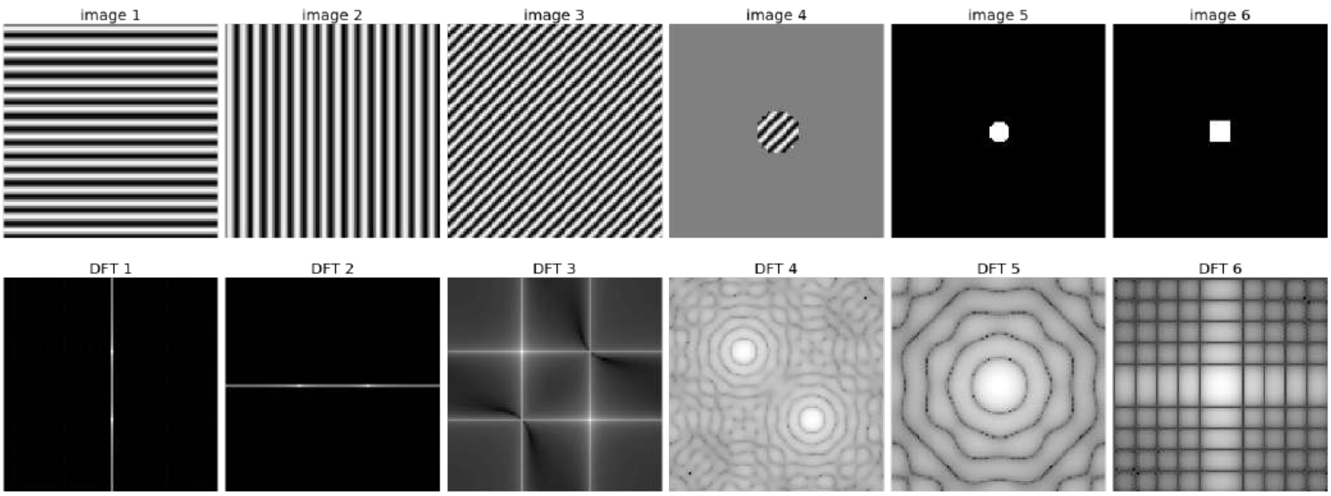
Chapter 3: Sampling, Convolution and Discrete Fourier Transform

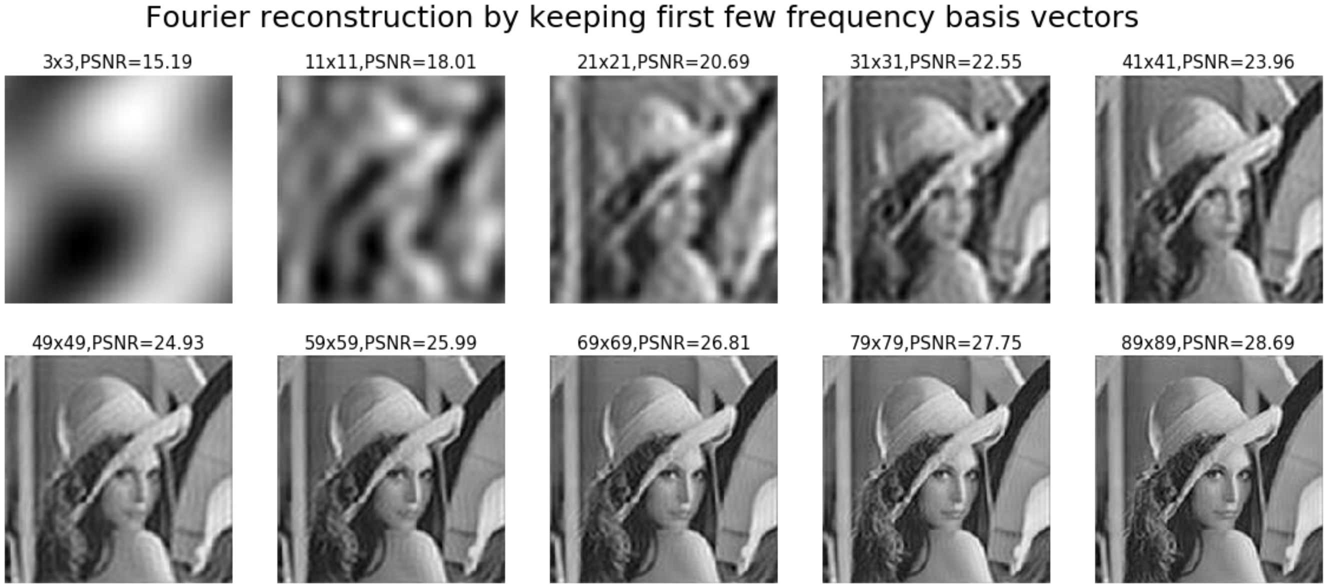
Problems

0. Fourier Transform Basics

In [14]:
```
plt.figure(figsize=(25,10))
i = 1
for im in images:
    plt.subplot(2,6,i), plot_image(im, 'image {}'.format(i))
    plt.subplot(2,6,i+6), plot_freq_spectrum(fp.fft2(im), 'DFT {}'.format(i), show_axis=
    i += 1
plt.tight_layout()
plt.show()
```



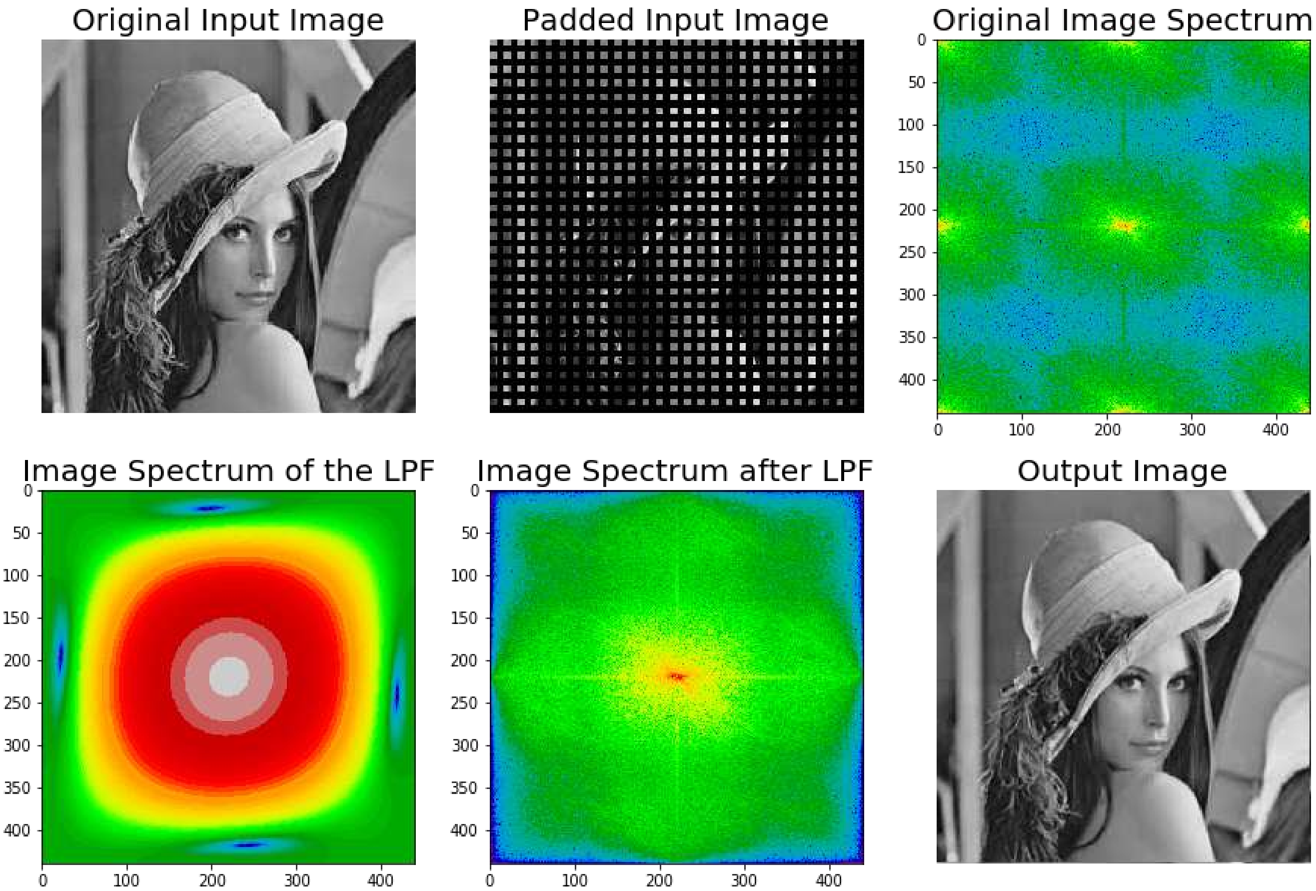
In [31]:
```
xs = list(map(int, np.linspace(1, h//5, 10)))
ys = list(map(int, np.linspace(1, w//5, 10)))
plt.figure(figsize=(20,8))
plt.gray()
for i in range(10):
    F_mask = np.zeros((h, w))
    F_mask[h//2-xs[i]:h//2+xs[i]+1, w//2-ys[i]:w//2+ys[i]+1] = 1
    F1 = F_shifted*F_mask
    im_out =  fp.ifft2(fp.ifftshift(F1)).real #np.abs()
    plt.subplot(2,5,i+1), plt.imshow(im_out), plt.axis('off')
    plt.title('{}x{},PSNR={}'.format(2*xs[i]+1, 2*ys[i]+1, round(peak_signal_noise_ratic
plt.suptitle('Fourier reconstruction by keeping first few frequency basis vectors', size
plt.show()
```



1. Sampling to increase/decrease the resolution of an image

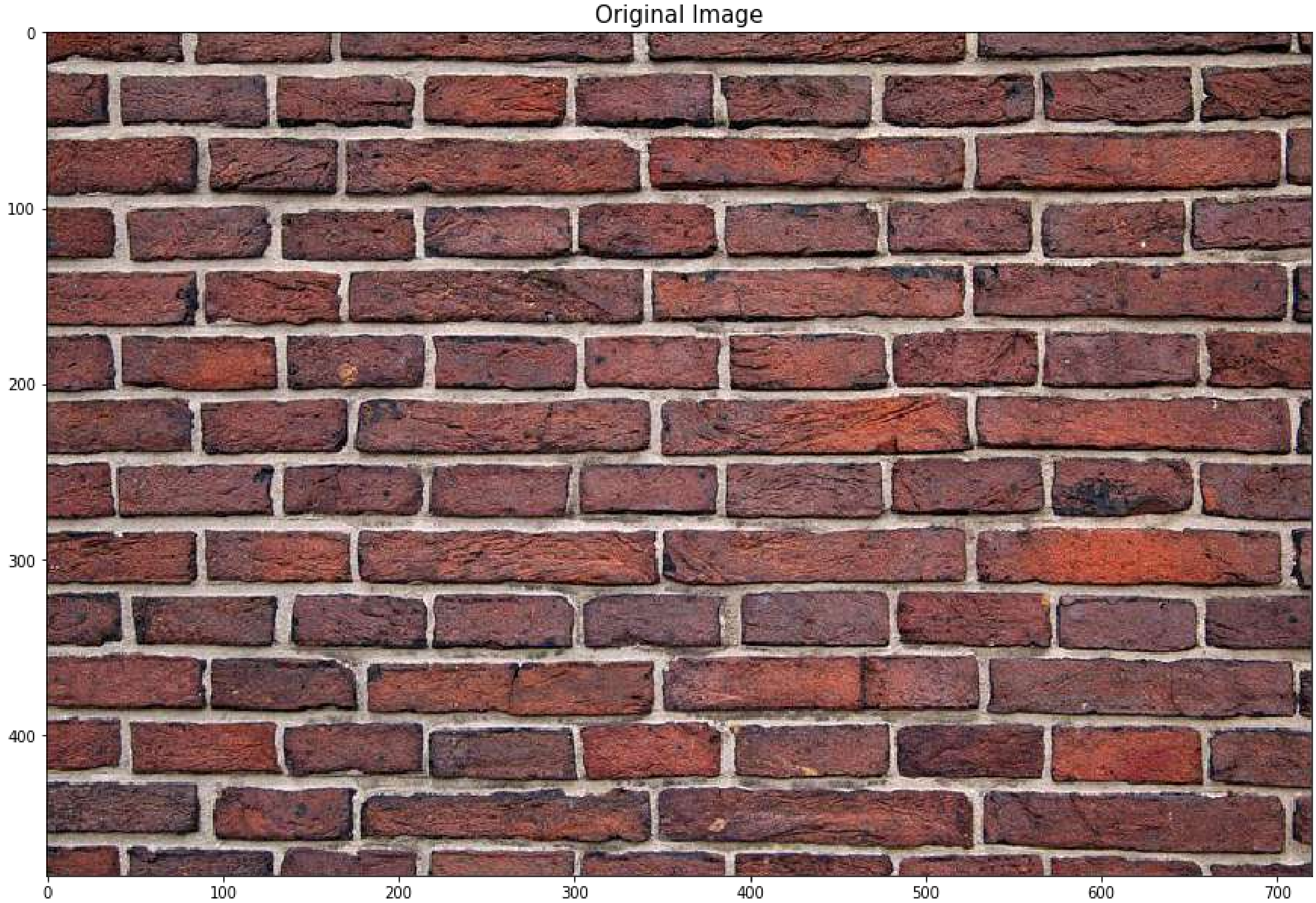
1.1. Up-sampling an image by using DFT and a Low-pass-filter (LPF)

In [12]:
```python
plt.figure(figsize=(15,10))
plt.gray() # show the filtered result in grayscale
cmap = 'nipy_spectral' #'viridis'
plt.subplot(231), plot_image(im, 'Original Input Image')
plt.subplot(232), plot_image(im1, 'Padded Input Image')
plt.subplot(233), plot_freq_spectrum(freq, 'Original Image Spectrum', cmap=cmap)
plt.subplot(234), plot_freq_spectrum(freq_kernel, 'Image Spectrum of the LPF', cmap=cmap
plt.subplot(235), plot_freq_spectrum(fp.fft2(im2), 'Image Spectrum after LPF', cmap=cmap
plt.subplot(236), plot_image(im2.astype(np.uint8), 'Output Image')
plt.show()
```

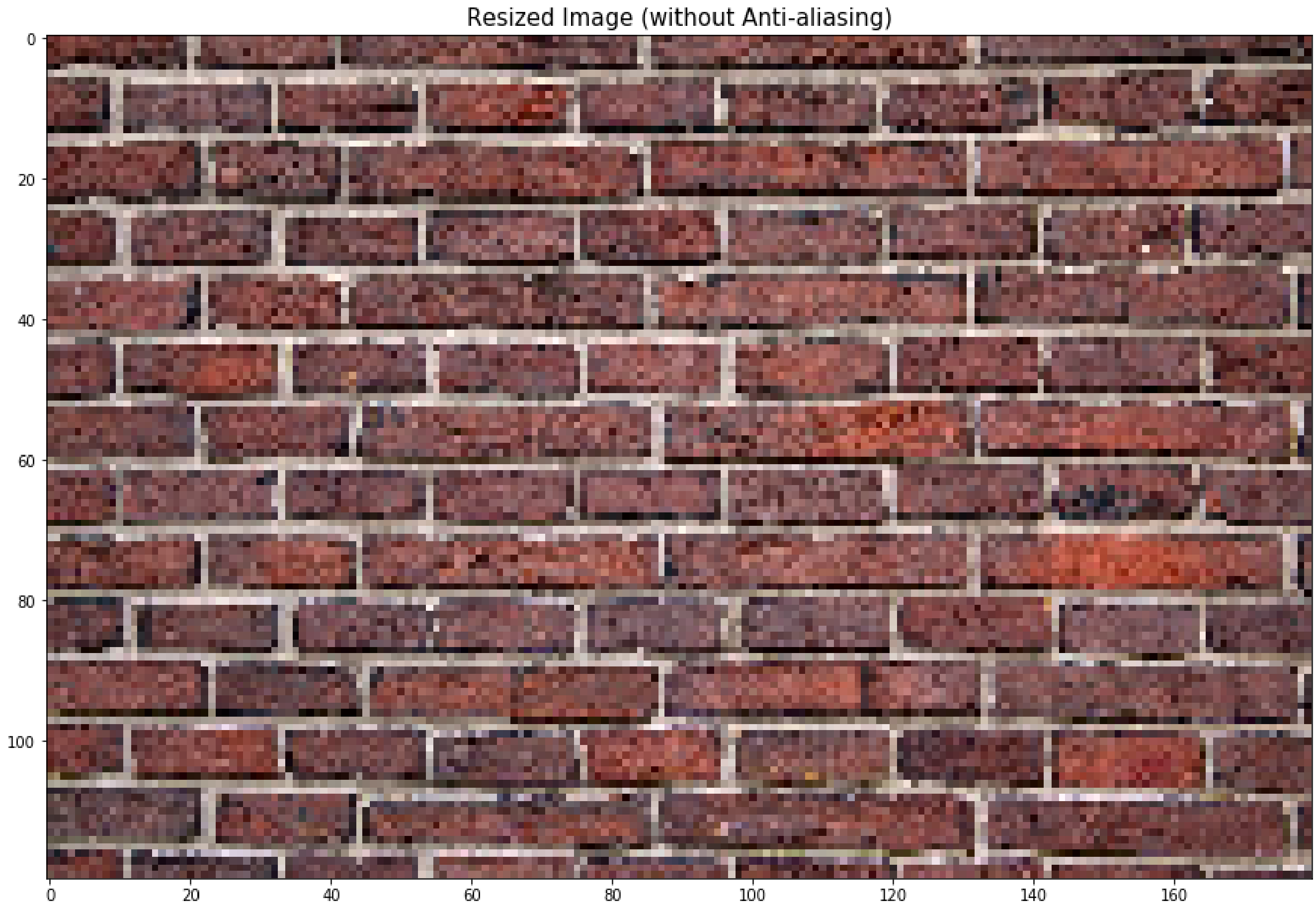


1.2. Down-sampling with Anti-Aliasing using Gaussian Filter

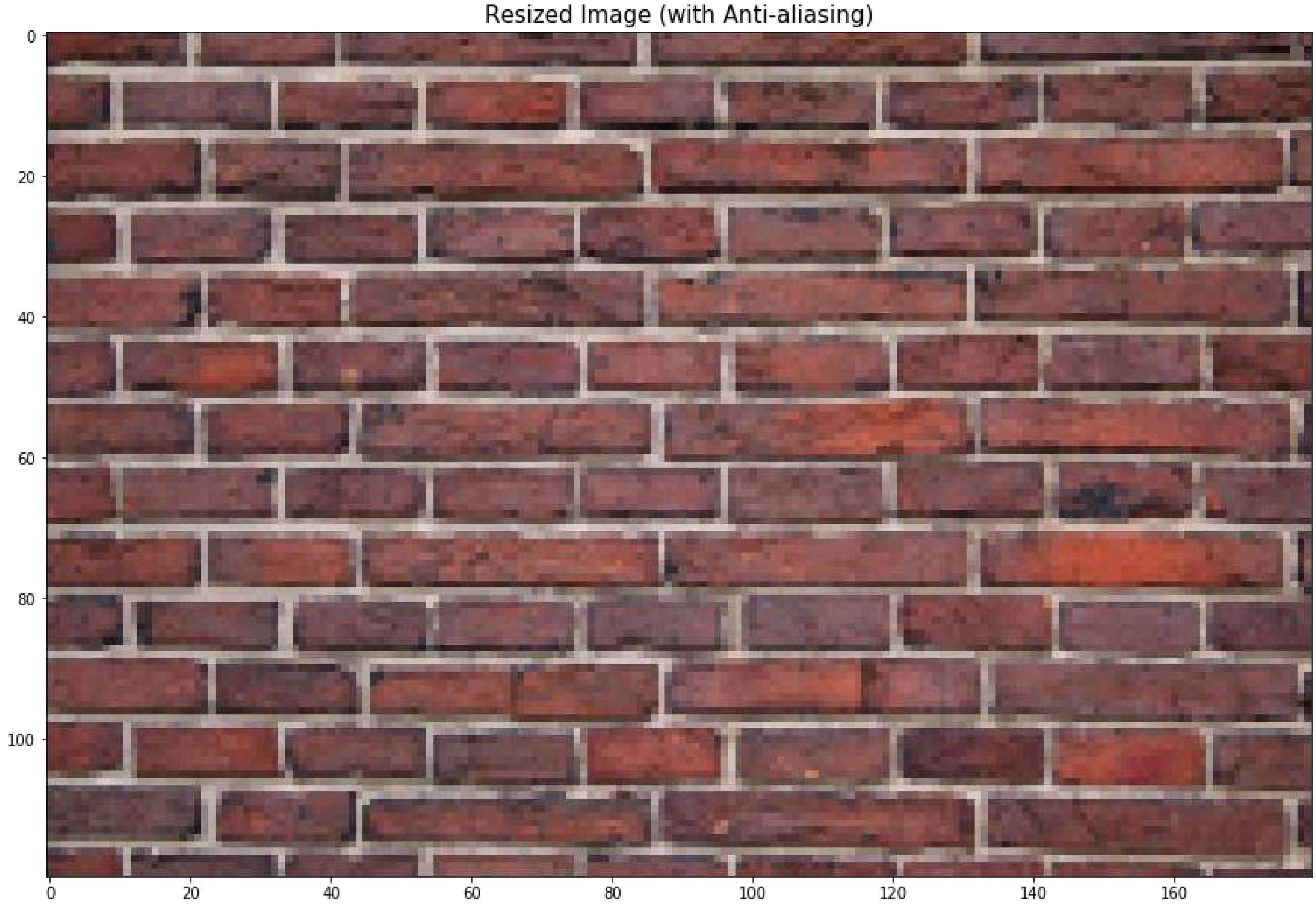
In [56]:
```python
plt.figure(figsize=(15,15))
plt.imshow(im), plt.title('Original Image', size=15)
plt.show()
```



In [57]:
```python
plt.figure(figsize=(15,15))
plt.imshow(im_small), plt.title('Resized Image (without Anti-aliasing)', size=15)
plt.show()
```



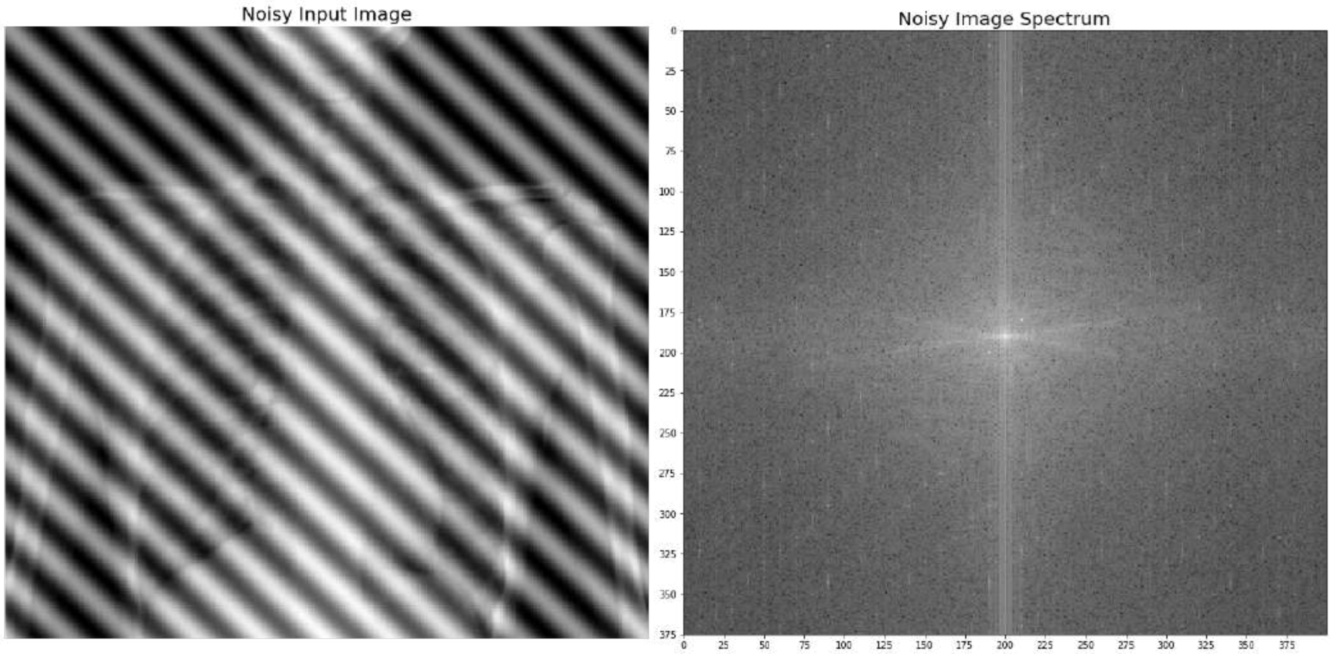
In [58]:
```
plt.figure(figsize=(15,15))
plt.imshow(im_small_aa), plt.title('Resized Image (with Anti-aliasing)', size=15)
plt.show()
```



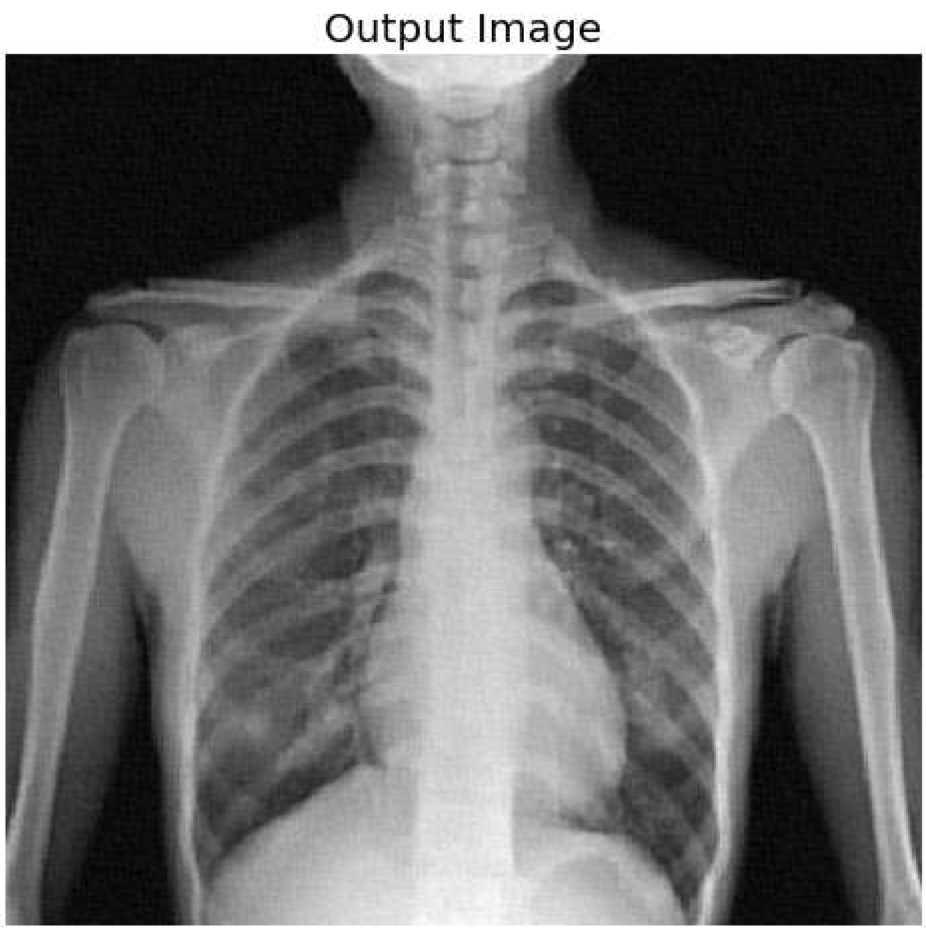
2. Denoising an Image with LPF/Notch filter in the Frequency domain

2.1 Removing Periodic Noise with Notch Filter

In [212]:
```
plt.figure(figsize=(20,10))

plt.subplot(121), plot_image(im_noisy, 'Noisy Input Image')
plt.subplot(122), plot_freq_spectrum(F_noisy, 'Noisy Image Spectrum')

plt.tight_layout()
plt.show()
```

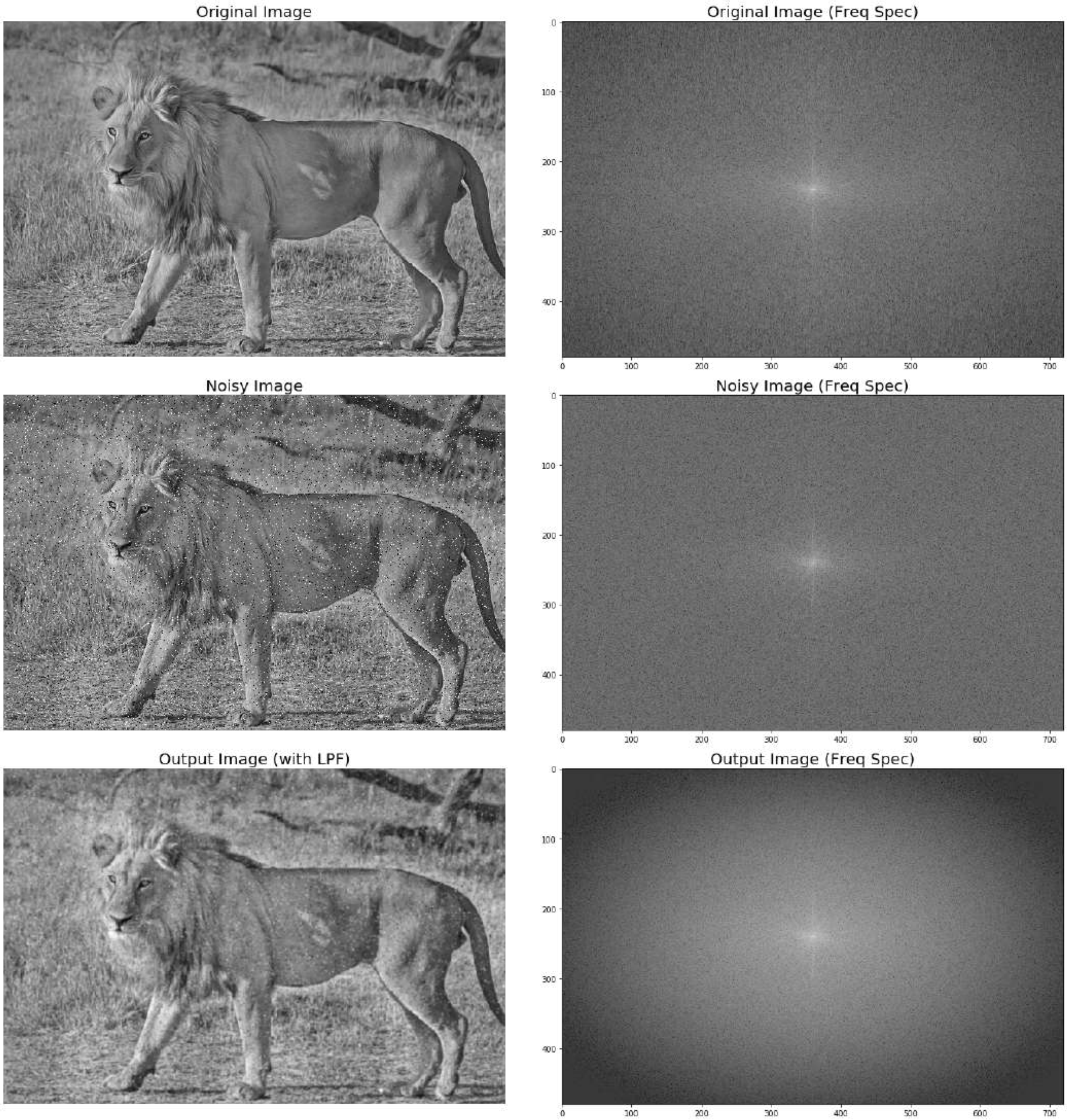


In [211]:
```
#print(signaltonoise(im1, axis=None))
plt.figure(figsize=(10,8))
plot_image(im_out, 'Output Image')
plt.show()
```



2.2 Removing salt-and-pepper noise using Gaussian LPF with scipy fftpack

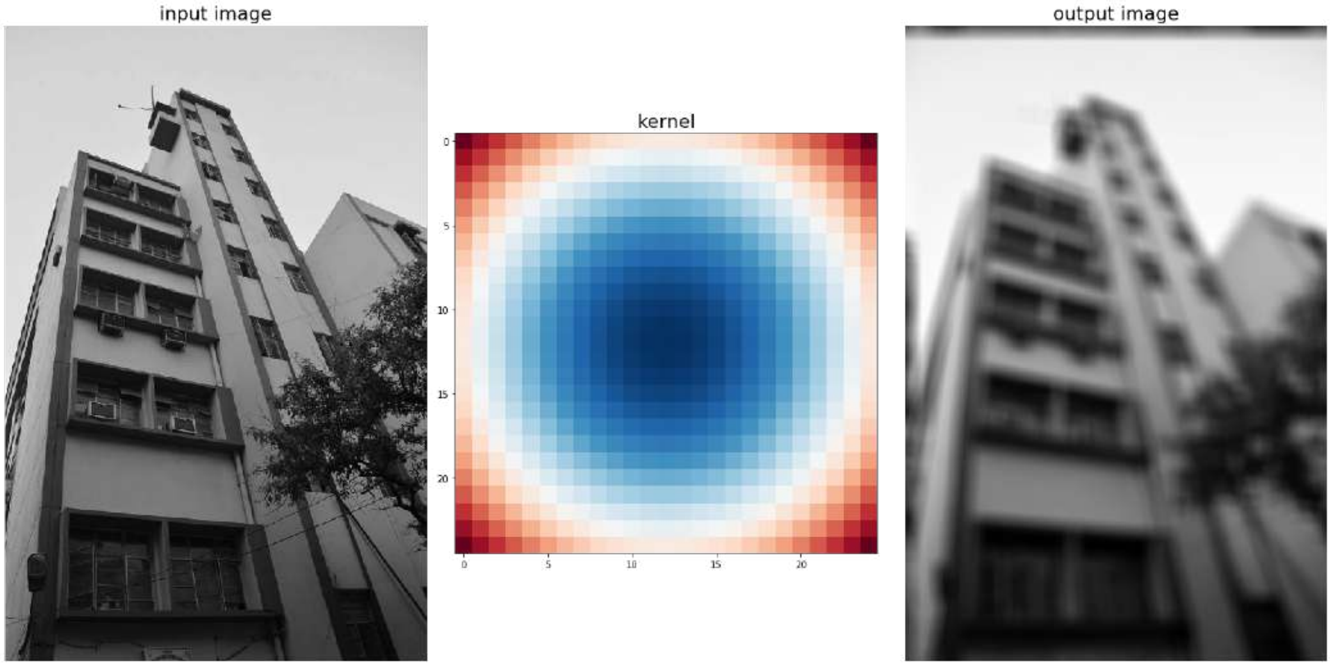
In [18]:
```
fig, ((ax1, ax2), (ax3, ax4), (ax5, ax6)) = plt.subplots(3, 2, figsize=(20,20))
plt.gray()  # show the filtered result in grayscale
ax1.imshow(im), ax1.axis('off'), ax1.set_title('Original Image', size=20)
ax2.imshow((20*np.log10(0.1 + fftpack.fftshift(im_freq))).real.astype(int))
ax2.set_title('Original Image (Freq Spec)', size=20)
ax3.imshow(noisy), ax3.axis('off'), ax3.set_title('Noisy Image', size=20)
ax4.imshow((20*np.log10( 0.1 + fftpack.fftshift(noisy_freq))).real.astype(int))
ax4.set_title('Noisy Image (Freq Spec)', size=20)
# the imaginary part is an artifact
ax5.imshow(noisy_smoothed.real), ax5.axis('off'), ax5.set_title('Output Image (with LPF
ax6.imshow( (20*np.log10( 0.1 + fftpack.fftshift(noisy_smoothed_freq))).real.astype(int
ax6.set_title('Output Image (Freq Spec)', size=20)
plt.tight_layout()
plt.show()
```



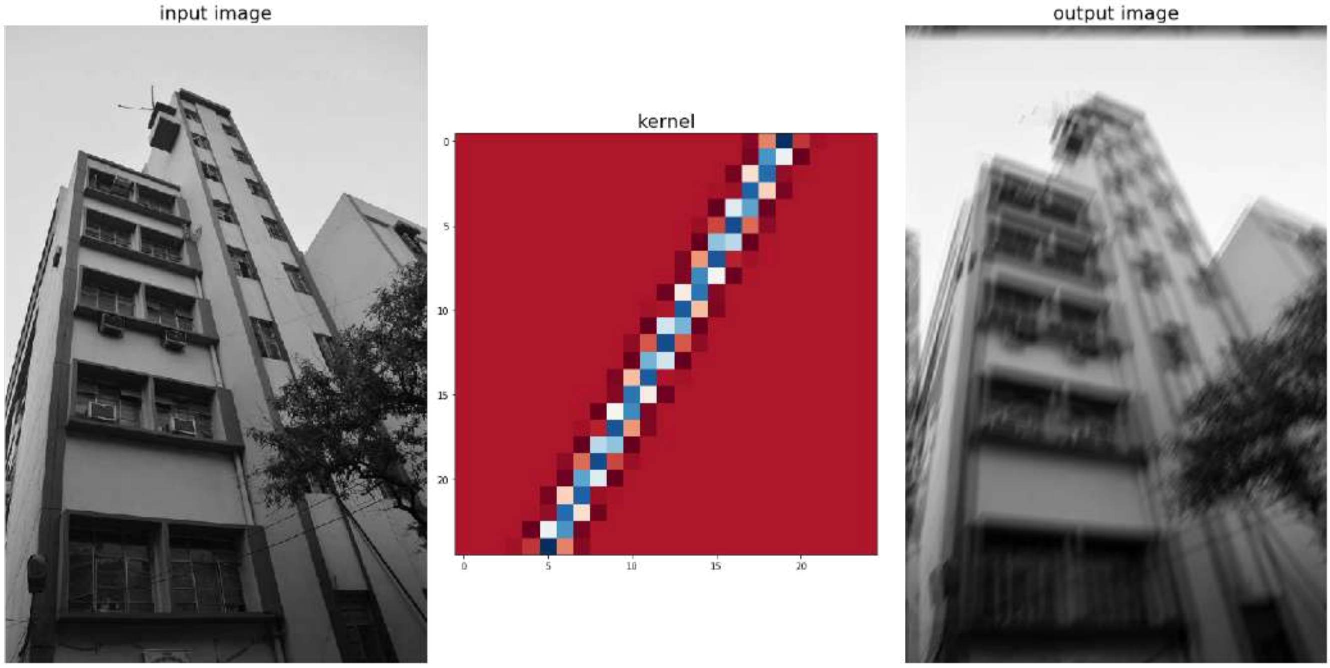
3. Blurring an Image with an LPF in the Frequency domain

3.0 Different Blur Kernels and Convolution in the Frequency domain

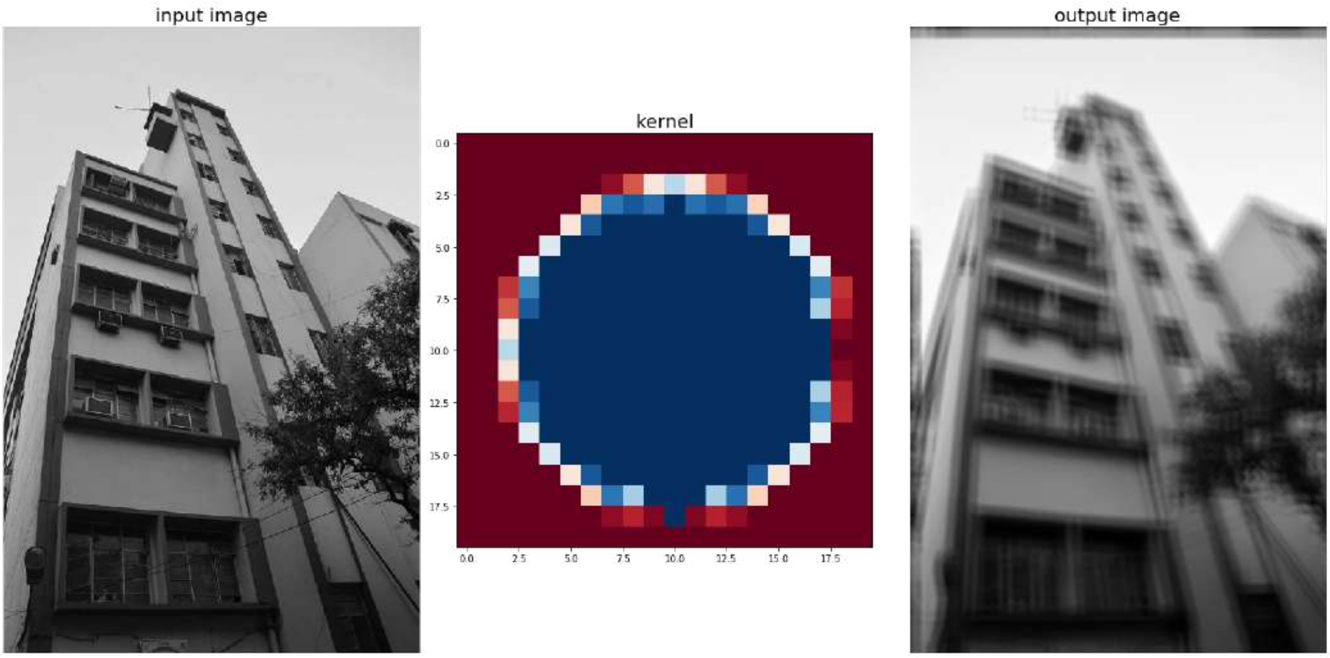
In [40]:
```python
im = rgb2gray(imread('images/Img_03_03.jpg'))

kernel = get_gaussian_edge_blur_kernel(25, 25)
dft_convolve(im, kernel)
```



In [41]:
```python
kernel = get_motion_blur_kernel(30, 60, 25)
dft_convolve(im, kernel)
```



In [42]:
```
kernel = get_out_of_focus_kernel(15, 20)
dft_convolve(im, kernel)
```

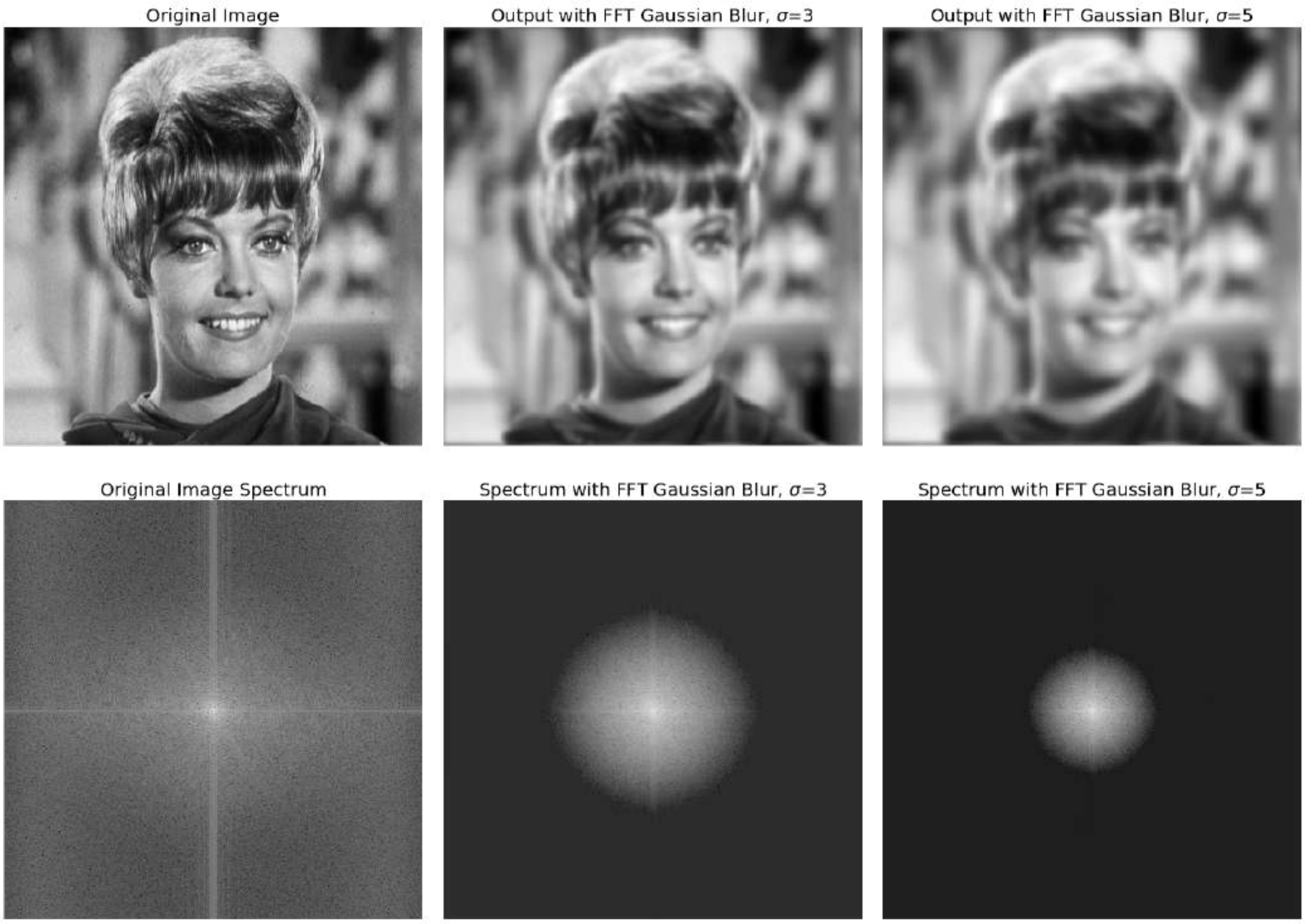


3.1 Blurring with scipy.ndimage frequency-domain filters

With fourier_gaussian

In [39]:

```
fig, axes = plt.subplots(2, 3, figsize=(20,15))
plt.subplots_adjust(0,0,1,0.95,0.05,0.05)
plt.gray() # show the filtered result in grayscale
axes[0, 0].imshow(im), axes[0, 0].set_title('Original Image', size=20)
axes[1, 0].imshow((20*np.log10( 0.1 + fp.fftshift(freq))).real.astype(int)), axes[1, 0]
i = 1
for sigma in [3,5]:
    convolved_freq = ndimage.fourier_gaussian(freq, sigma=sigma)
    convolved = fp.ifft2(convolved_freq).real # the imaginary part is an artifact
    axes[0, i].imshow(convolved)
    axes[0, i].set_title(r'Output with FFT Gaussian Blur, $\sigma$={}'.format(sigma), s
    axes[1, i].imshow((20*np.log10( 0.1 + fp.fftshift(convolved_freq))).real.astype(int
    axes[1, i].set_title(r'Spectrum with FFT Gaussian Blur, $\sigma$={}'.format(sigma),
    i += 1
for a in axes.ravel():
    a.axis('off')
plt.show()
```

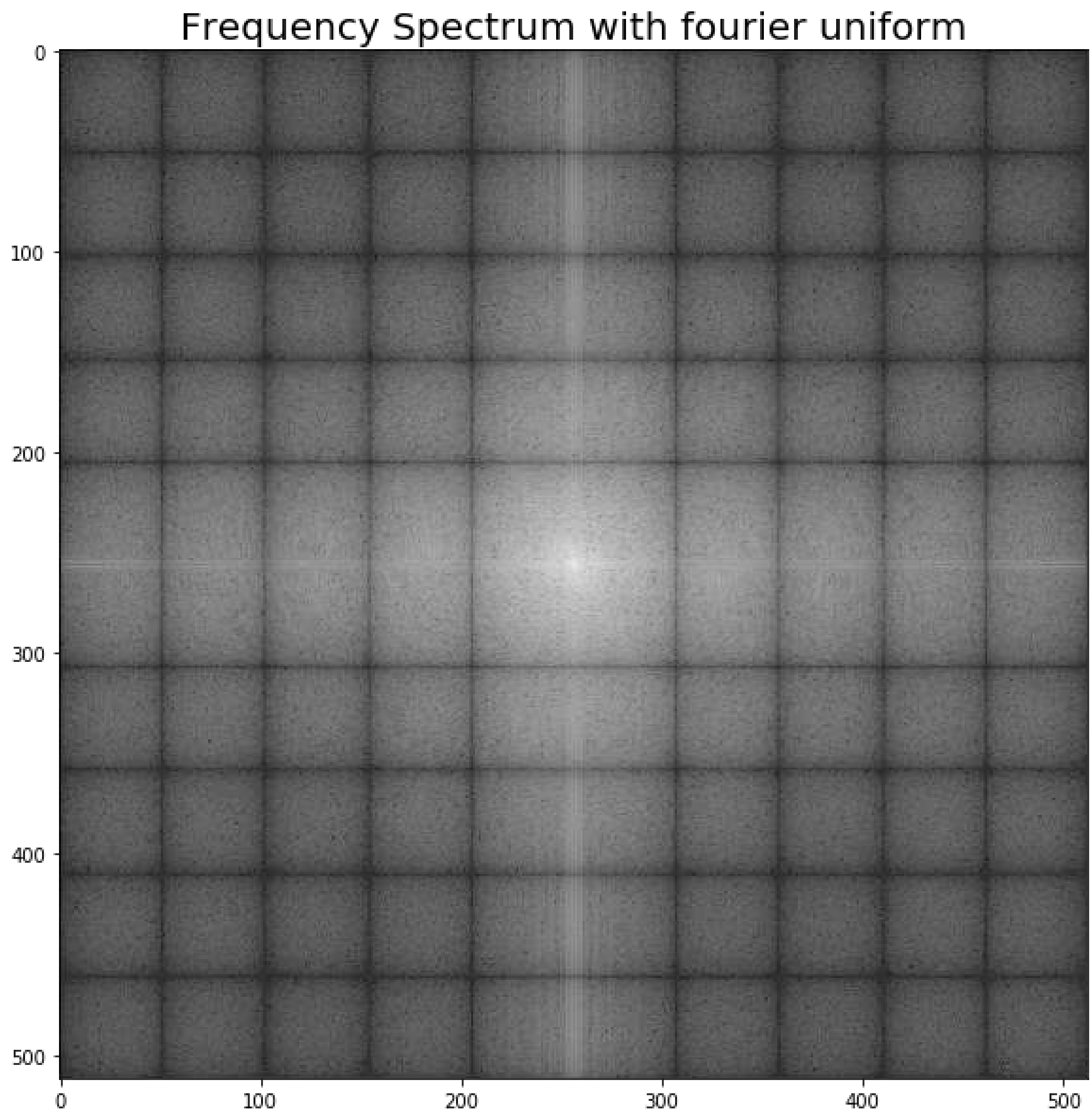


With fourier_uniform

In [30]:

```
fig, (axes1, axes2) = plt.subplots(1, 2, figsize=(20,10))
plt.gray() # show the result in grayscale
im1 = fp.ifft2(freq_uniform)
axes1.imshow(im), axes1.axis('off')
axes1.set_title('Original Image', size=20)
axes2.imshow(im1.real) # the imaginary part is an artifact
axes2.axis('off')
axes2.set_title('Blurred Image with Fourier Uniform', size=20)
plt.tight_layout()
plt.show()
```

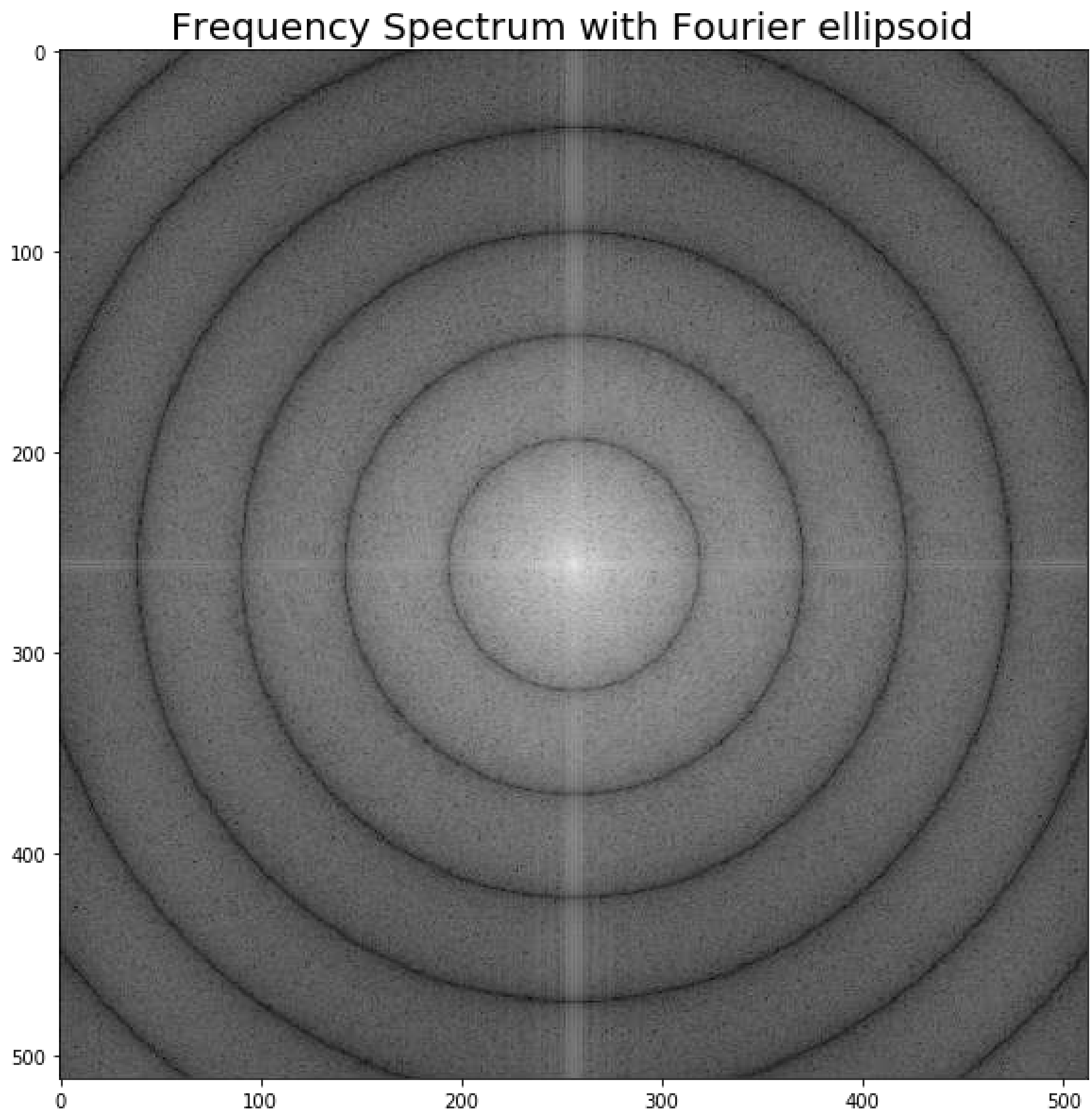
In [29]:
```
plt.figure(figsize=(10,10))
plt.imshow( (20*np.log10( 0.1 + fp.fftshift(freq_uniform))).real.astype(int))
plt.title('Frequency Spectrum with fourier uniform', size=20)
plt.show()
```



With fourier_ellipsoid

In [31]:
```
fig, (axes1, axes2) = plt.subplots(1, 2, figsize=(20,10))
axes1.imshow(im), axes1.axis('off')
axes1.set_title('Original Image', size=20)
axes2.imshow(im1.real) # the imaginary part is an artifact
axes2.axis('off')
axes2.set_title('Blurred Image with Fourier Ellipsoid', size=20)
plt.tight_layout()
plt.show()
```

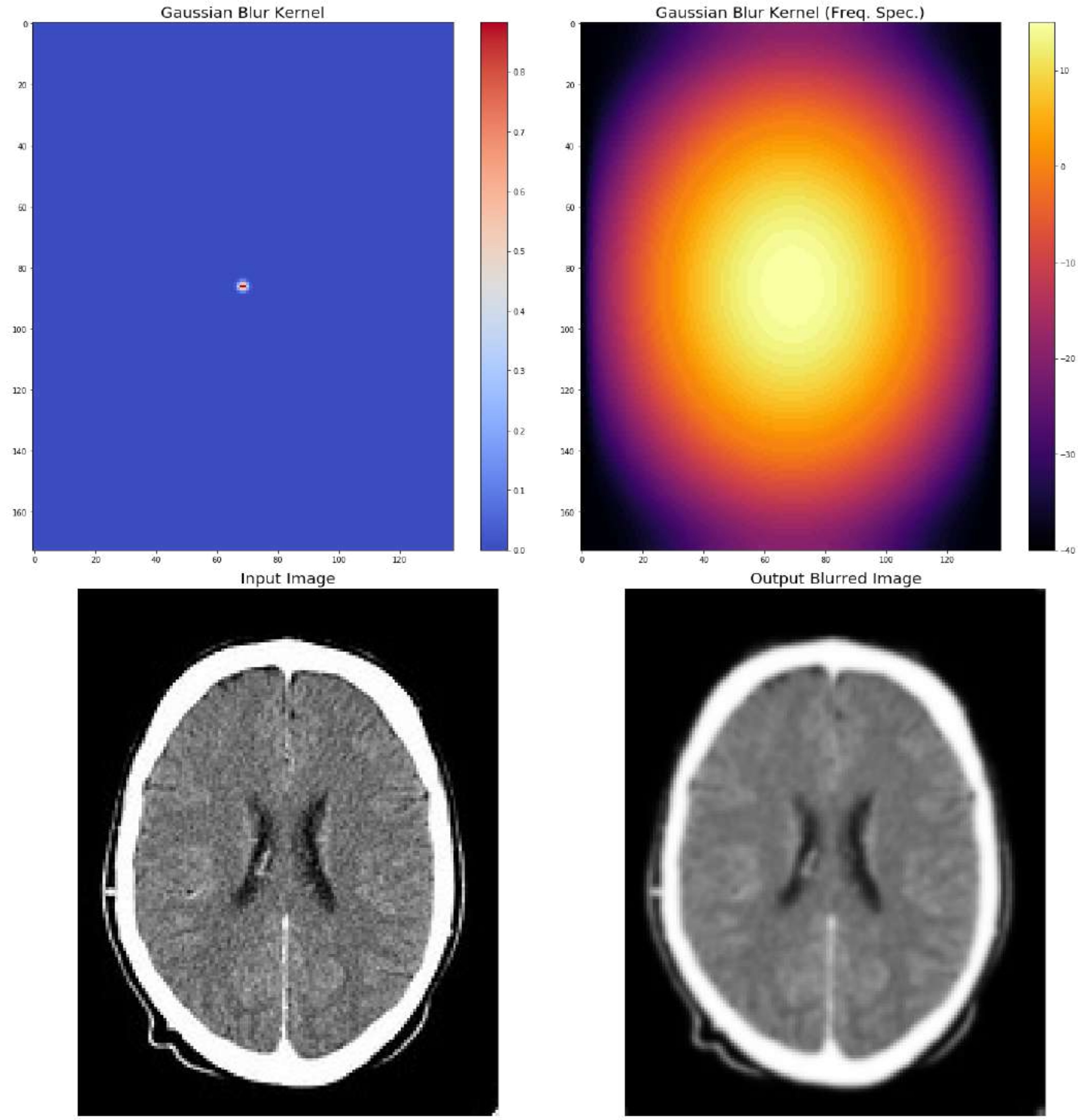
In [28]:
```
plt.figure(figsize=(10,10))
plt.imshow( (20*np.log10( 0.1 + fp.fftshift(freq_ellipsoid))).real.astype(int))
plt.title('Frequency Spectrum with Fourier ellipsoid', size=20)
plt.show()
```



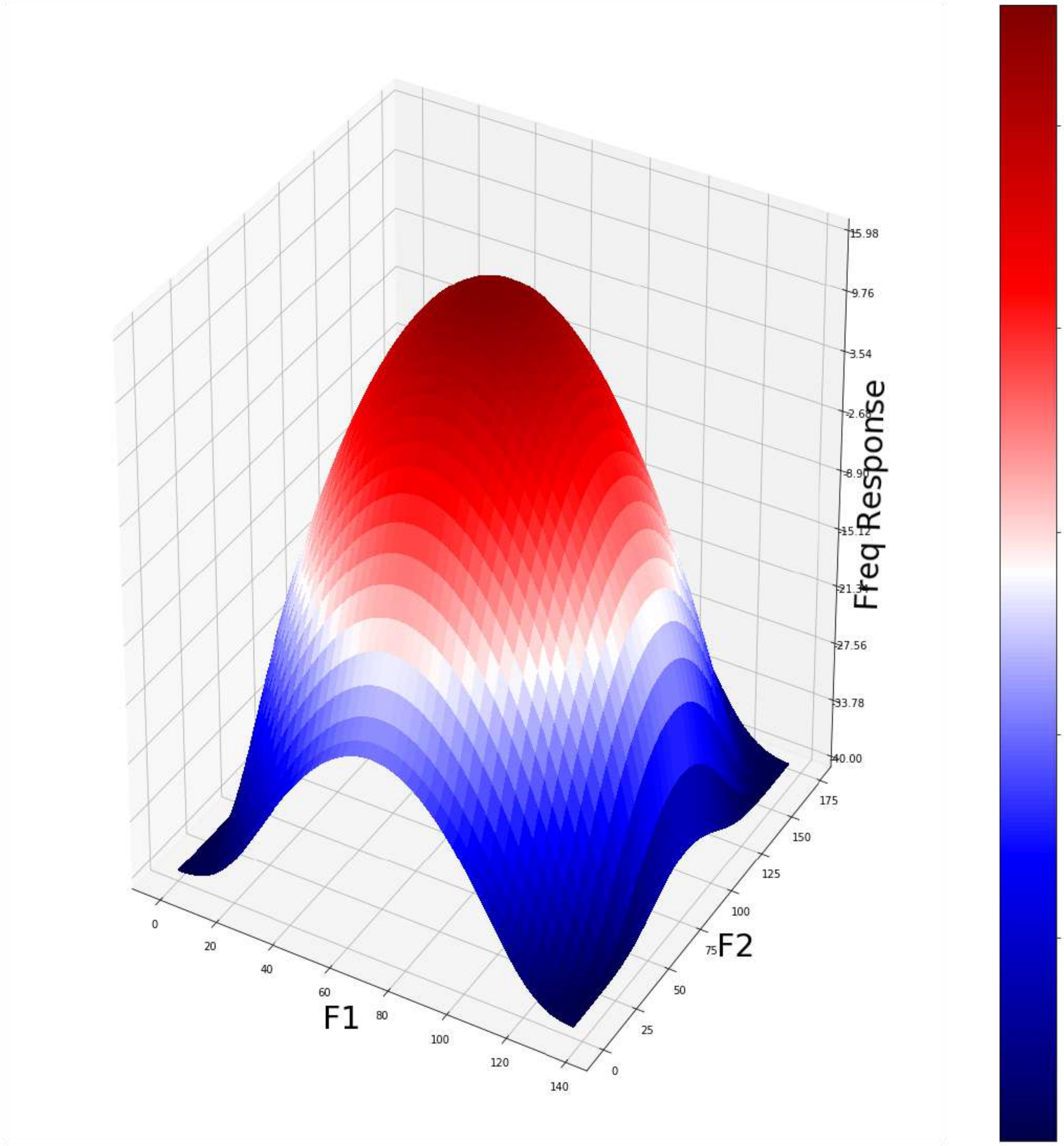
3.2 Gaussian Blur LowPass Filter with scipy.fftpack

In [87]:

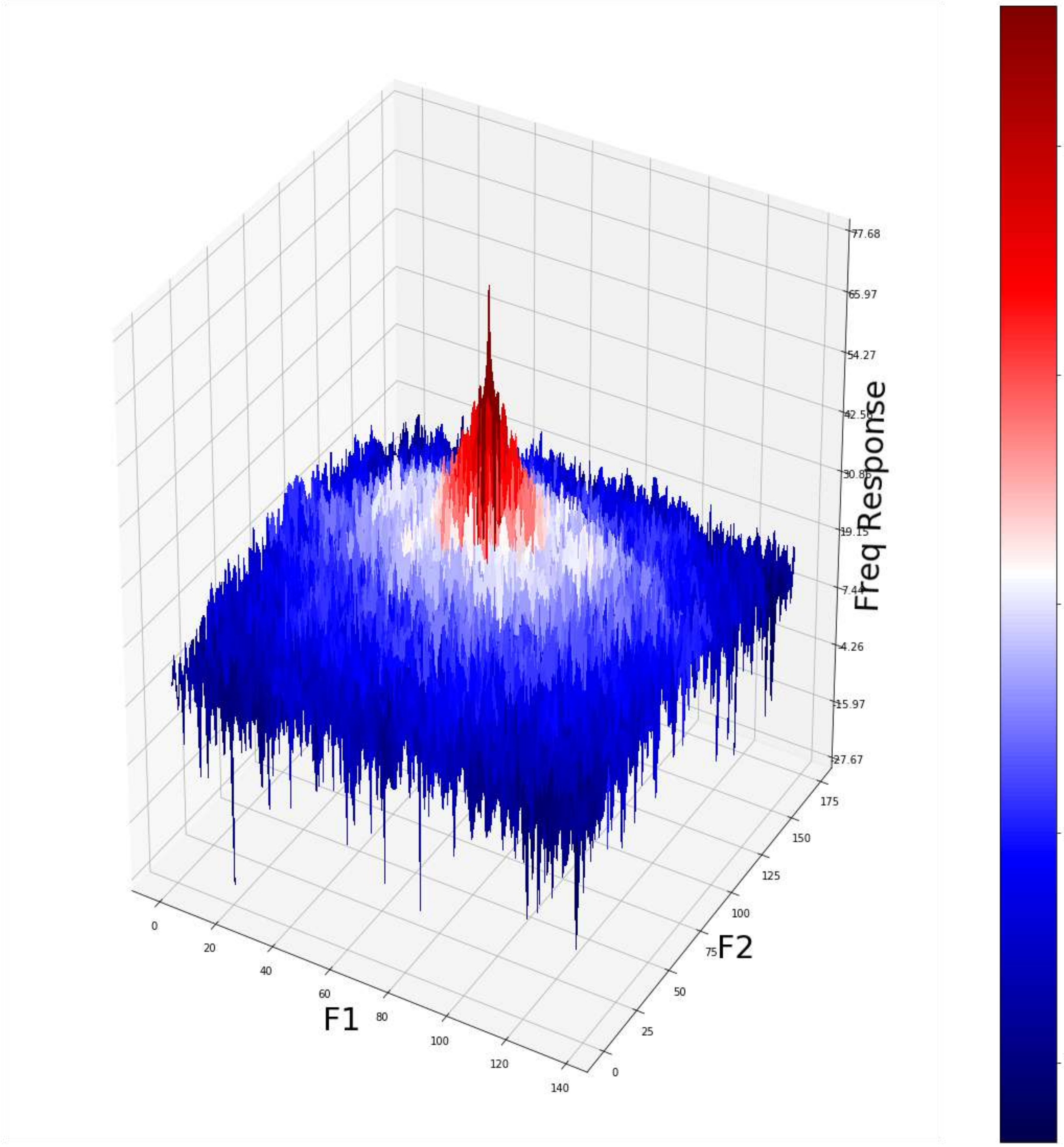
```
plt.figure(figsize=(20,20))
plt.subplot(221), plt.imshow(kernel, cmap='coolwarm'), plt.colorbar()
plt.title('Gaussian Blur Kernel', size=20)
# center the frequency response
plt.subplot(222)
plt.imshow( (20*np.log10( 0.01 + fp.fftshift(freq_kernel))).real.astype(int), cmap='infe
plt.colorbar()
plt.title('Gaussian Blur Kernel (Freq. Spec.)', size=20)
plt.subplot(223), plt.imshow(im, cmap='gray'), plt.axis('off'), plt.title('Input Image',
plt.subplot(224), plt.imshow(im_blur, cmap='gray'), plt.axis('off'), plt.title('Output B
plt.tight_layout()
plt.show()
```



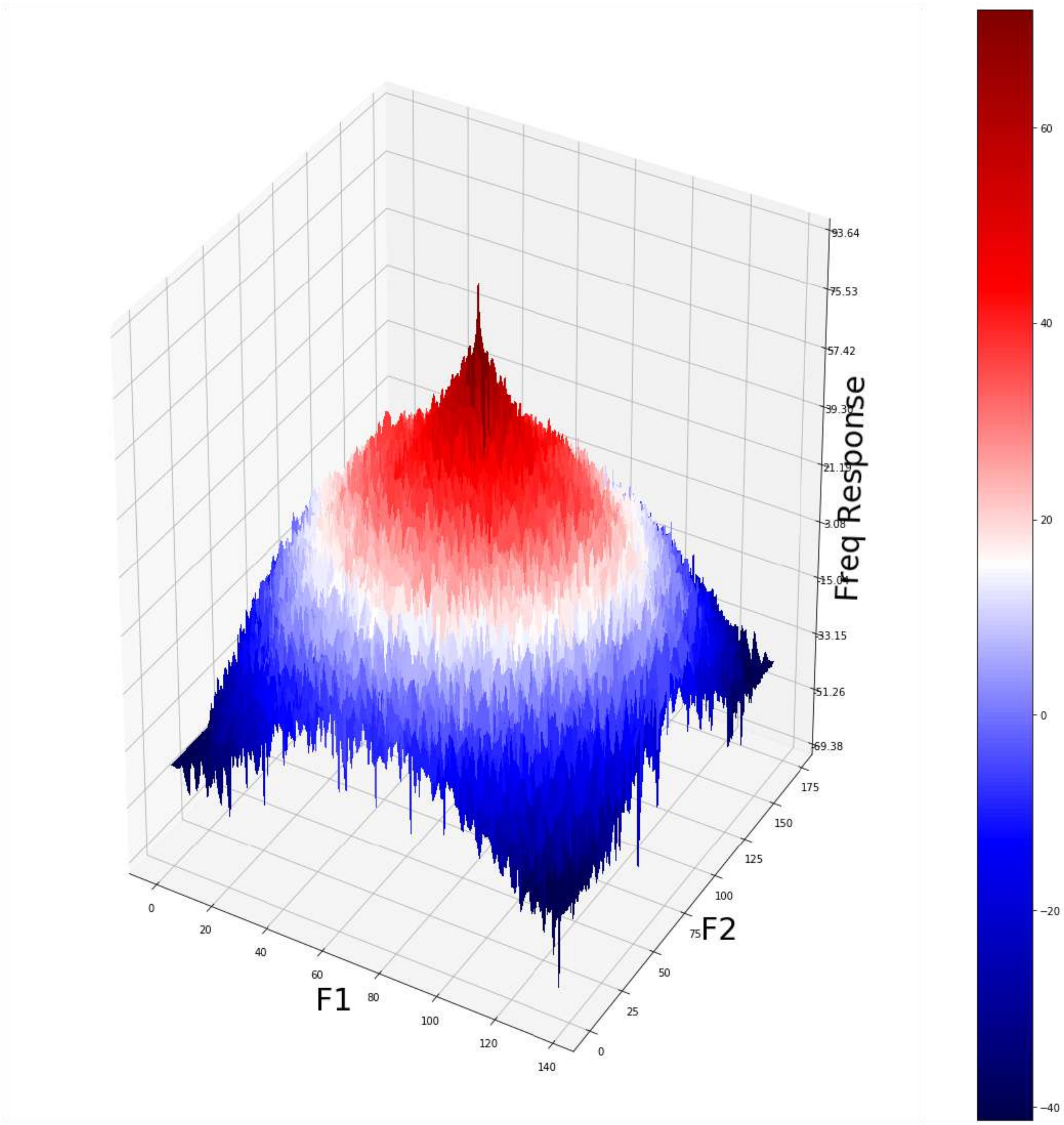
In [73]:
```python
Y = np.arange(freq.shape[0]) #-freq.shape[0]//2,freq.shape[0]-freq.shape[0]//2)
X = np.arange(freq.shape[1]) #-freq.shape[1]//2,freq.shape[1]-freq.shape[1]//2)
X, Y = np.meshgrid(X, Y)
Z = (20*np.log10( 0.01 + fp.fftshift(freq_kernel))).real
plot_3d(X,Y,Z)
```



In [74]:
```
Z = (20*np.log10( 0.01 + fp.fftshift(freq))).real
plot_3d(X,Y,Z)
```



In [75]:
```python
Z = (20*np.log10( 0.01 + fp.fftshift(convolved))).real
plot_3d(X,Y,Z)
```



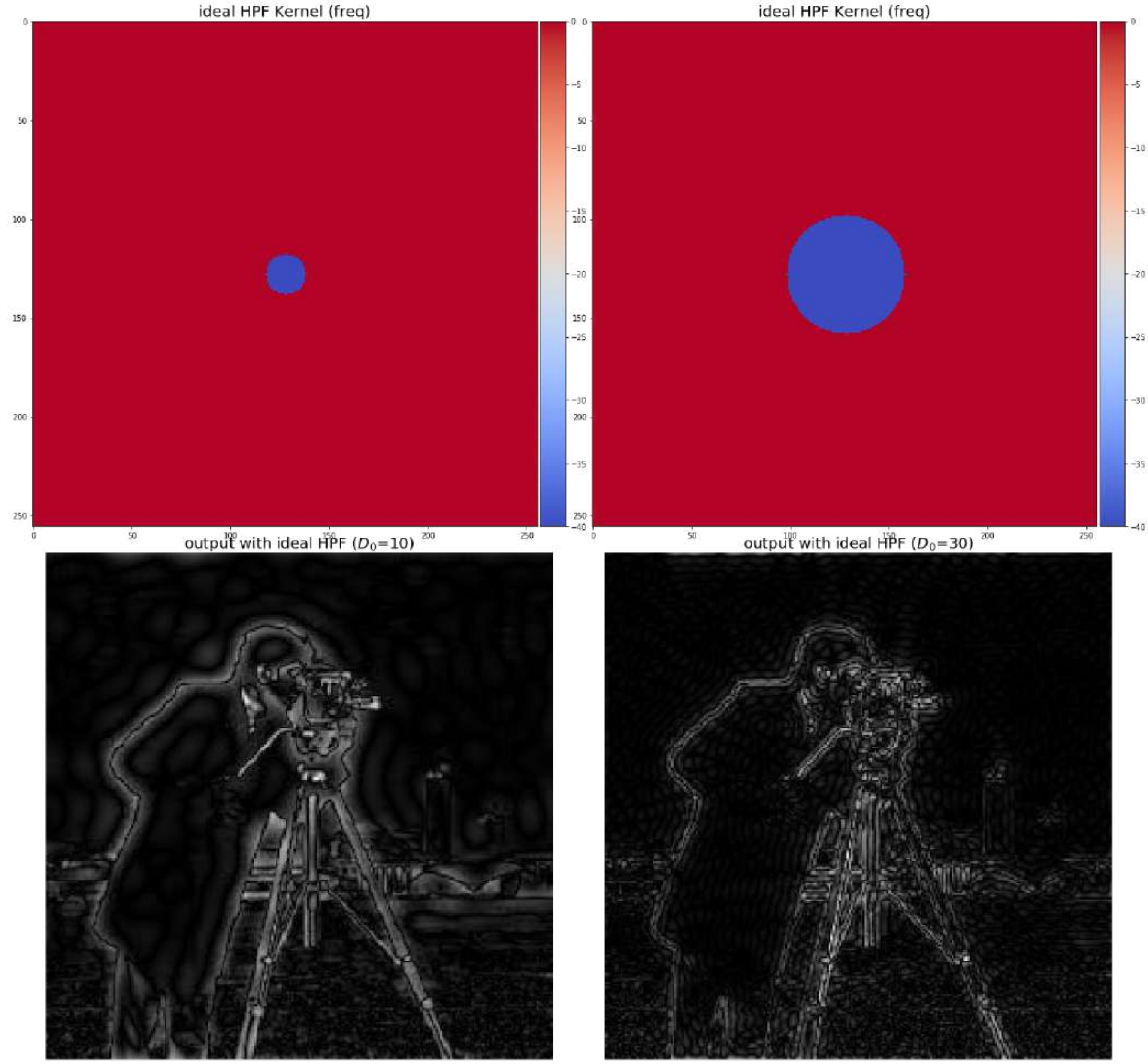
3.3 Convolution in frequency domain with a colored image using fftconvolve from scipy signal

```
In [94]: plt.figure(figsize=(20,10))
plt.subplot(131), plt.imshow(im), plt.axis('off'), plt.title('original image', size=20)
plt.subplot(132), plt.imshow(im1), plt.axis('off'), plt.title('output with Gaussian LPF
plt.subplot(133), plt.imshow(im2), plt.axis('off'), plt.title('output with Laplacian HPF
plt.tight_layout()
plt.show()
```

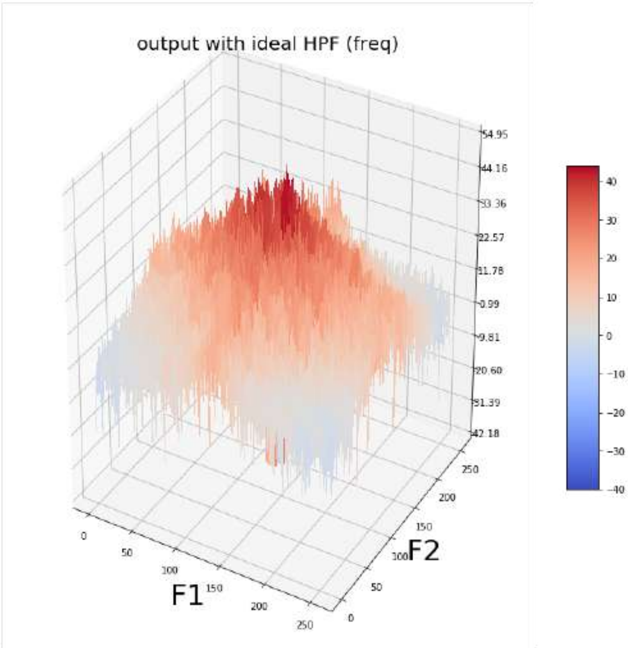
4. Edge Detection with High-Pass Filters (HPF) in the Frequency domain

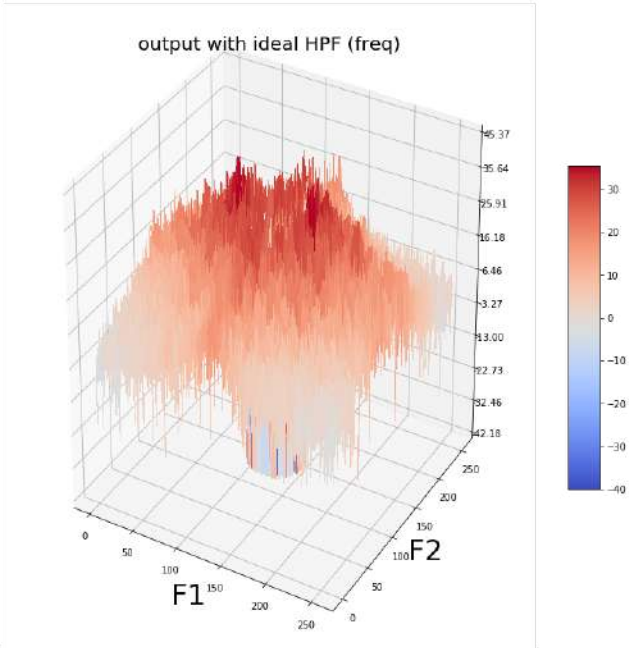
```
In [65]: im = plt.imread('images/Img_03_12.png')
im = rgb2gray(im)
plt.figure(figsize=(7,12))
plt.imshow(im), plt.axis('off'), plt.title('original image')
plt.show()
```

In [67]:
```
plot_HPF(im, ideal, D0)
```

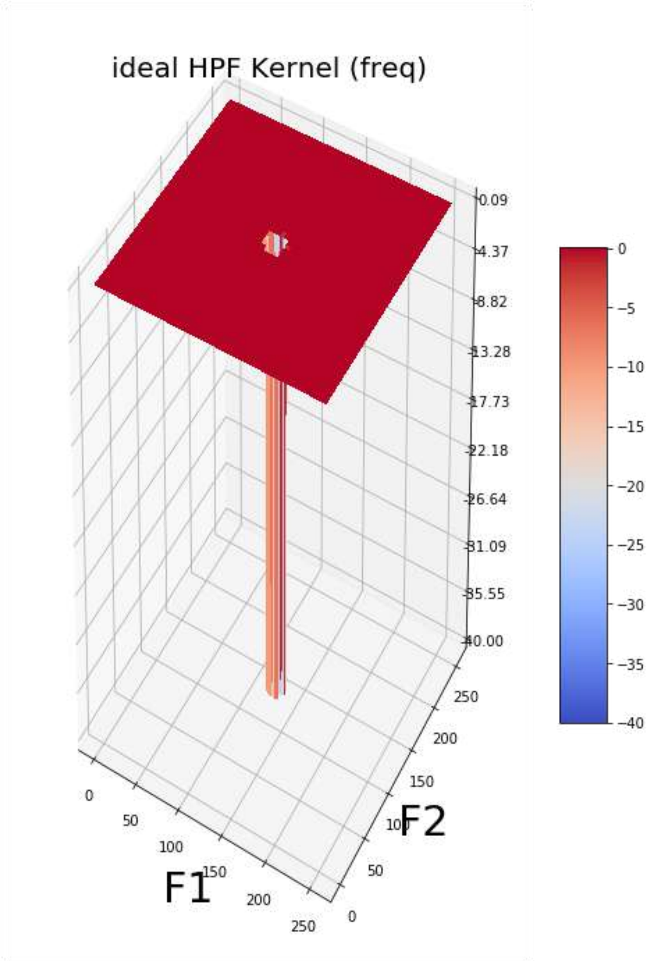


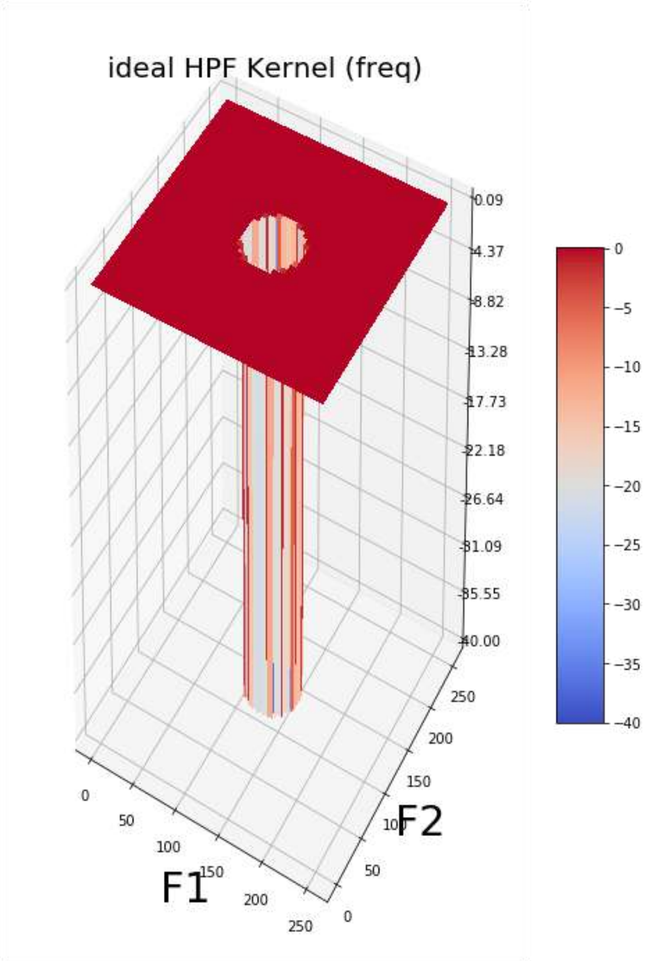
In [252]:
```
plot_HPF_3d(im, ideal, D0)
```





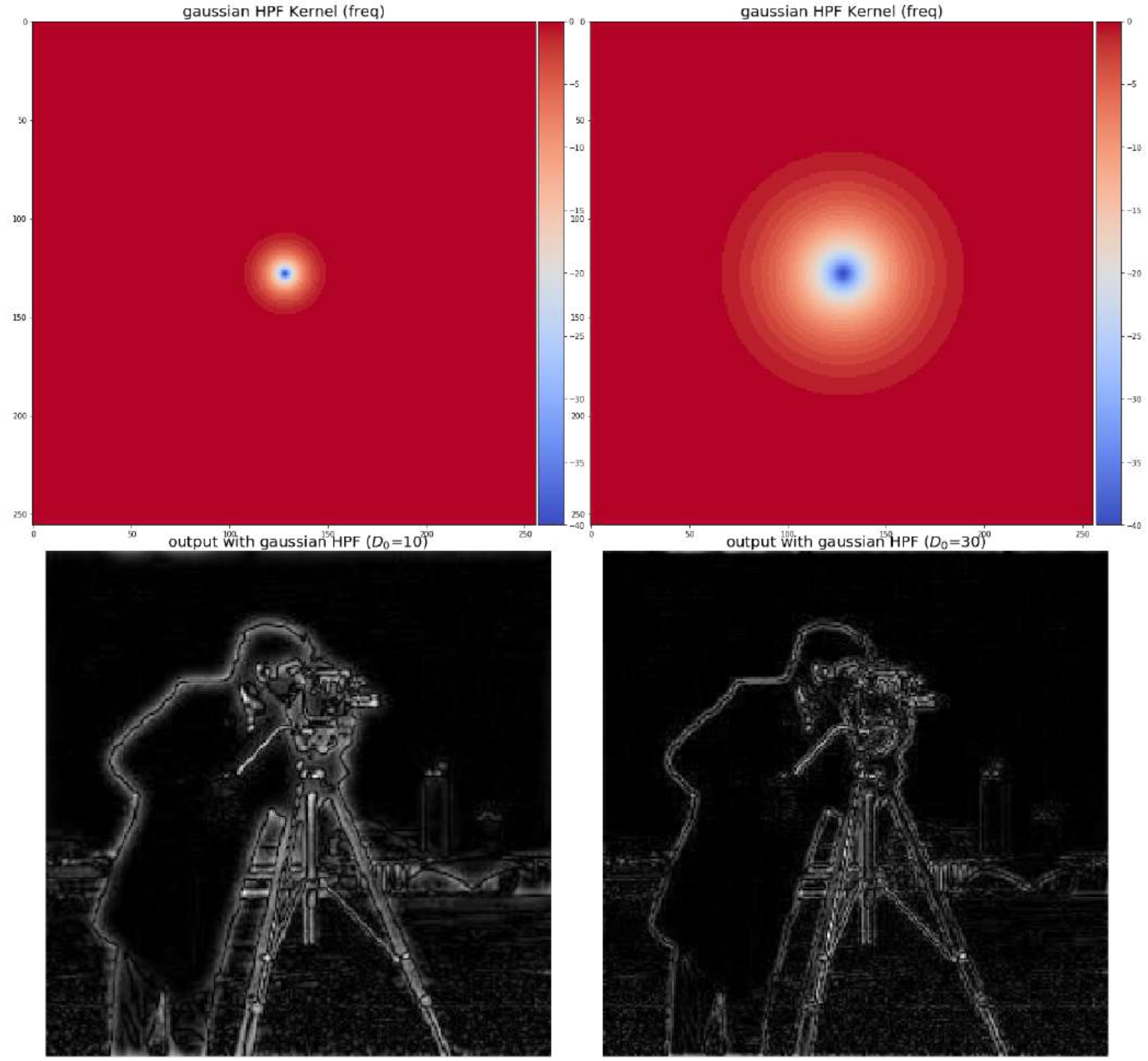
In [253]:
```
plot_filter_3d(im.shape, ideal, D0)
```



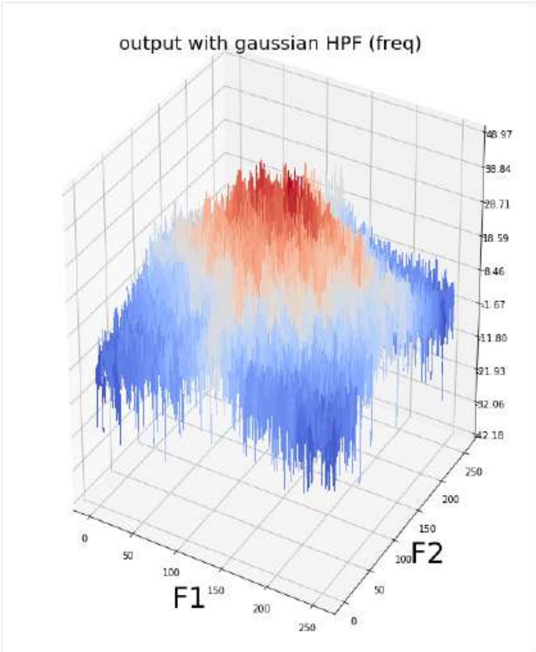


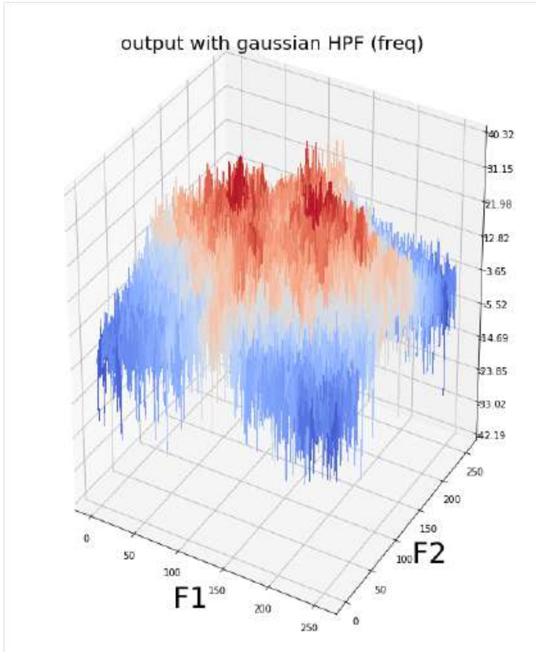
In [254]:

```
plot_HPF(im, gaussian, D0)
```

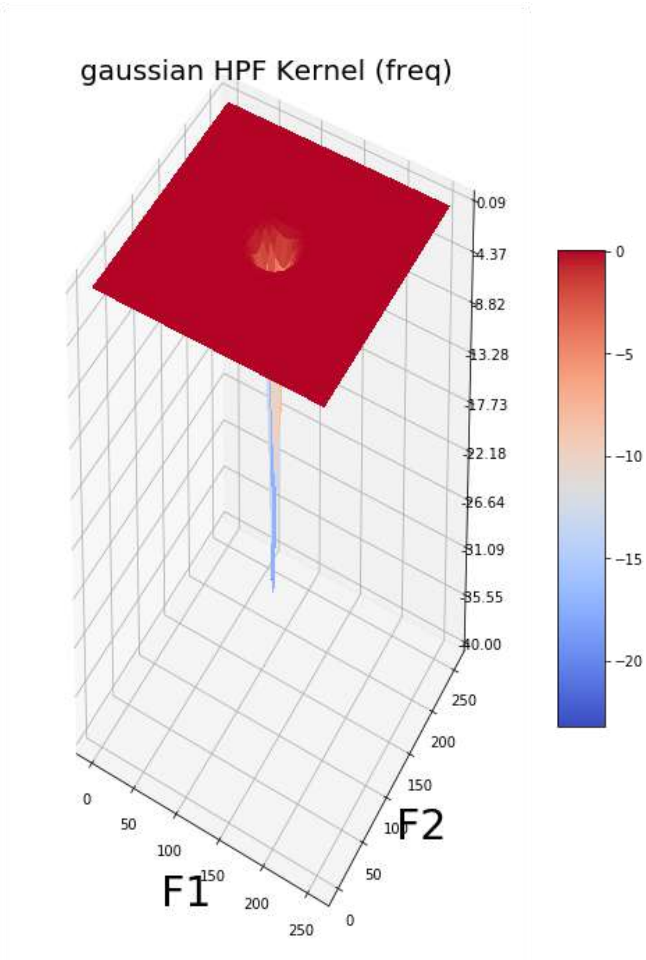


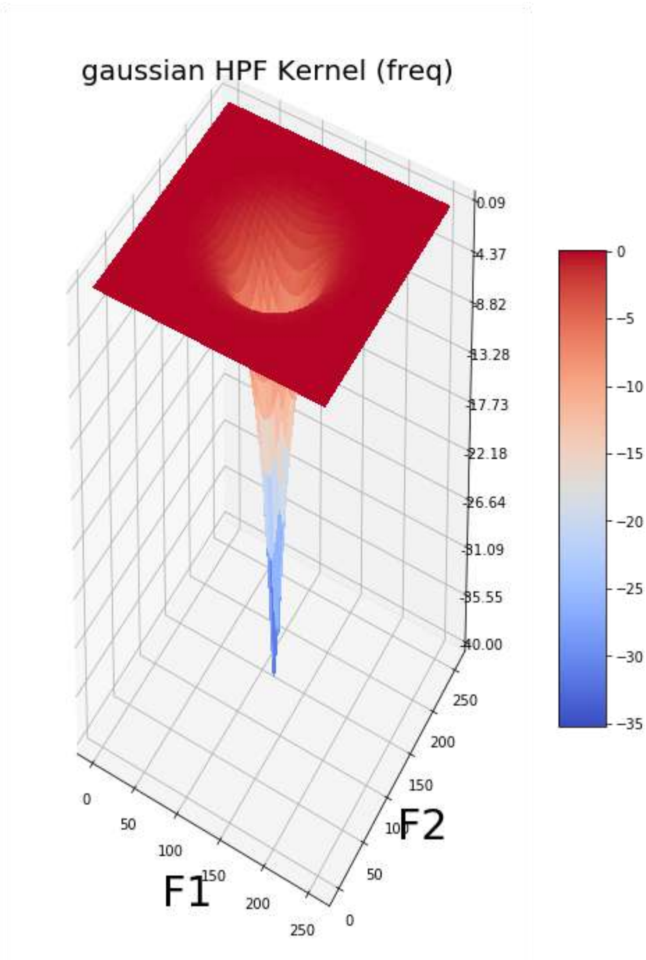
In [255]:
```
plot_HPF_3d(im, gaussian, D0)
```



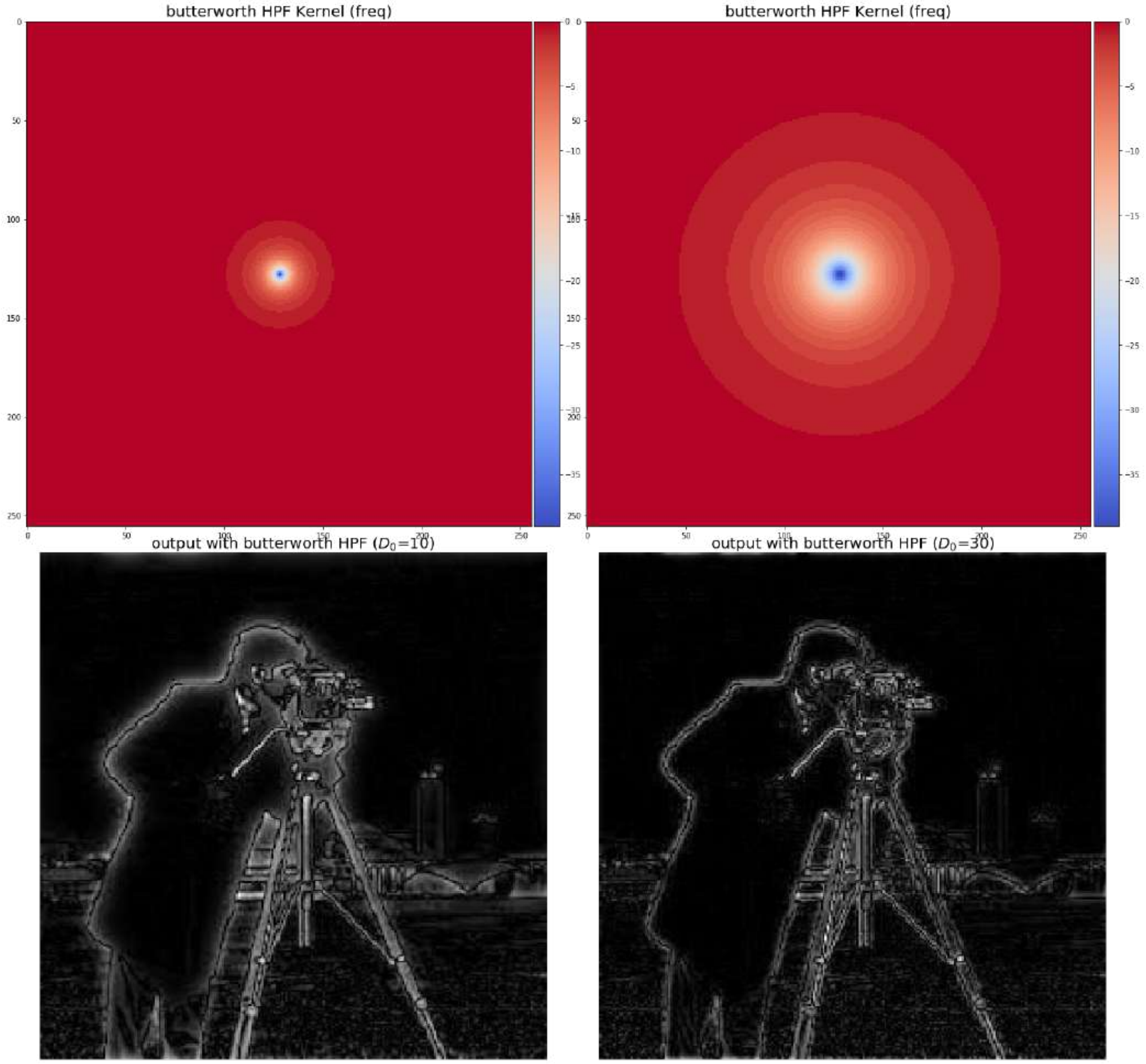


In [256]:
```
plot_filter_3d(im.shape, gaussian, D0)
```

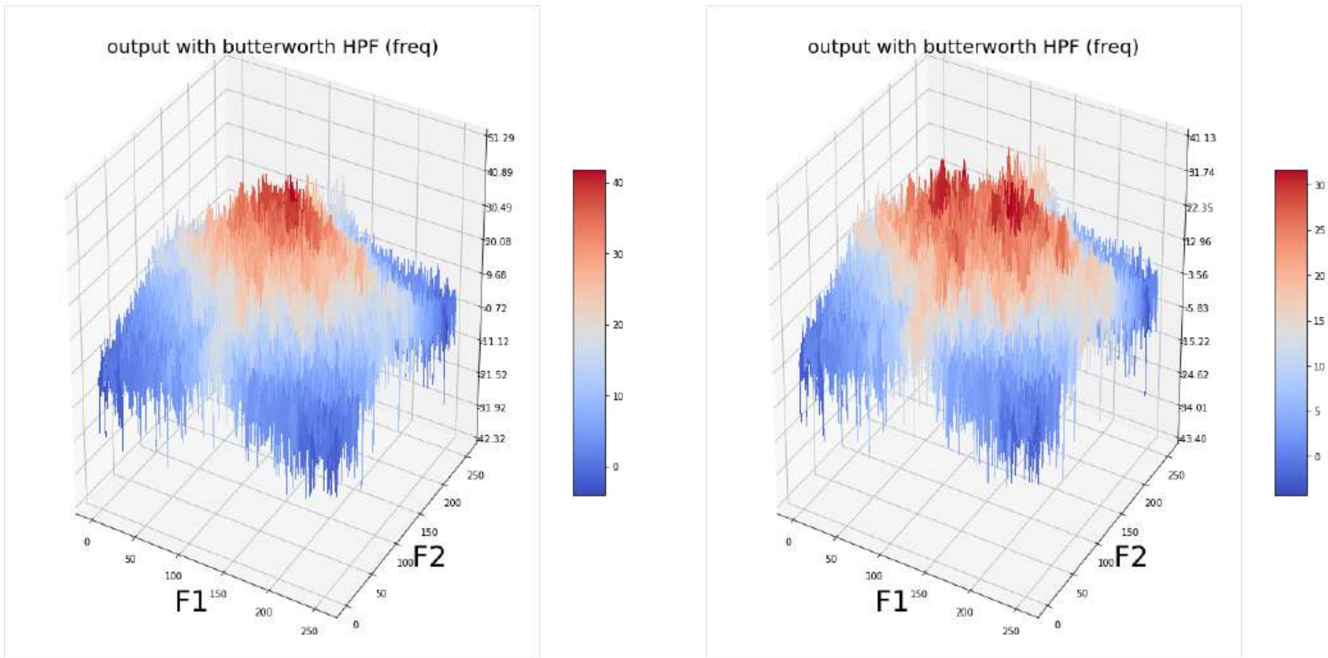




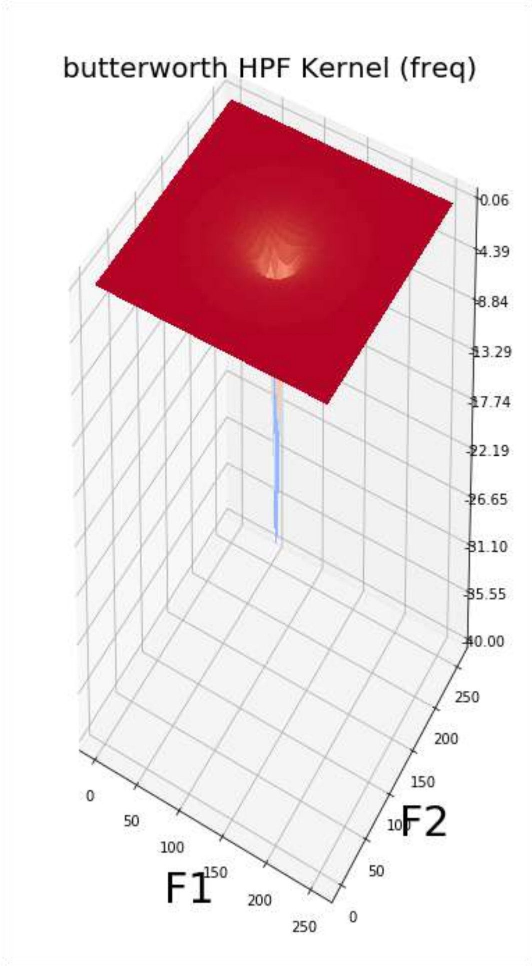
```
In [258]: plot_HPF(im, butterworth, D0)
```

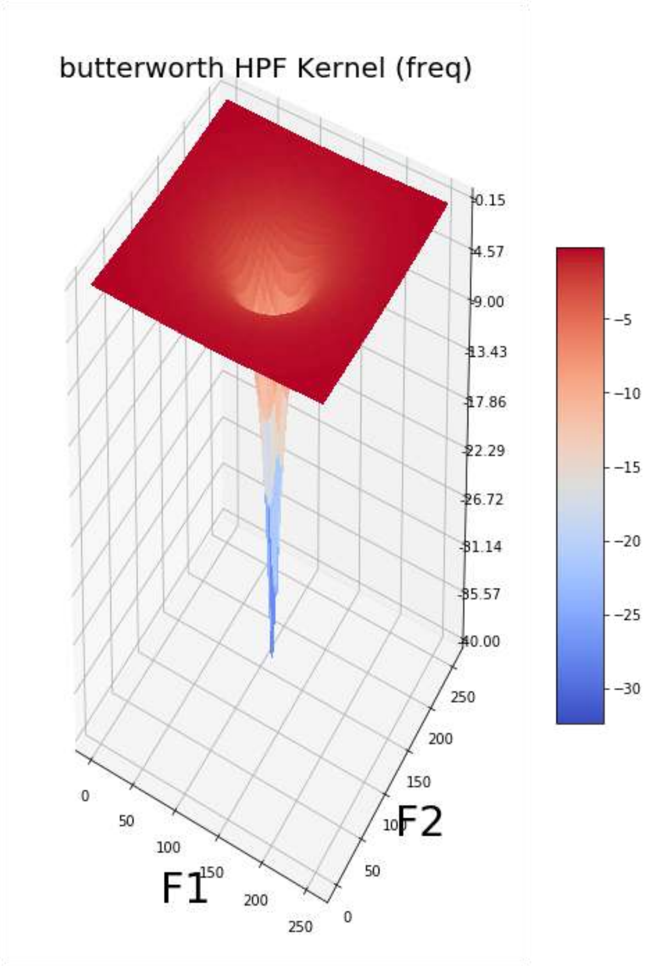



```
In [259]: plot_HPF_3d(im, butterworth, D0)
```



```
In [260]: plot_filter_3d(im.shape, butterworth, D0)
```





5. Implement Homomorphic Filters

In [315]:

```
plt.figure(figsize=(21,17))
plt.gray()
plt.subplots_adjust(0,0,1,0.95,0.01,0.05)
plt.subplot(221), plt.imshow(image), plt.axis('off'), plt.title('original image', size=
plt.subplot(222), plt.imshow(image_filtered), plt.axis('off'), plt.title('filtered image
plt.subplot(223), plt.imshow(image_edges), plt.axis('off'), plt.title('original image ed
plt.subplot(224), plt.imshow(image_filtered_edges), plt.axis('off'), plt.title('filtered
plt.show()
```