interferometric_images

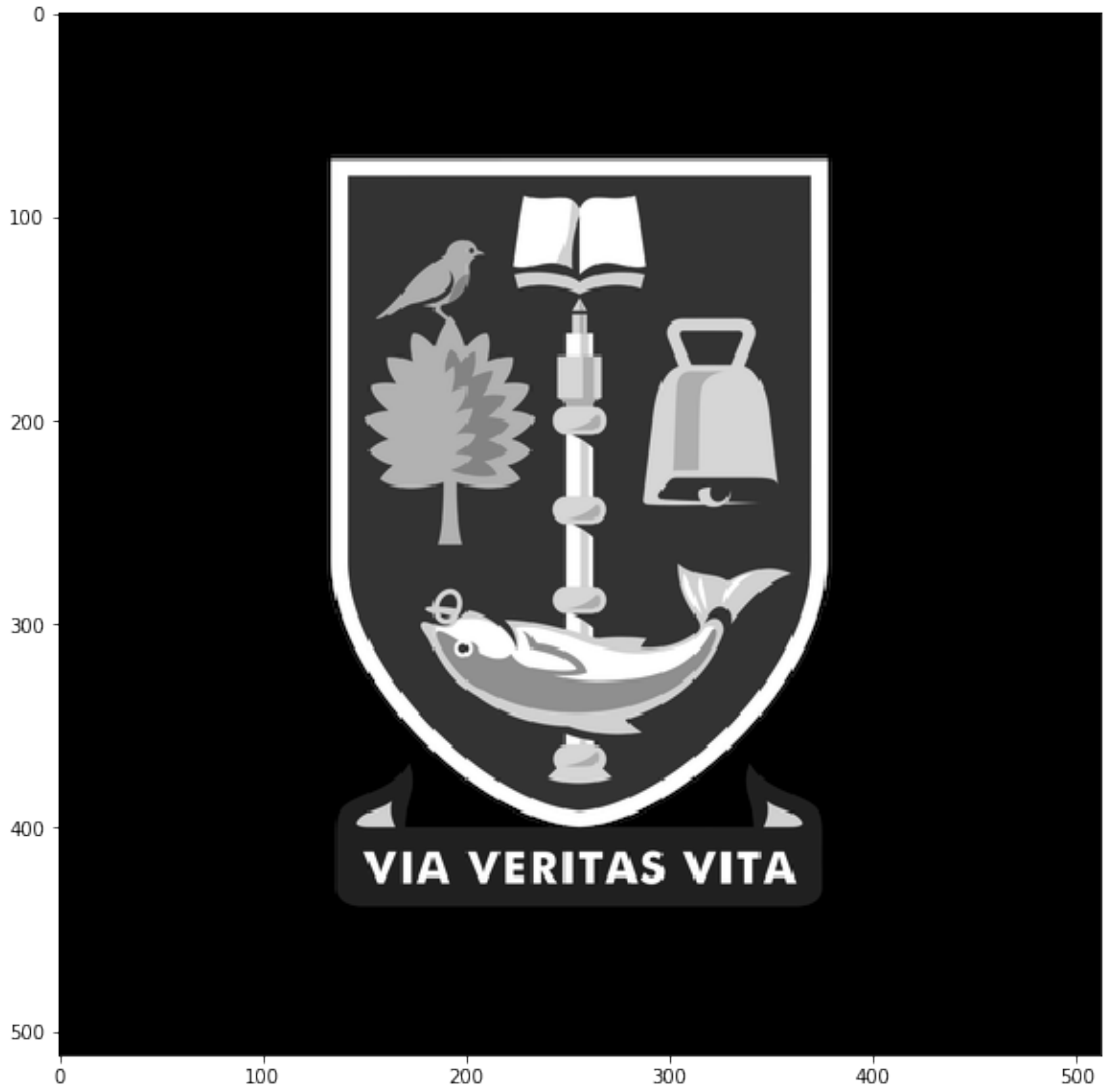
November 11, 2018

1 Images and spatial filtering in radio astronomy

This notebook explores some of the properties of the Fourier transforms of images in two dimensions. First we import some standard packages and read a reference image (the motto of the University of Glasgow).

In [1]:
```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

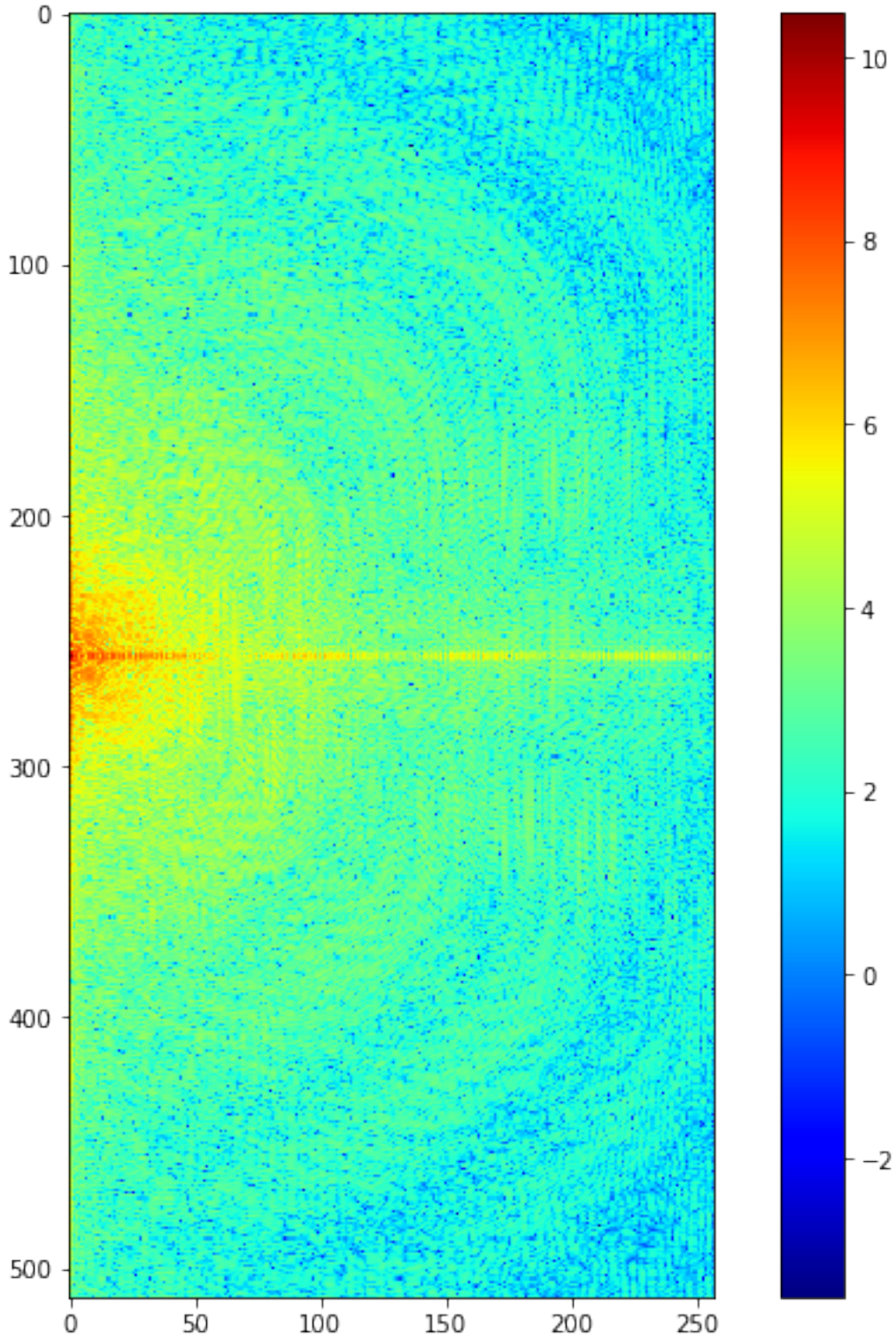
In [2]:
```
img=mpimg.imread('Uni_Glasgow_2017_arms.png')
logo = img[:,:,0]

plt.rcParams["figure.figsize"] = (10,10)
plt.imshow(logo,cmap = 'gray');
```



Now, let's look at the Fourier transform of this image. The image is real, so the transform is Hermitian conjugate, i.e. $F(\mathbf{r})^* = F(-\mathbf{r})$. We therefor only need to plot half the transform plane. Python has a useful version of the FFT specifically for real functions, `rfft2`. Also the transform will be complex, so we'll just show its modulus.

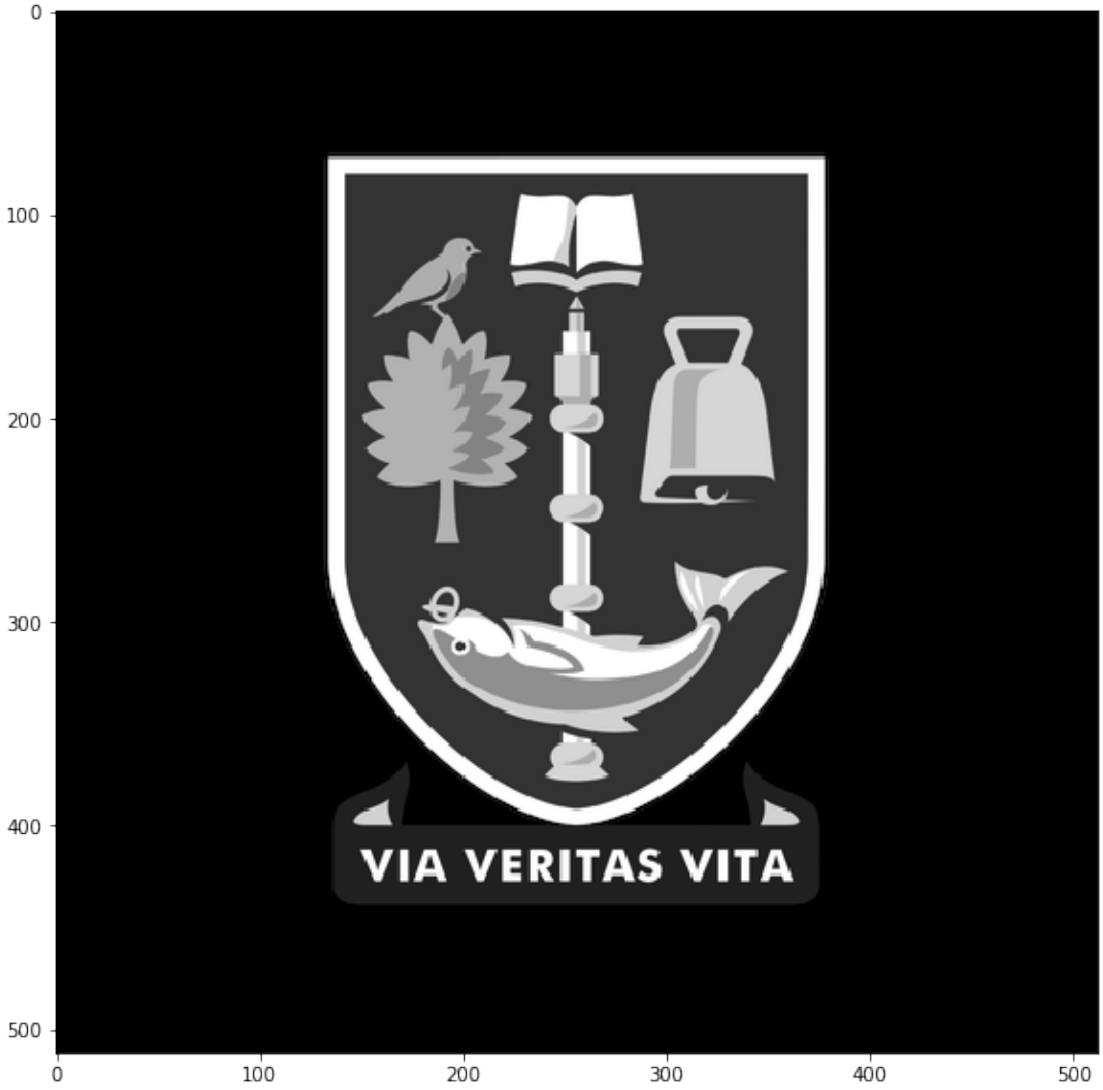
The transforms of simple images like this tend to have lots of power close to the origin (representing the large-scale structure in the image) and rather less towards the edges. For that reason we plot the log of the modulus.

```
In [3]: F = np.fft.rfft2(logo)
        P = np.abs(F)
        plt.imshow(np.fft.fftshift(np.log(P),0),cmap = 'jet')
        plt.colorbar();
```



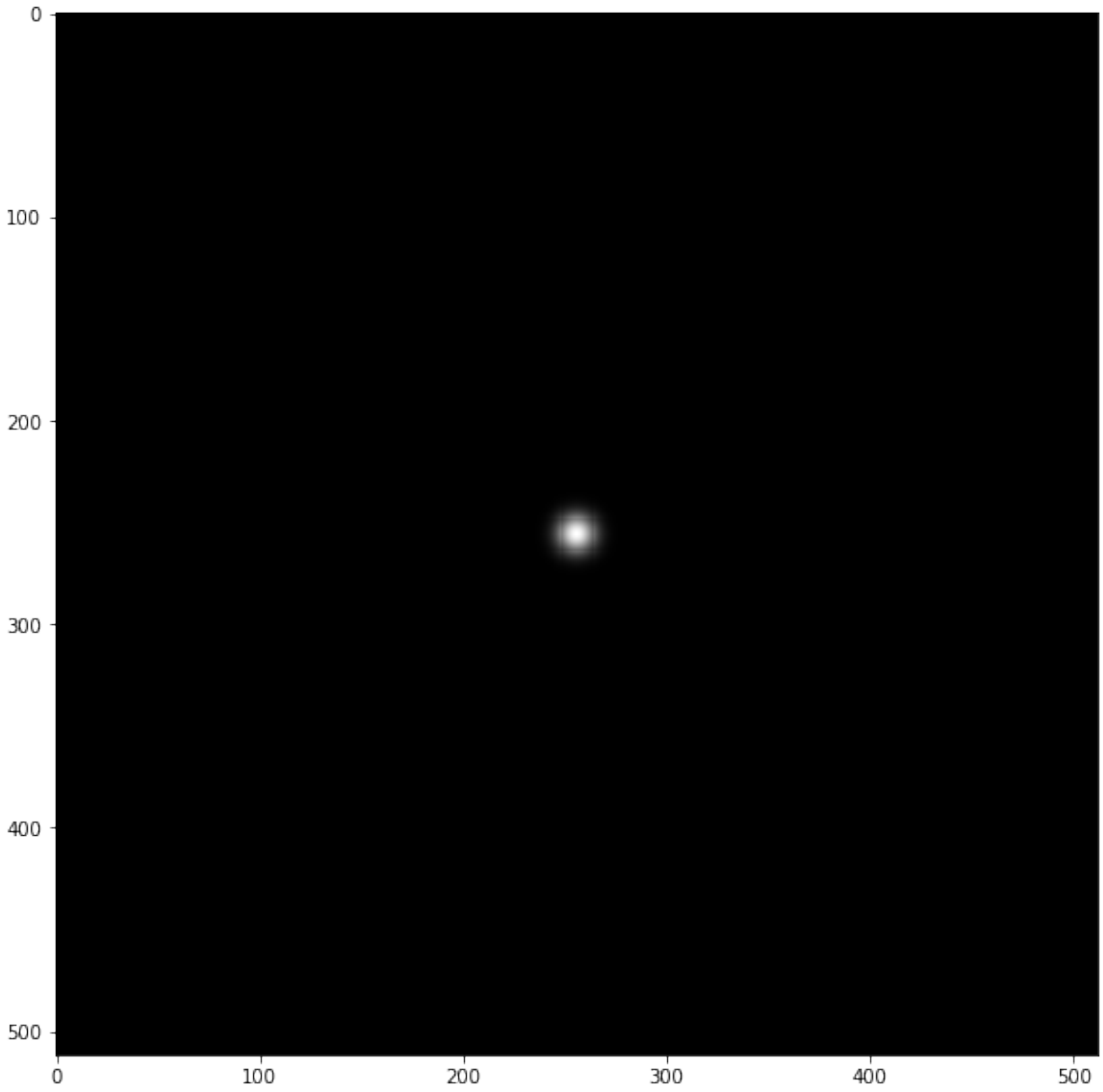
We know that no information is lost in doing this transform. Just to prove it, here is the inverse transform, reconstructing the original image.

```
In [4]: recon = np.fft.irfft2(F)
        plt.imshow(recon,cmap = 'gray');
```



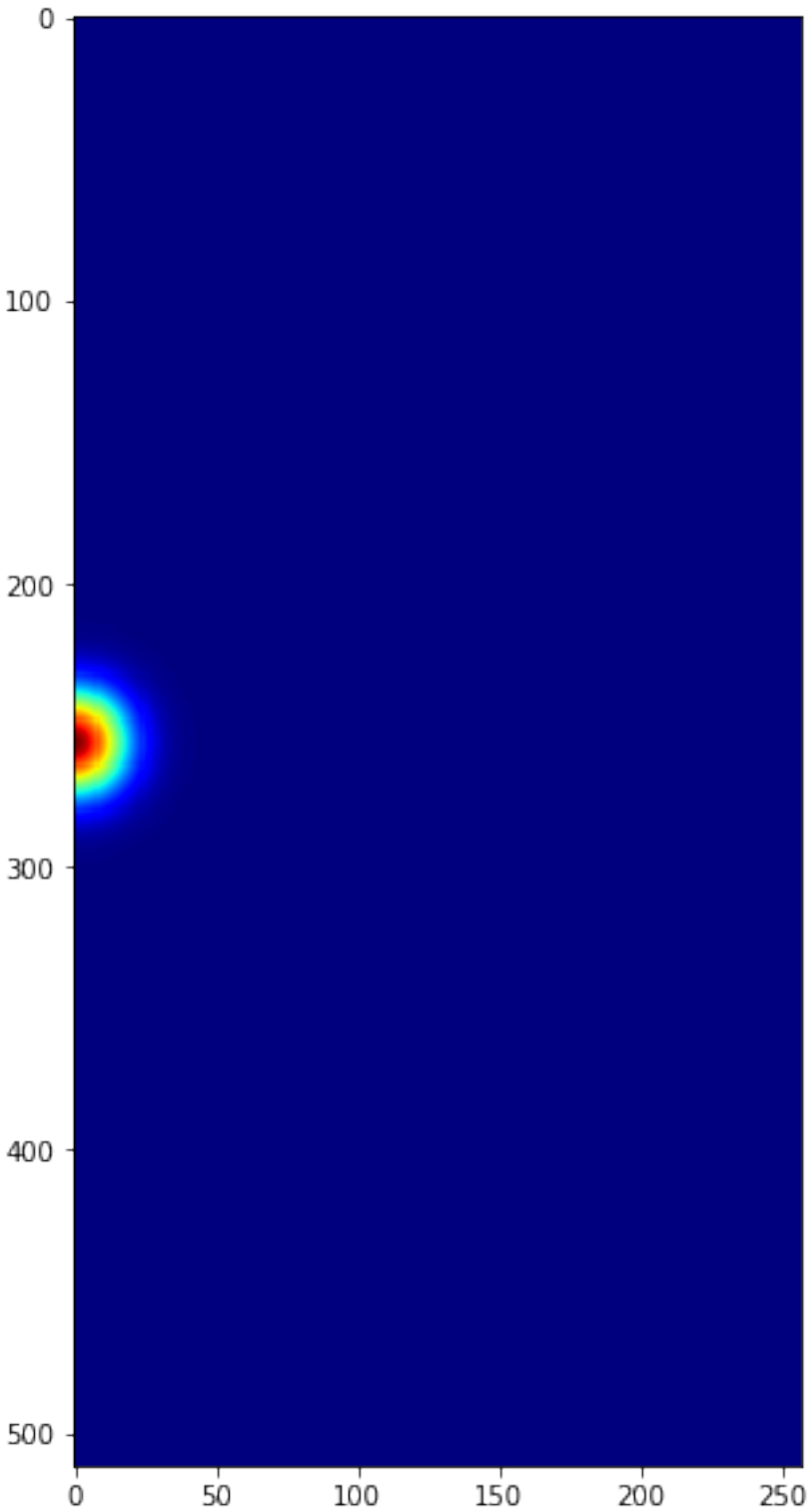
Now we create a two-dimensional gaussian 'beam' of width σ, and display it for reference.

```
In [5]: x, y = np.meshgrid(np.linspace(-1,1,512), np.linspace(-1,1,512))
        d = np.sqrt(x*x+y*y)
        sigma = 0.025
        beam = np.exp(-( d**2 / ( 2.0 * sigma**2 ) ) )

        plt.imshow(beam, cmap = 'gray');
```



The transform of this beam is of course another gaussian:

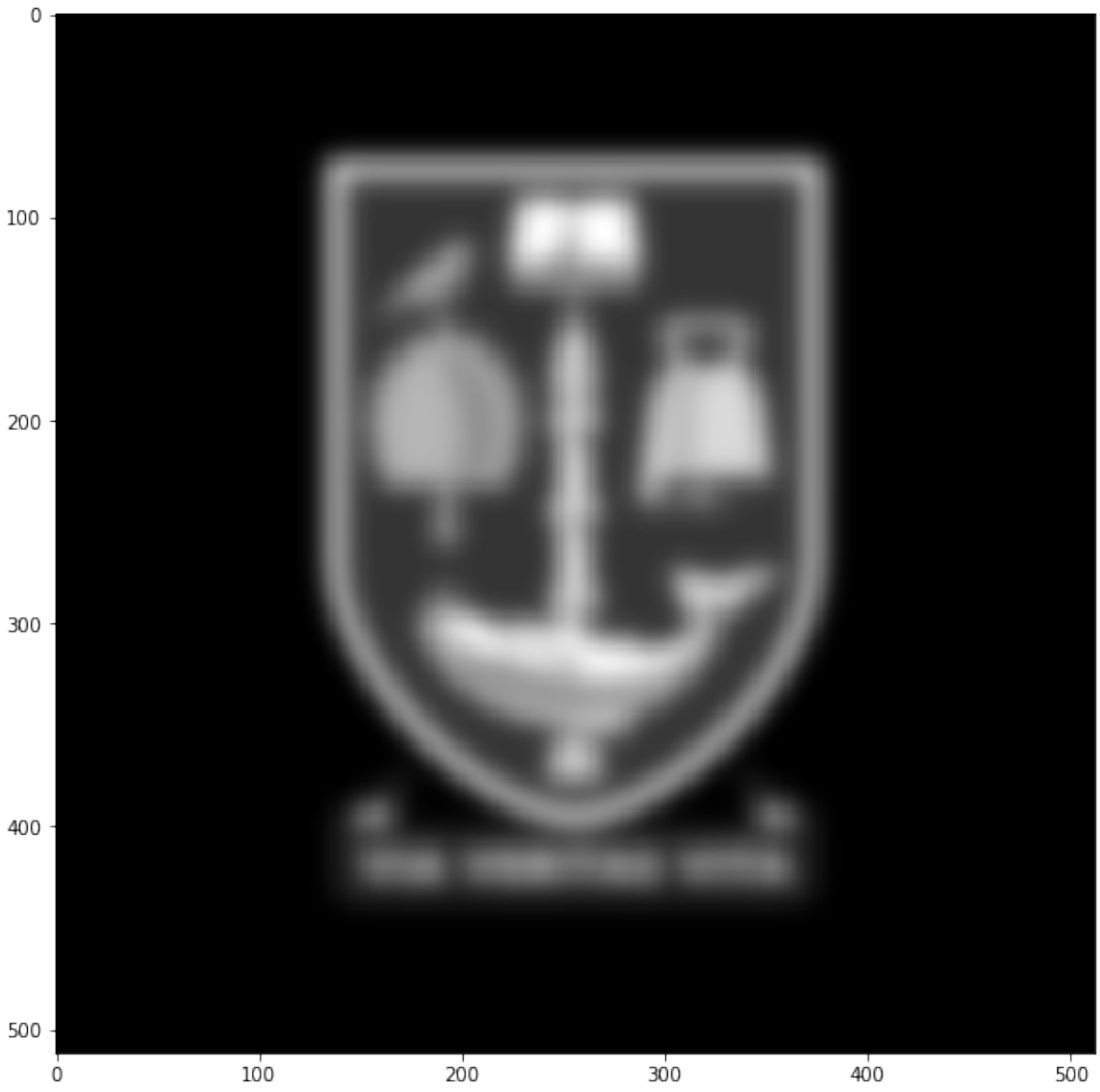
```
In [6]: tbeam = abs(np.fft.rfft2(beam))
        tbeam = tbeam/np.max(tbeam)
        plt.imshow(np.fft.fftshift(tbeam,0), cmap = 'jet');
```



We will multiply the transform of the image and the transform of the beam and then take the inverse transform of this product. The result is the same as *convolving* the image with the beam in

the image plane. We should get a blurred image:

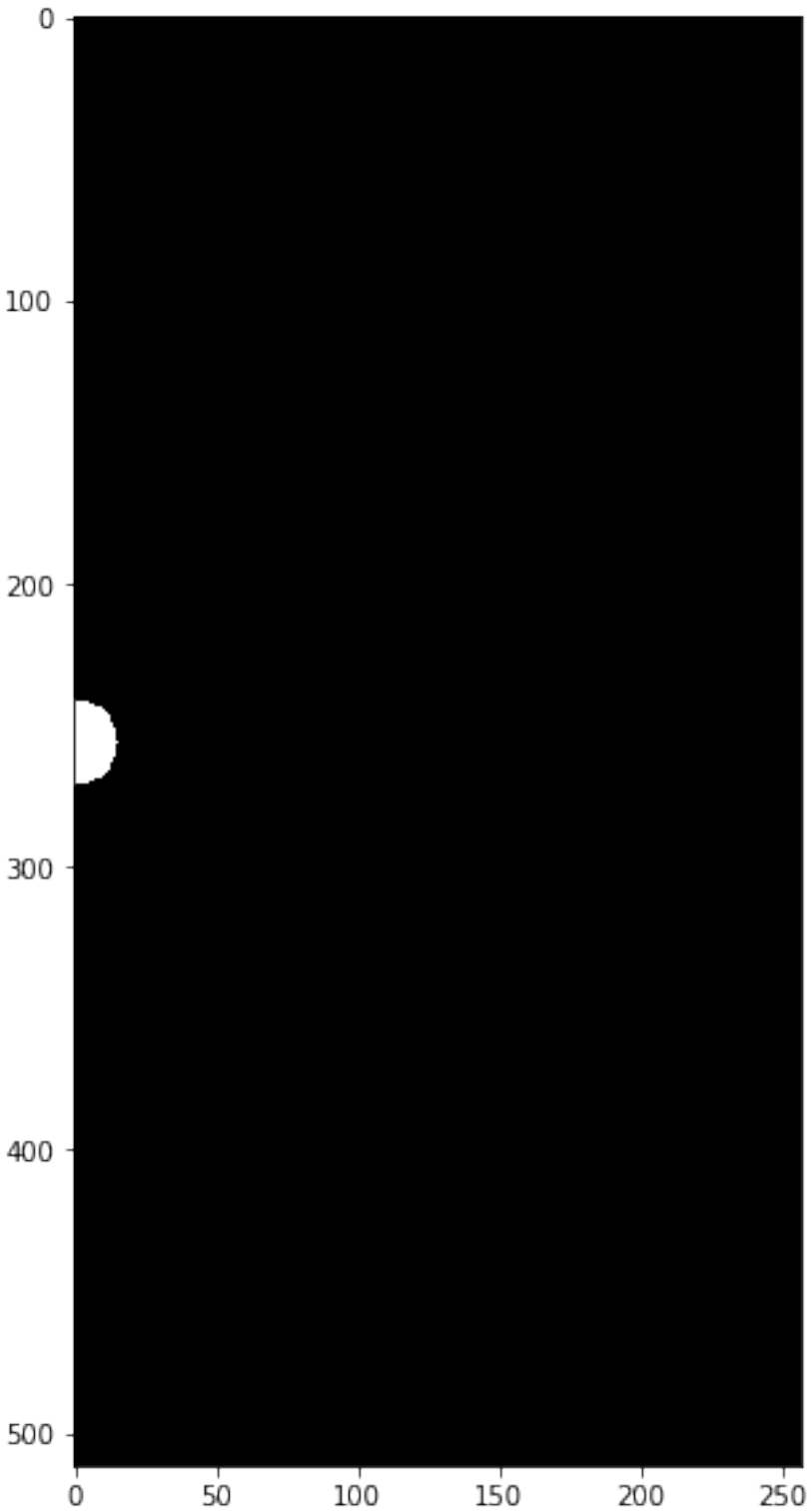
```
In [7]: recon = np.fft.irfft2(F*tbeam)
        plt.imshow((recon),cmap = 'gray');
```



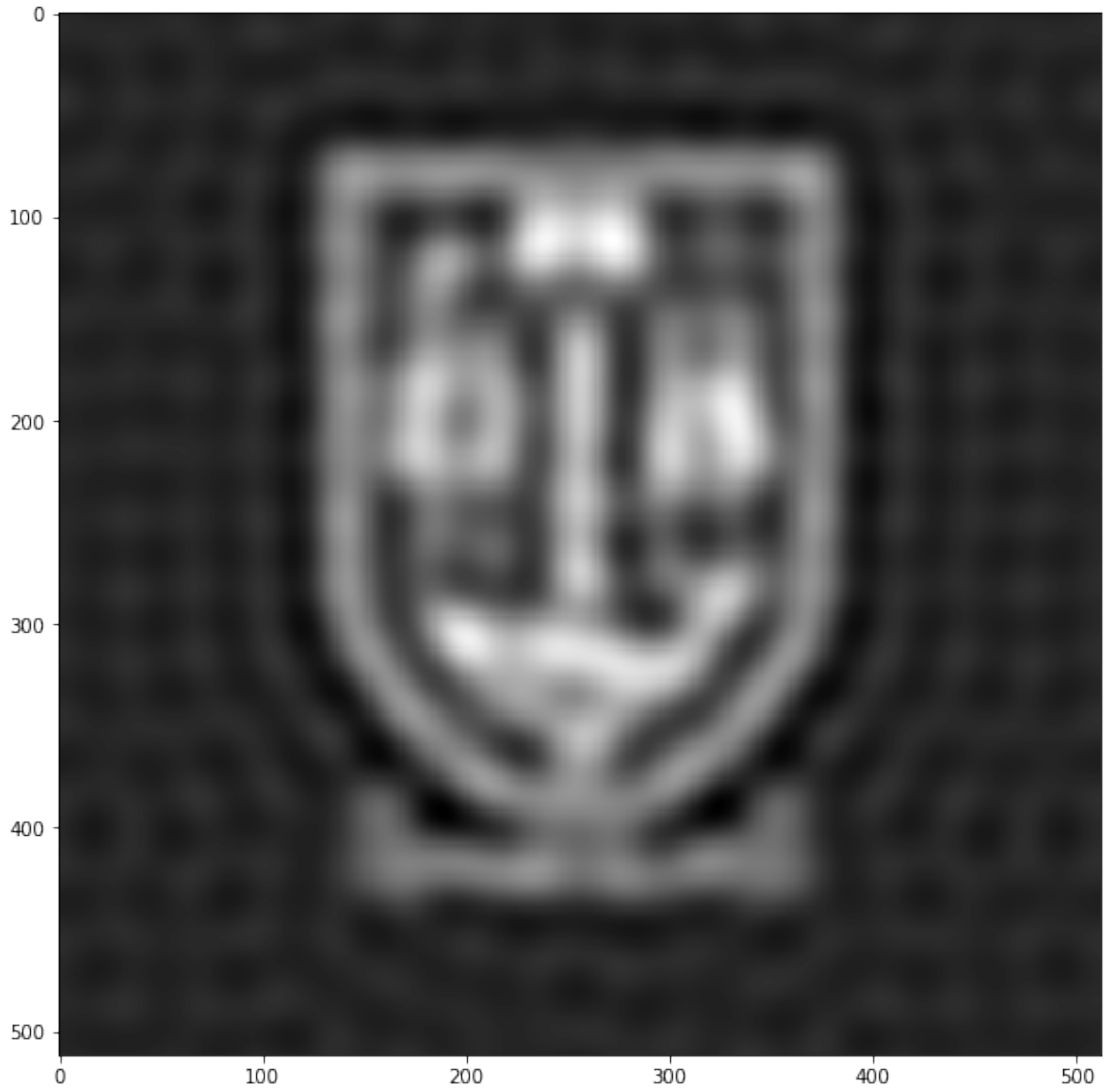
Try varying the width of the beam and see how the blurring changes. Convolutions are usually done through this (apparently) circuitous route because it's faster than a direct convolution in the image plane, especially for large images.

Now let's see what happens if the transform of the beam is a "top hat" rather an a gaussian. We'll threshhold the gaussian to generate this:

```
In [8]: top_hat = tbeam>1/2
        plt.imshow(np.fft.fftshift(top_hat,0),cmap = 'gray');
```

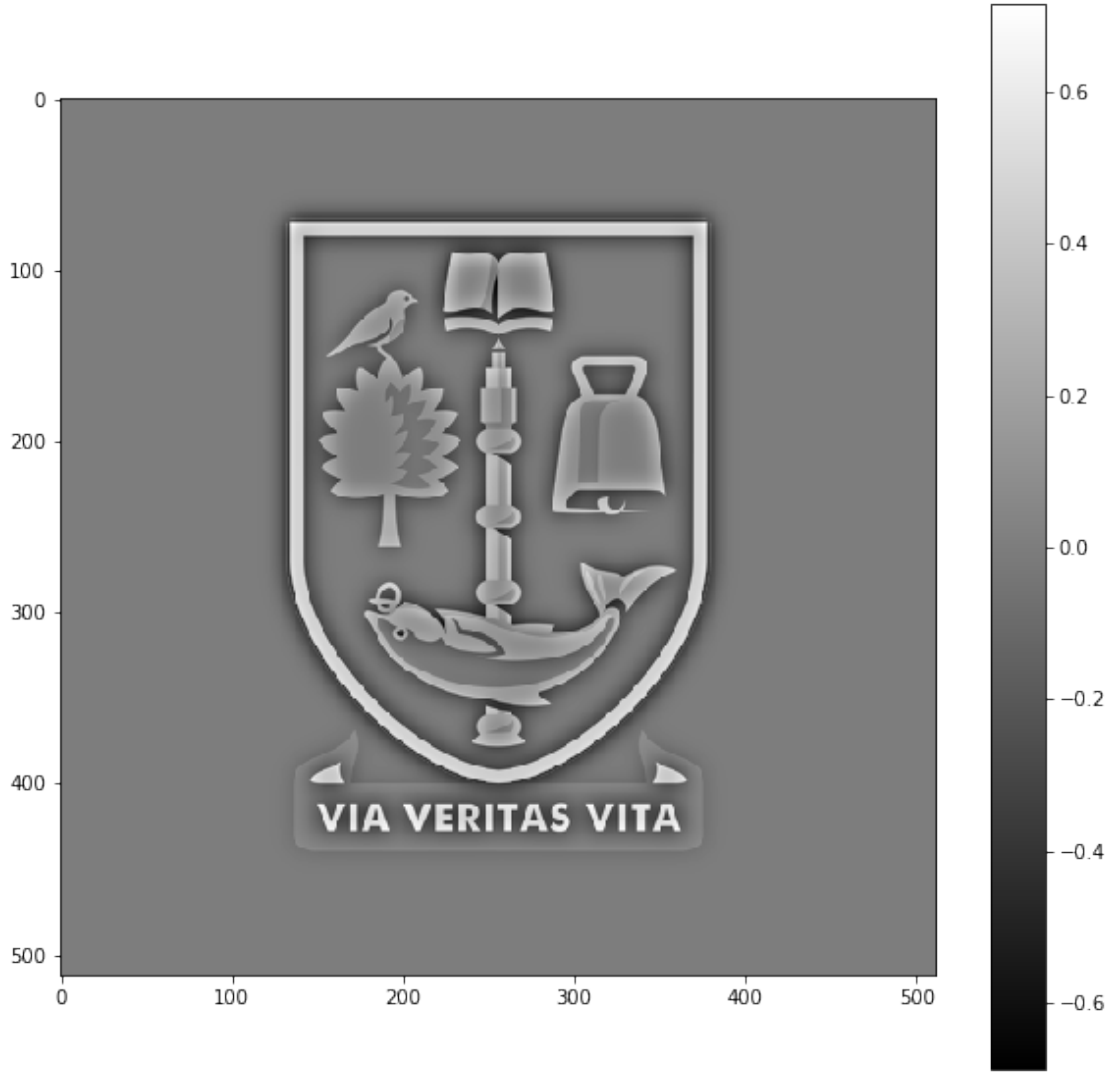



```
In [9]: recon = np.fft.irfft2(F*top_hat)
        plt.imshow(recon,cmap = 'gray');
```



We can see that the image is blurred, but there is also "ringing". The top hat acts as a sharp low-pass filter, removing spatial frequencies higher that the threshold. Let's see what the image looks like of we just let through the high spacial frequency components:

```
In [10]: recon = np.fft.irfft2(F*(1-tbeam))
         plt.imshow(recon,cmap = 'gray')
         plt.colorbar();
```

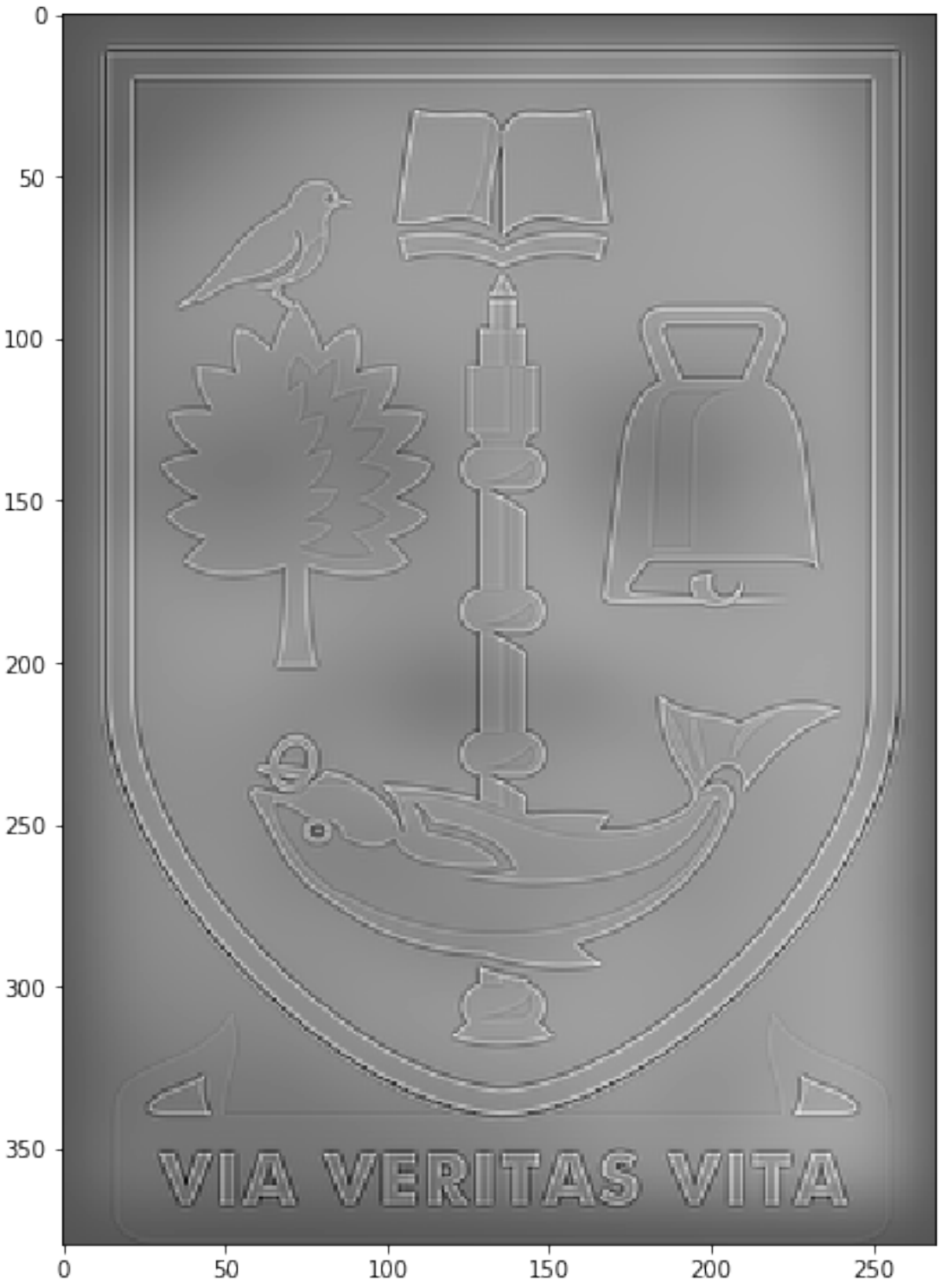


This image has a mean of zero because the zero-spatial-frequency component at the origin of its transform has been set to zero.

As an aside, if we combine the low spatial frequencies of one image with the high spatial frequency of another we get a composite image that looks different at different distances:

```
In [11]: face=mpimg.imread('face.jpg')
         face = face/face.max()
         F_p = np.fft.rfft2(face)

         sigma1 = 0.04
         beam1 = np.exp(-( d**2 / ( 2.0 * sigma1**2 ) ) )
         tbeam1 = abs(np.fft.rfft2(beam1))
         tbeam1 = tbeam1/np.max(tbeam1)
```

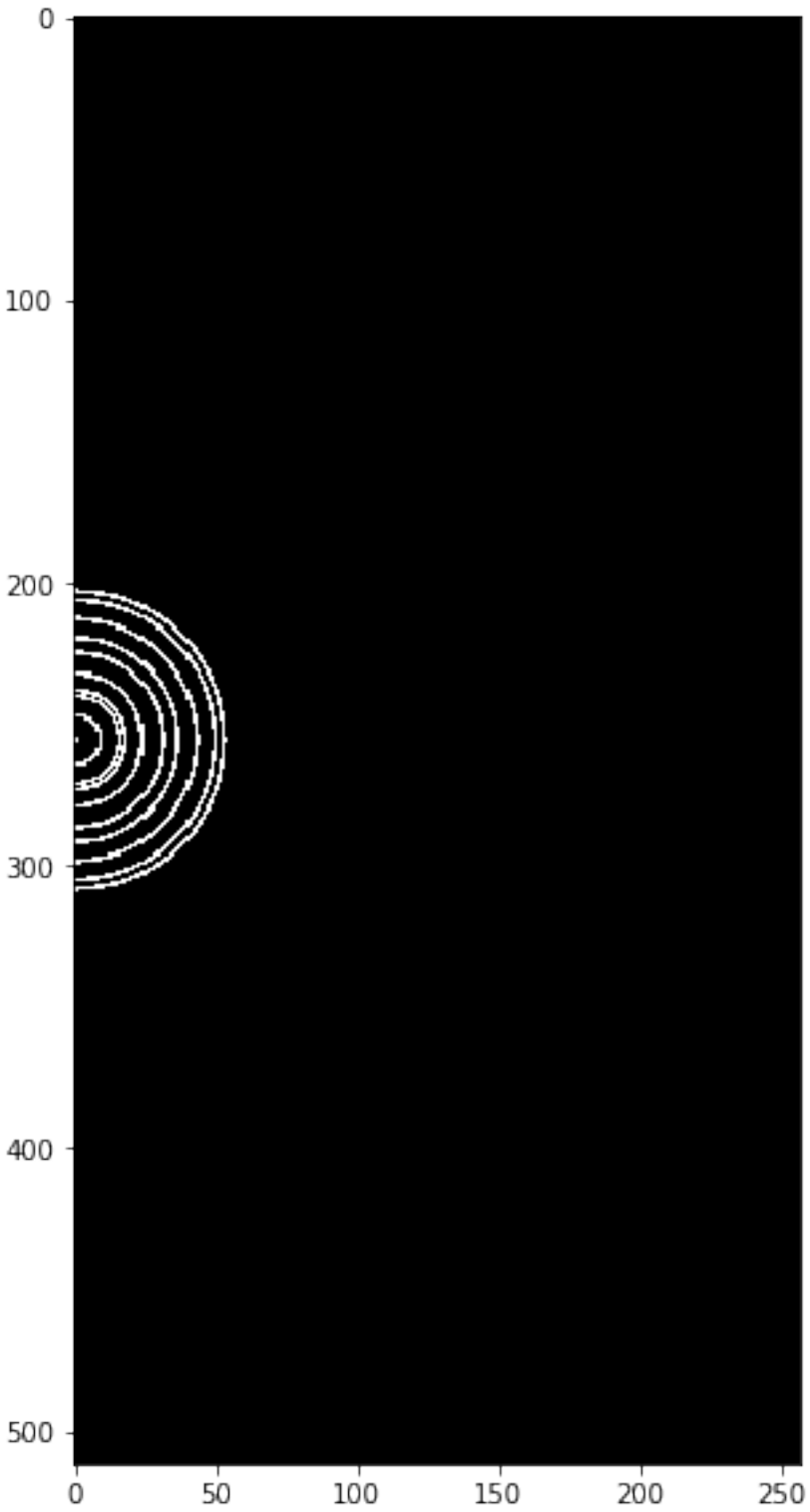
```
sigma2 = 0.0002
beam2 = np.exp(-( d**2 / ( 2.0 * sigma2**2 ) ) )
tbeam2 = abs(np.fft.rfft2(beam2))
tbeam2 = tbeam2/np.max(tbeam2)
tbeam2 = 1-tbeam2

recon = 0.2*np.fft.irfft2(F_p*tbeam1) + np.fft.irfft2(F*tbeam2)
plt.imshow(recon[60:440,120:390],cmap = 'gray');
```



In radio interferometry, the synthesised beam of the interferometer (i.e., the effective beam corresponding to the baselines used to generate the image) can be messy. Let's see what sort of a sythesised beam we get from concentric rings of measurements in the transform plane, from Earth

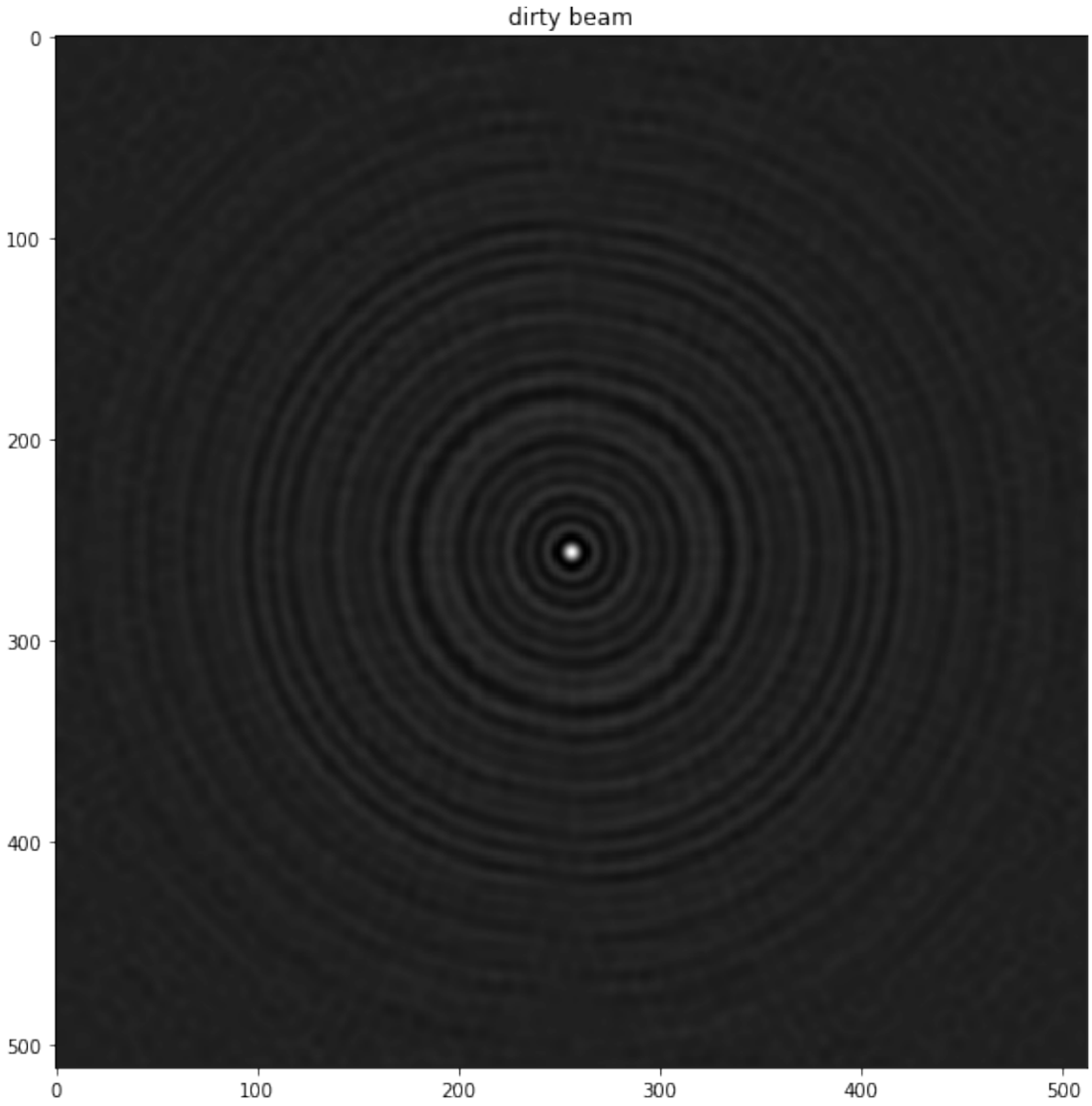
rotation aperture synthesis:

```
In [12]: rings = np.zeros((512,512))
         #radius = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7]
         radius = np.linspace(0,0.2,10) + np.random.rand(10) * 0.02
         radius[0]=0
         for r in radius:
             rings += (abs(d-r)<0.003 ).astype(float)

         aperture = np.fft.fftshift(rings[0:512, 255:512],0)

         plt.imshow(np.fft.fftshift(aperture,0), cmap = 'gray');
```



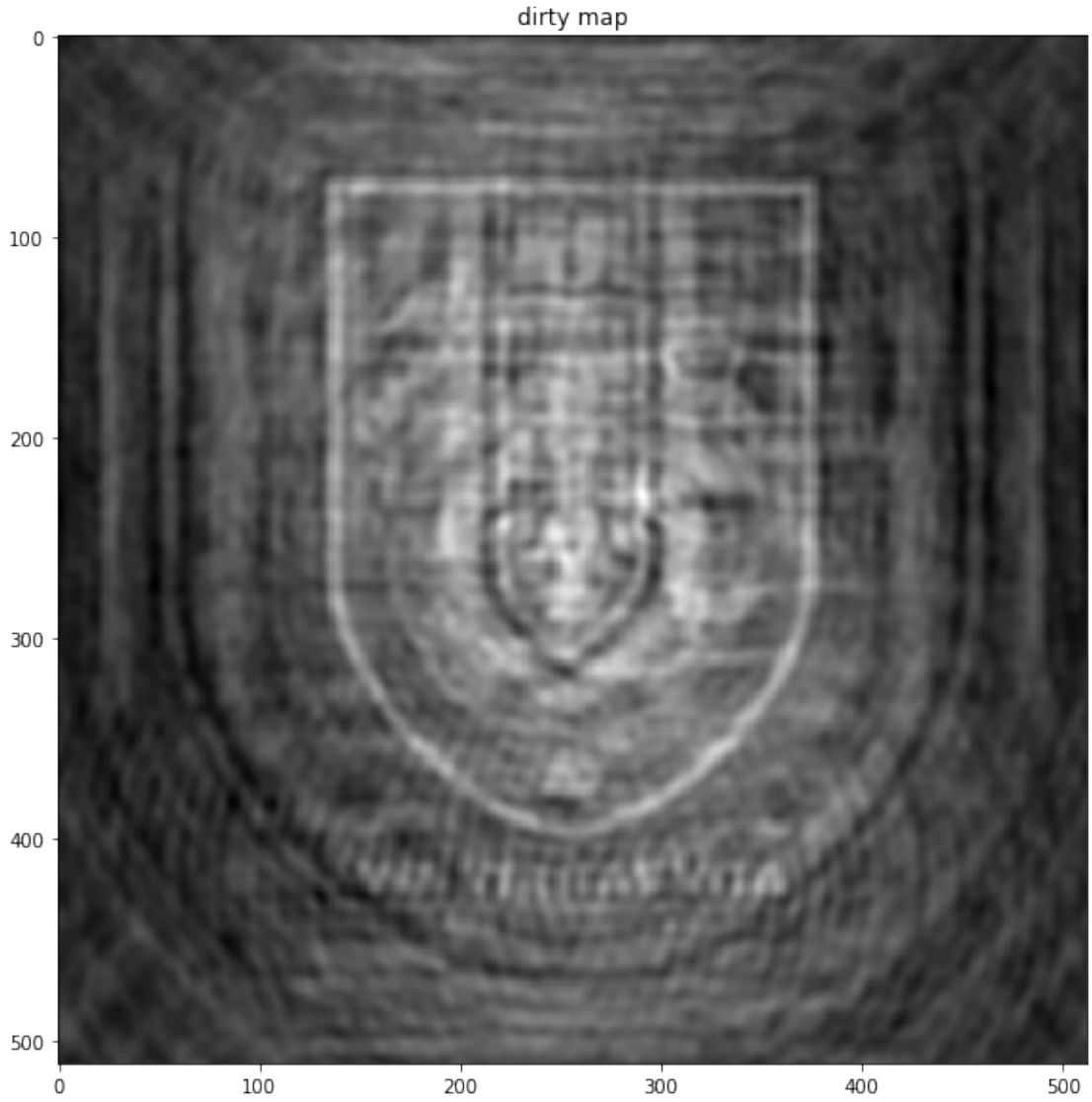
In [13]: synth_beam = np.fft.fftshift(np.fft.irfft2(aperture))

```
plt.imshow(synth_beam, cmap = 'gray')
plt.title('dirty beam');
```



This is quite a messy synthesised beam, with lots of sidelobe structure. It's usually referred to as the "dirty beam" (because it can be cleaned-up later with image processing techniques) and the sky map (image) it generates is the "dirty map":

```
In [14]: recon = np.fft.irfft2(F*aperture)
         plt.imshow((recon),cmap = 'gray')
         plt.title('dirty map');
```


dirty map
0
100
200
300
400
500
0
100
200
300
400
500