[i].imshow(shows[i],cmap='gray')
    ax[i].axis('off')
```



When the object is broader when a is large, bandwidth of the spatial frequency become narrower. This is reflected in the visualization (Left: object, Right: its reciprocal domain). Just note that the object in the frequency domain (Right) is also circular as described in the hand-written paper. Also, the results are consistent with the uncertainty principle.

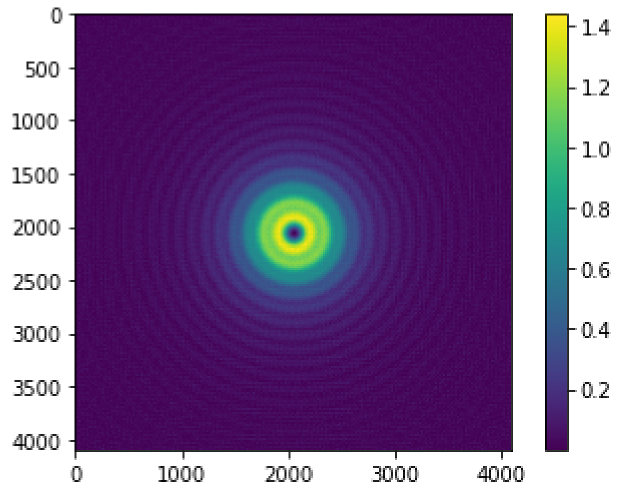
In [47]:
```
## c and d
```

In [48]:
```
# part-c
# create a meshgrid with pixel size 2e-6
N=2**(12)
r=1e-3 # 1mm
dx=2e-6
apert = circ(r, N, dx)
dk = np.linspace(-(1 - 1/N)/(2*dx),(1-1/N)/(2*dx),N)
kx,ky = np.meshgrid(dk,dk)
FTapert = np.fft.fft2(apert)

wavelength = 5e-7 # 500nm
k = 2*np.pi/wavelength
distance = 0.1 # 10cm
H = np.exp(-1j*np.pi*wavelength*distance*(kx**2+ky**2))*np.exp(1
H = np.fft.fftshift(H)

Hf = H*FTapert
InvHf = np.fft.ifft2(Hf)

plt.figure()
plt.imshow(np.abs(InvHf))
plt.colorbar()
plt.show()
```



In [49]:

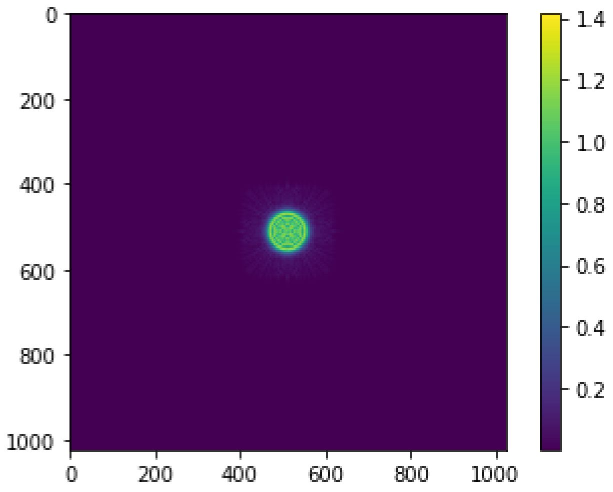
```python

apert = circ(r, N, dx)
dk = np.linspace(-(1 - 1/N)/(2*dx),(1-1/N)/(2*dx),N)
kx,ky = np.meshgrid(dk,dk)
FTapert = np.fft.fft2(apert)

wavelength = 5e-7 # 500nm
k = 2*np.pi/wavelength
distance = 1 # 10cm
H = np.exp(-1j*np.pi*wavelength*distance*(kx**2+ky**2))*np.exp(1
H = np.fft.fftshift(H)

Hf = H*FTapert
InvHf = np.fft.ifft2(Hf)

plt.figure()
plt.imshow(np.abs(InvHf))
plt.colorbar()
plt.show()
```



The results are very intuituve. If the distance between detector and aperture become larger, the wave transmitted through the edge of aperture are more likely to deviate from the center point of detector, which makes larger diffraction pattern in the detector.

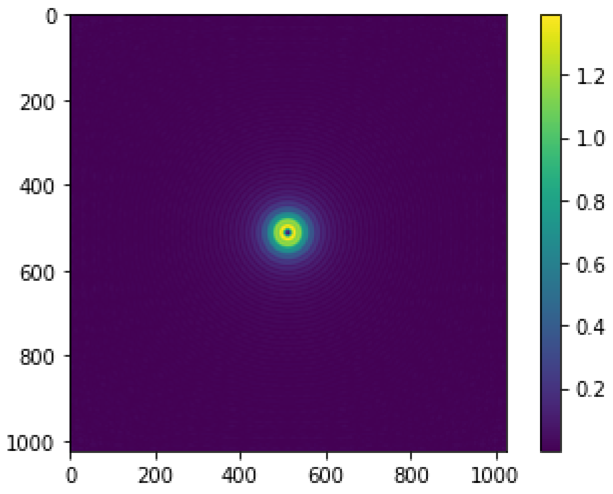
In [50]:

```
# part-d
# create a meshgrid with pixel size 2e-5 on 1024X 1024 grid
N=2**(10)
r=1e-3 # 1mm
dx=2e-5
apert = circ(r, N, dx)
dk = np.linspace(-(1 - 1/N)/(2*dx),(1-1/N)/(2*dx),N)
kx,ky = np.meshgrid(dk,dk)
FTapert = np.fft.fft2(apert)

wavelength = 5e-7 # 500nm
k = 2*np.pi/wavelength
distance = 0.1 # 10cm
H = np.exp(-1j*np.pi*wavelength*distance*(kx**2+ky**2))*np.exp(1
H = np.fft.fftshift(H)

Hf = H*FTapert
InvHf = np.fft.ifft2(Hf)

plt.figure()
plt.imshow(np.abs(InvHf))
plt.colorbar()
plt.show()
```



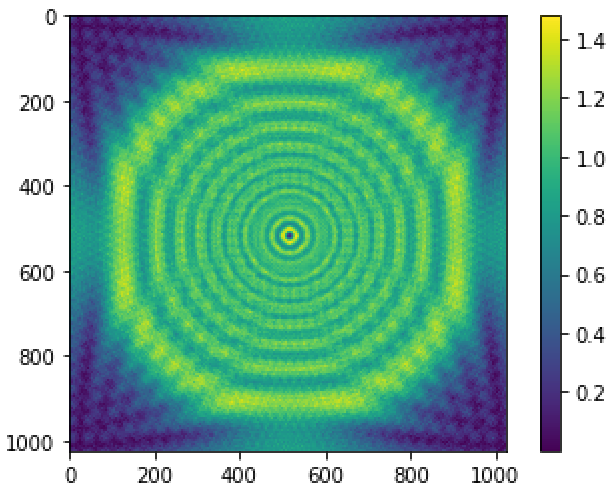
In [51]:

```
# part-d
# create a meshgrid with pixel size 2e-5 on 1024X 1024 grid

apert = circ(r, N, dx)
dk = np.linspace(-(1 - 1/N)/(2*dx),(1-1/N)/(2*dx),N)
kx,ky = np.meshgrid(dk,dk)
FTapert = np.fft.fft2(apert)

wavelength = 5e-7 # 500nm
k = 2*np.pi/wavelength
distance = 1 # 10cm
H = np.exp(-1j*np.pi*wavelength*distance*(kx**2+ky**2))*np.exp(1
H = np.fft.fftshift(H)

Hf = H*FTapert
InvHf = np.fft.ifft2(Hf)

plt.figure()
plt.imshow(np.abs(InvHf))
plt.colorbar()
plt.show()
```



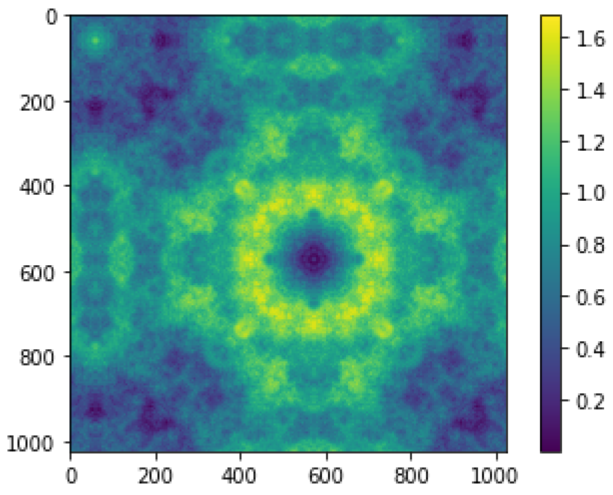
In [53]:

```
# part-d Recompute part c on 1024X 1024 with smaller pixel size
# create a meshgrid with pixel size 2e-5 on 1024X 1024 grid
N=2**(10)
r=1e-3 # 1mm
dx=2e-6
apert = circ(r, N, dx)
dk = np.linspace(-(1 - 1/N)/(2*dx),(1-1/N)/(2*dx),N)
kx,ky = np.meshgrid(dk,dk)
FTapert = np.fft.fft2(apert)

wavelength = 5e-7 # 500nm
k = 2*np.pi/wavelength
distance = 0.1 # 10cm
H = np.exp(-1j*np.pi*wavelength*distance*(kx**2+ky**2))*np.exp(1
H = np.fft.fftshift(H)

Hf = H*FTapert
InvHf = np.fft.ifft2(Hf)

plt.figure()
plt.imshow(np.abs(InvHf))
plt.colorbar()
plt.show()
```



In [54]:

```
# part-d Recompute part c on 1024X 1024 with smaller pixel size
# create a meshgrid with pixel size 2e-5 on 1024X 1024 grid
N=2**(10)
r=1e-3 # 1mm
dx=2e-6
apert = circ(r, N, dx)
dk = np.linspace(-(1 - 1/N)/(2*dx),(1-1/N)/(2*dx),N)
kx,ky = np.meshgrid(dk,dk)
FTapert = np.fft.fft2(apert)

wavelength = 5e-7 # 500nm
k = 2*np.pi/wavelength
distance = 1 # 10cm
H = np.exp(-1j*np.pi*wavelength*distance*(kx**2+ky**2))*np.exp(1
H = np.fft.fftshift(H)

Hf = H*FTapert
InvHf = np.fft.ifft2(Hf)

plt.figure()
plt.imshow(np.abs(InvHf))
plt.colorbar()
plt.show()
```



Effect of discretization

The diffraction effect becomes larger when the pixel-size is small. This is consistent with the physics of diffraction pattern regarding the uncertainty principle