In [ ]:

```python
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import math

mpl.rc('image', cmap='grey')
```

In [ ]:

```python
# Creating image
def CreateImage(size):
    width = height = size
    image = np.zeros((width, height))

    return image


######################################
#  FUNCTIONS FOR DEFINING APERTURES  #
######################################
def SetDiffractionGrating(size, slitWidth, value=1):

    image = CreateImage(size)

    i = 0
    even = False
    while i < len(image):

        if even == True:
            image[:,i:i+slitWidth] = value
            even = False

        elif even == False:
            even = True

        i += slitWidth

    return image

def SetSineGrating(size, slitWidth, amplitude=1):

    image = CreateImage(size)

    i = 0
    while i < len(image):
        image[:,i] = np.sin(i/slitWidth)

        i += 1

    return image

def RectangularAperture(size, rect, amplitude):

    image = CreateImage(size)

    startX, endX, startY, endY = rect
    y = startY
    while y < endY:

        x = startX
        while x < endX:

            image[y][x] = amplitude

            x += 1
        y += 1

    return image

def CircularAperture(gridSize, centre, radius, amplitude=1):

    image = CreateImage(gridSize)

    centreX, centreY = centre

    y = 0
    while y < len(image):

        x = 0
        while x < len(image[:,0]):

            if (x - centreX)**2 + (y - centreY)**2 < radius**2:
                image[y][x] = amplitude

            x += 1
        y += 1

    return image
```

In [ ]:

```
# Function to determine the FFT of a given aperture function
def GetFFT(image):
    fft = np.fft.ifftshift(image)
    fft = np.fft.fft2(fft)
    fft = np.fft.fftshift(fft) # Shift the zero-frequency component to the center of the spectrum.
    fft = fft / np.max(fft)    # https://numpy.org/doc/stable/reference/generated/numpy.fft.fftshift.html#numpy.fft.fftshift

    return fft
```

In [ ]:

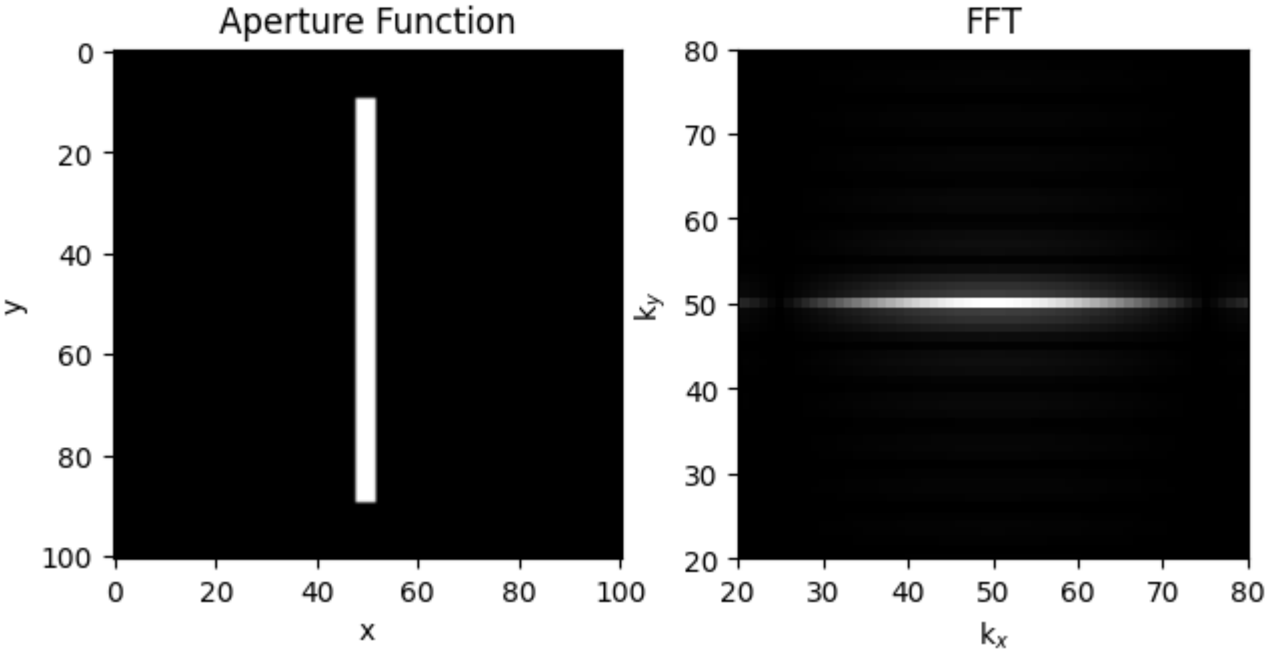
```
# Function to plot side-by-side the aperture function and resulting fft
def ApertureTransform_Multiplot(apertureFunction, fftImageBounds=[]):

    fig, axes = plt.subplots(1, 2)
    fig.tight_layout()

    axes[0].imshow(apertureFunction)

    fftImage = abs(GetFFT(apertureFunction)) # abs as fft contains complex components

    axes[1].imshow(fftImage)

    # Labels
    axes[0].set_title("Aperture Function")
    axes[1].set_title("FFT")

    axes[0].set_xlabel("x")
    axes[0].set_ylabel("y")
    axes[1].set_xlabel("k$_x$")
    axes[1].set_ylabel("k$_y$")


    # "Zooming in" on the fft image
    if fftImageBounds != []:
        xLower, xUpper, yLower, yUpper = fftImageBounds

        axes[1].set_xlim(xLower, xUpper)
        axes[1].set_ylim(yLower, yUpper)
```

In [ ]:

```
rectangularAperture0 = RectangularAperture(101, [48, 52, 10, 90], 1)
rectangularAperture1 = RectangularAperture(101, [45, 55, 10, 90], 1)
rectangularAperture2 = RectangularAperture(101, [40, 60, 10, 90], 1)
```

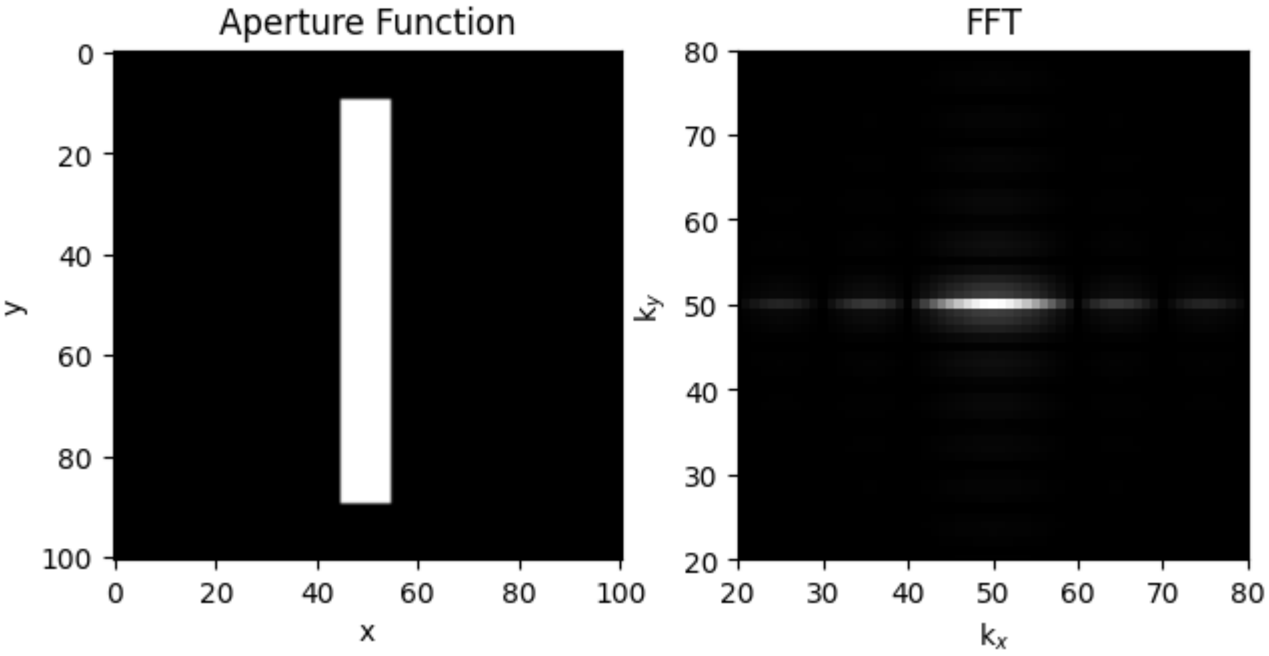
In [ ]:

```
ApertureTransform_Multiplot(rectangularAperture0, fftImageBounds=[20, 80, 20, 80])
```



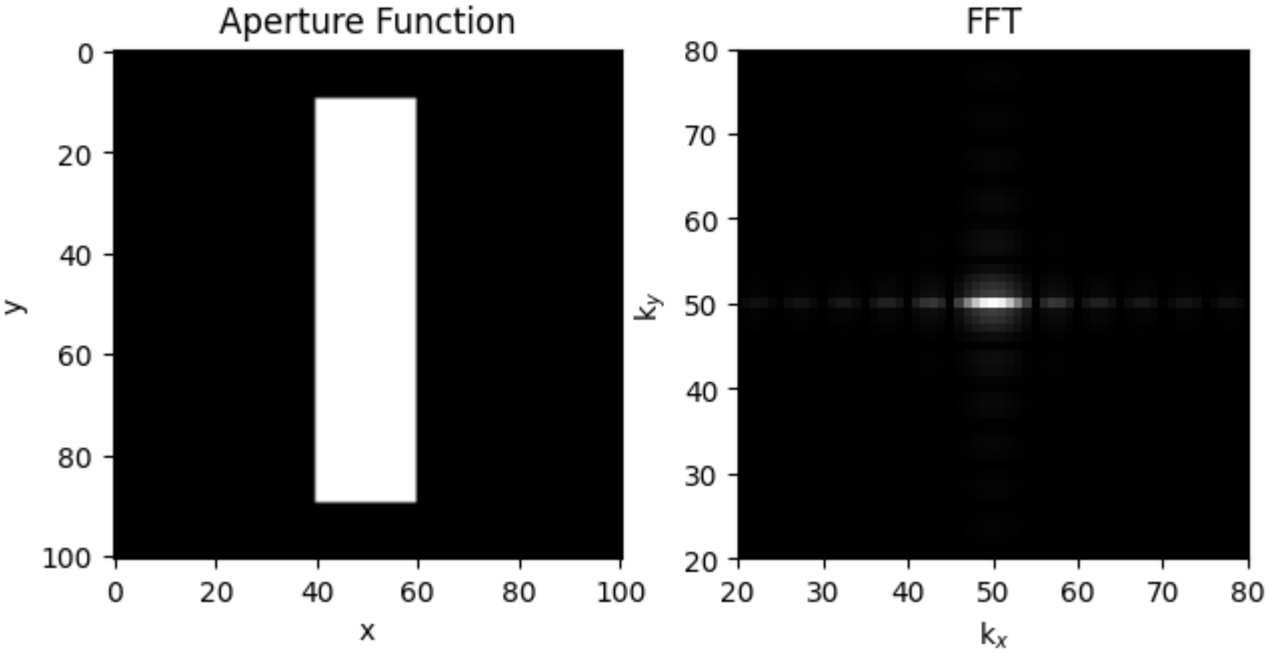
In [ ]:

```
ApertureTransform_Multiplot(rectangularAperture1, fftImageBounds=[20, 80, 20, 80])
```



In [ ]:

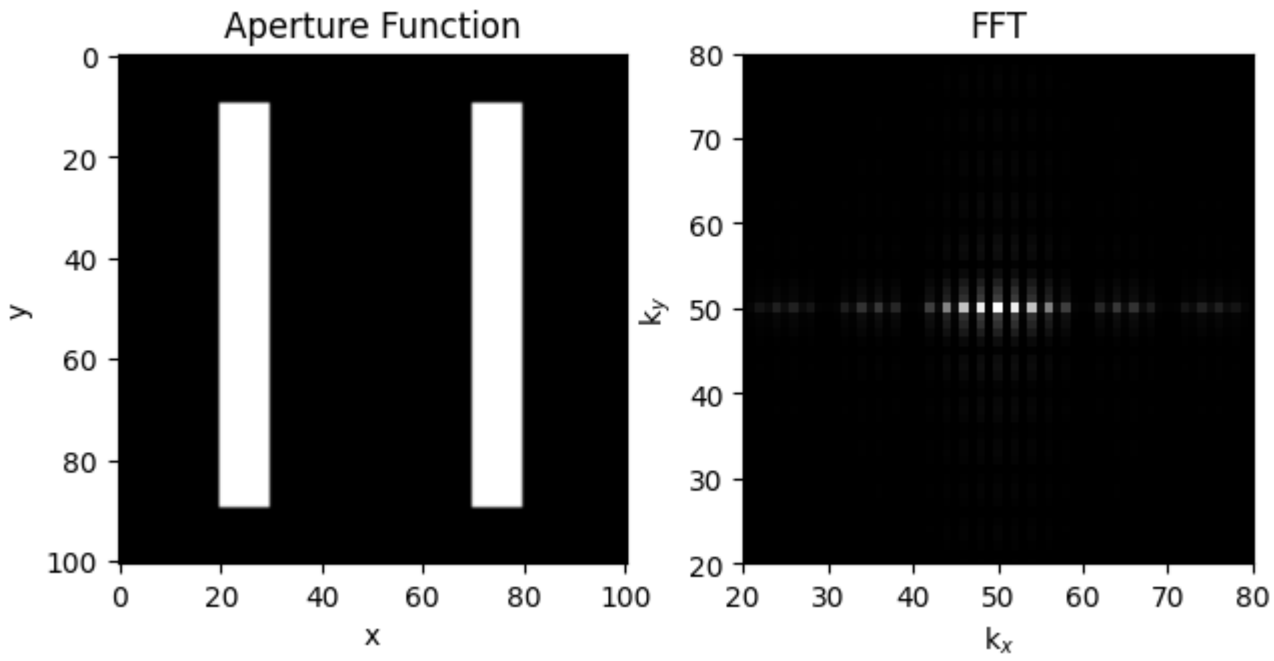
```
ApertureTransform_Multiplot(rectangularAperture2, fftImageBounds=[20, 80, 20, 80])
```


Aperture Function

FFT

In [ ]:

```
doubleSlitA = RectangularAperture(101, [20, 30, 10, 90], 1)
doubleSlitB = RectangularAperture(101, [70, 80, 10, 90], 1)


doubleSlitAperture = doubleSlitA + doubleSlitB

ApertureTransform_Multiplot(doubleSlitAperture, fftImageBounds=[20, 80, 20, 80])
```



Fourier Filtering

In [ ]:

```python
from PIL import Image
rootPath = "/home/daraghhollman/Main/labData/fourierOptics/post_variable_slit/"
```

In [ ]:

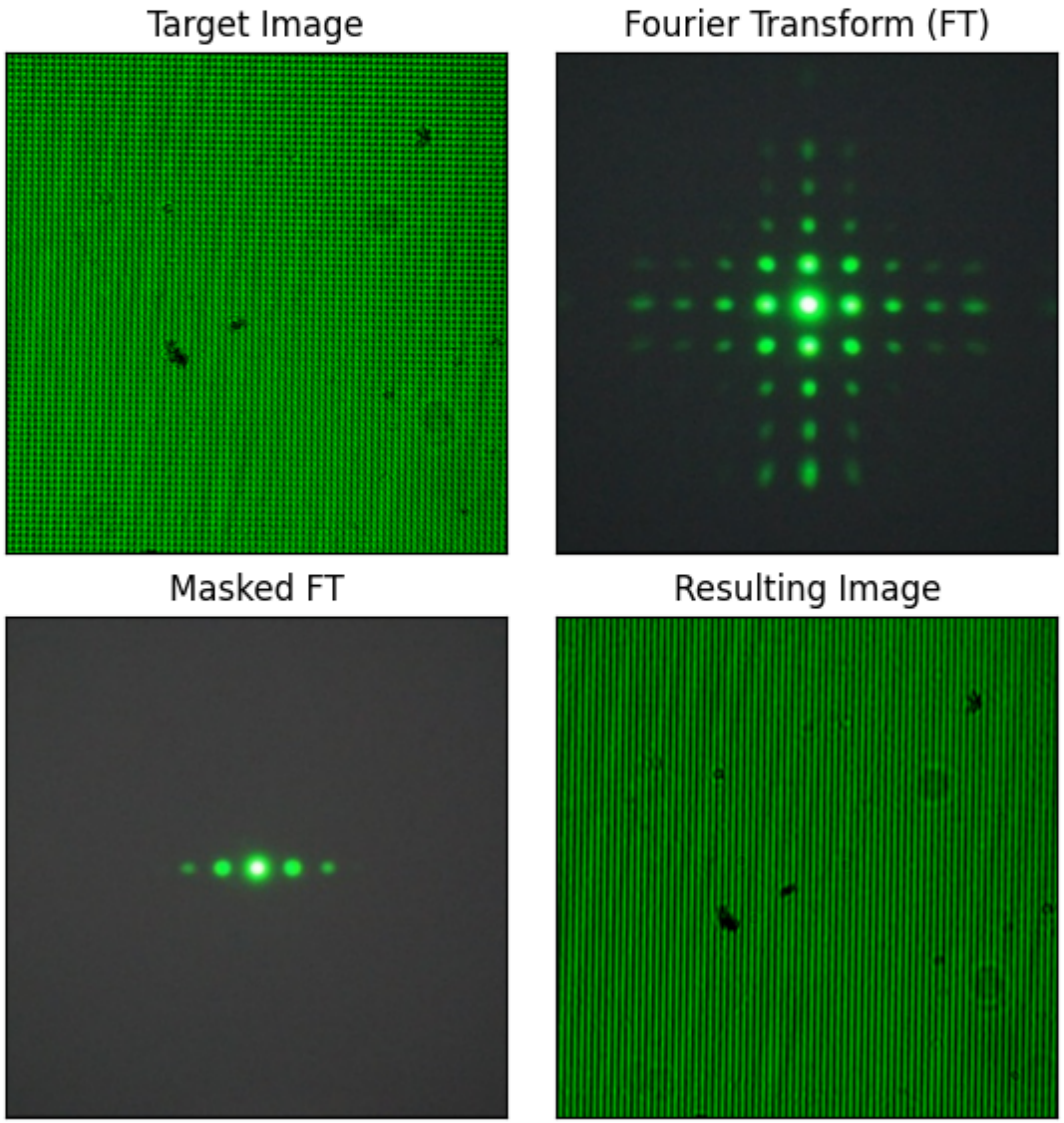
```python
def OpenImage(path, square=False):
    im = Image.open(rootPath + path)
    imArray = np.array(im)

    # Images are originally 1080 x 1440, resizing to 1080 x 1080 by taking 180 px off left and right
    if square: imArray = imArray[:, 180:-180]

    return imArray
```

In [ ]:

```
def FourierMultiplot(imageBefore, fourierTransform, maskedFourierTransform, imageAfter):
    imagePaths = [imageBefore, fourierTransform, maskedFourierTransform, imageAfter]

    fig, axes = plt.subplots(2, 2, figsize=(6,6))
    fig.tight_layout()
    plt.subplots_adjust(wspace=0.1, hspace=0.1)

    axes = axes.flatten()

    labels = ["Target Image", "Fourier Transform (FT)", "Masked FT", "Resulting Image"]

    for i, path in enumerate(imagePaths):

        if (i == 0) or (i == 3):
            im = OpenImage(path, square=True)
        else:
            im = OpenImage(path)

        axes[i].imshow(im)
        axes[i].set_xticks([])
        axes[i].set_yticks([])
        axes[i].set_title(labels[i])
```

In [ ]:

```
# ex8
FourierMultiplot("8/ex8Before.png", "manual/ex8_ft.png", "manual/ex8_masked.png", "8/ex8After.png")
```



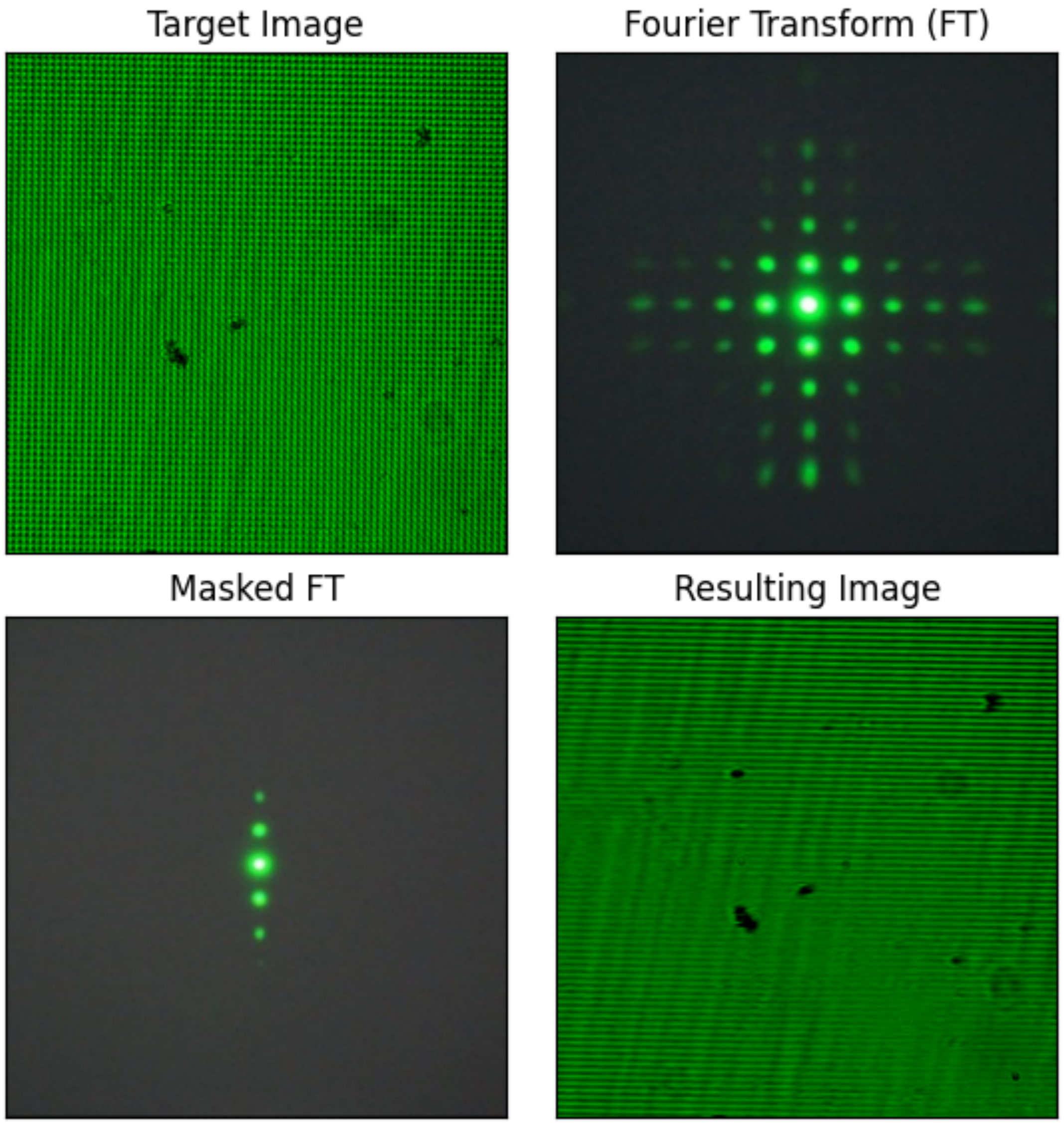
Target Image

Fourier Transform (FT)

Masked FT

Resulting Image

```
In [ ]: FourierMultiplot("8/ex8Before.png", "manual/ex8_ft.png", "manual/ex9_masked.png", "9/ex9After.png")
```



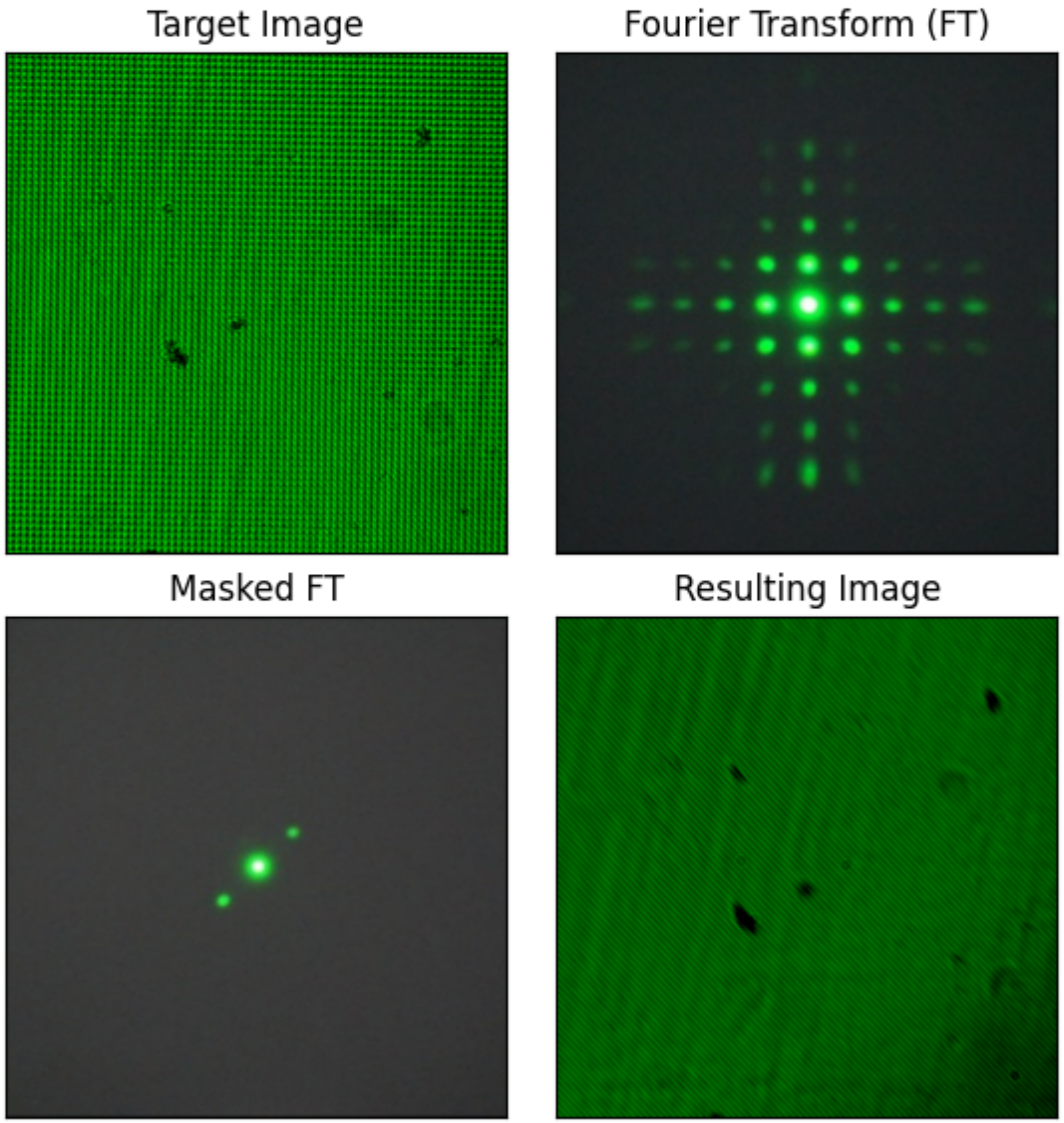
Target Image

Fourier Transform (FT)

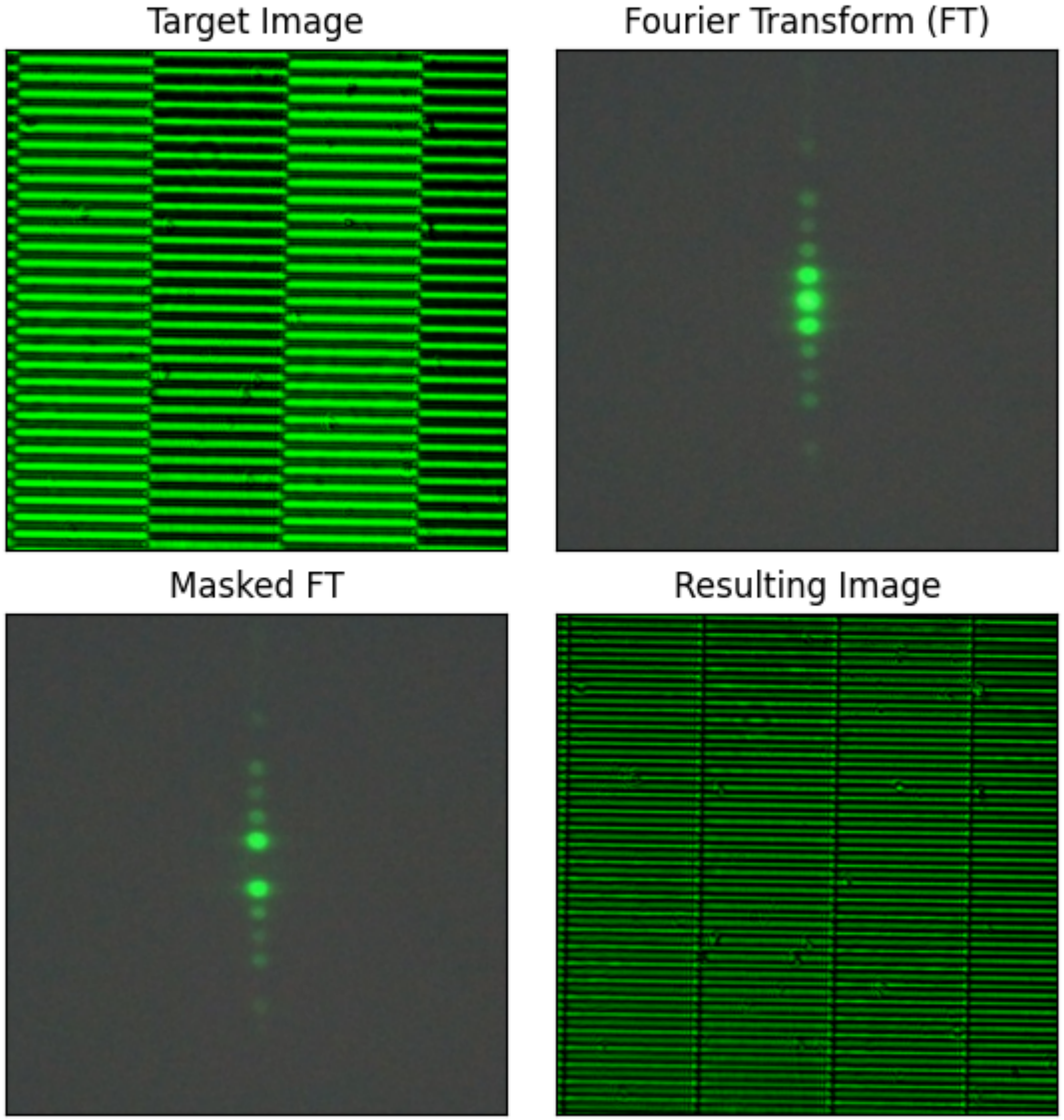
Masked FT

Resulting Image

```
In [ ]: FourierMultiplot("8/ex8Before.png", "manual/ex8_ft.png", "manual/ex10_masked.png", "10/ex10After.png")
```



```
In [ ]: FourierMultiplot("babinet/babinetBefore.png", "manual/babinets_ft.png", "manual/babinets_masked.png", "babinet/babinetAfter.png")
```



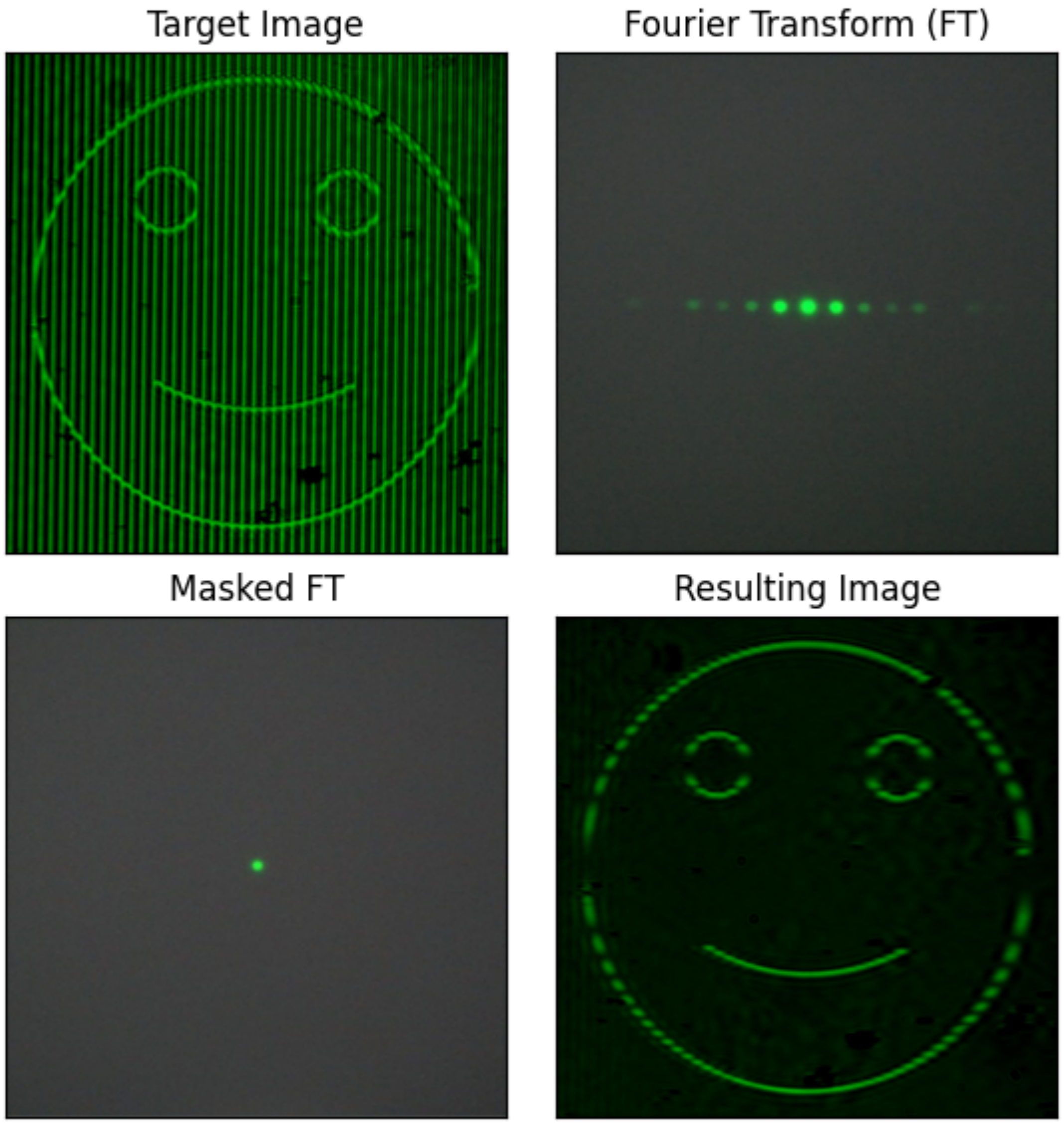
Target Image

Fourier Transform (FT)

Masked FT

Resulting Image

```
In [ ]: FourierMultiplot("face/faceBefore.png", "manual/face_ft.png", "manual/face_masked.png", "face/faceAfter.png")
```



Target Image

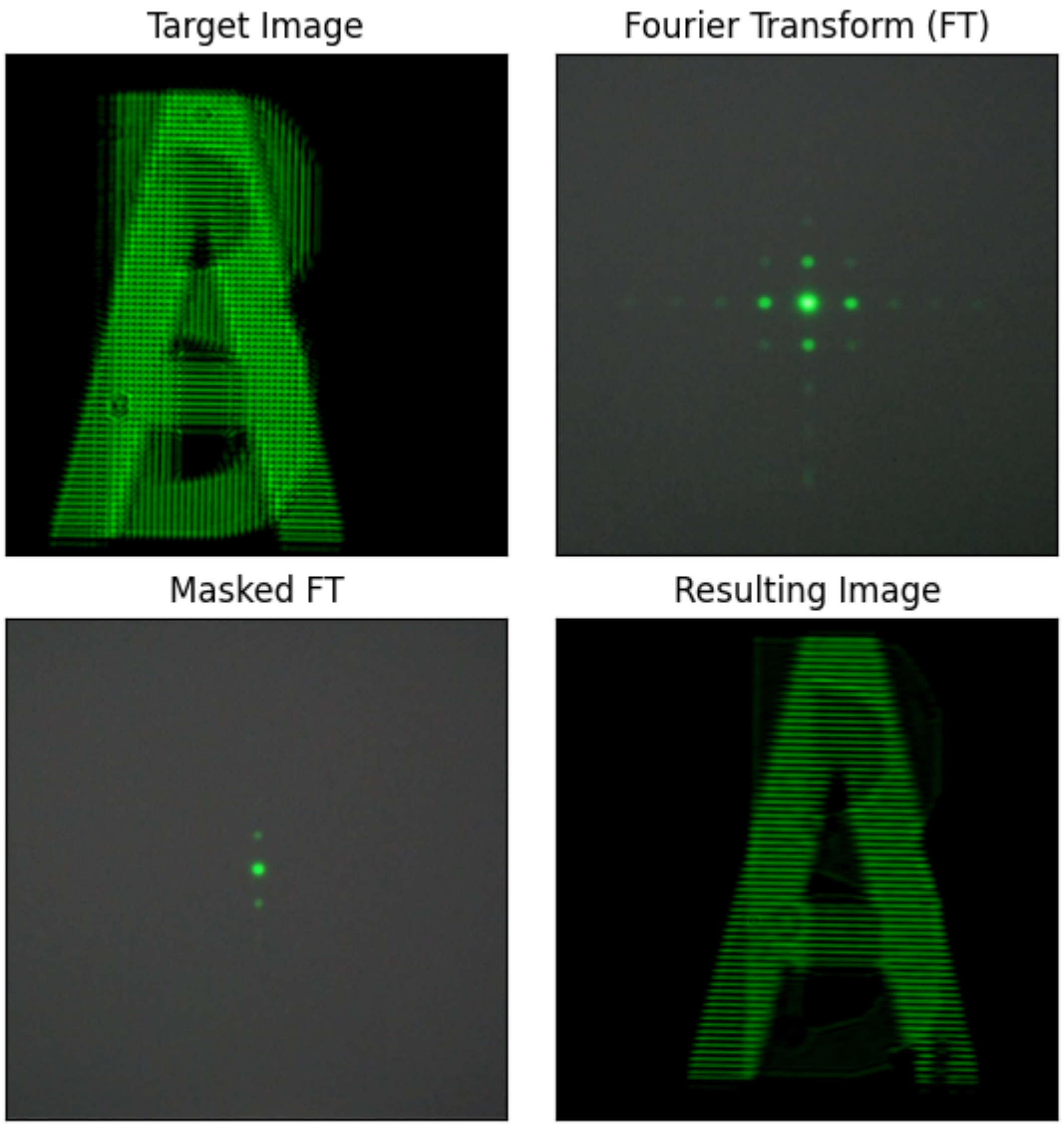
Fourier Transform (FT)

Masked FT

Resulting Image

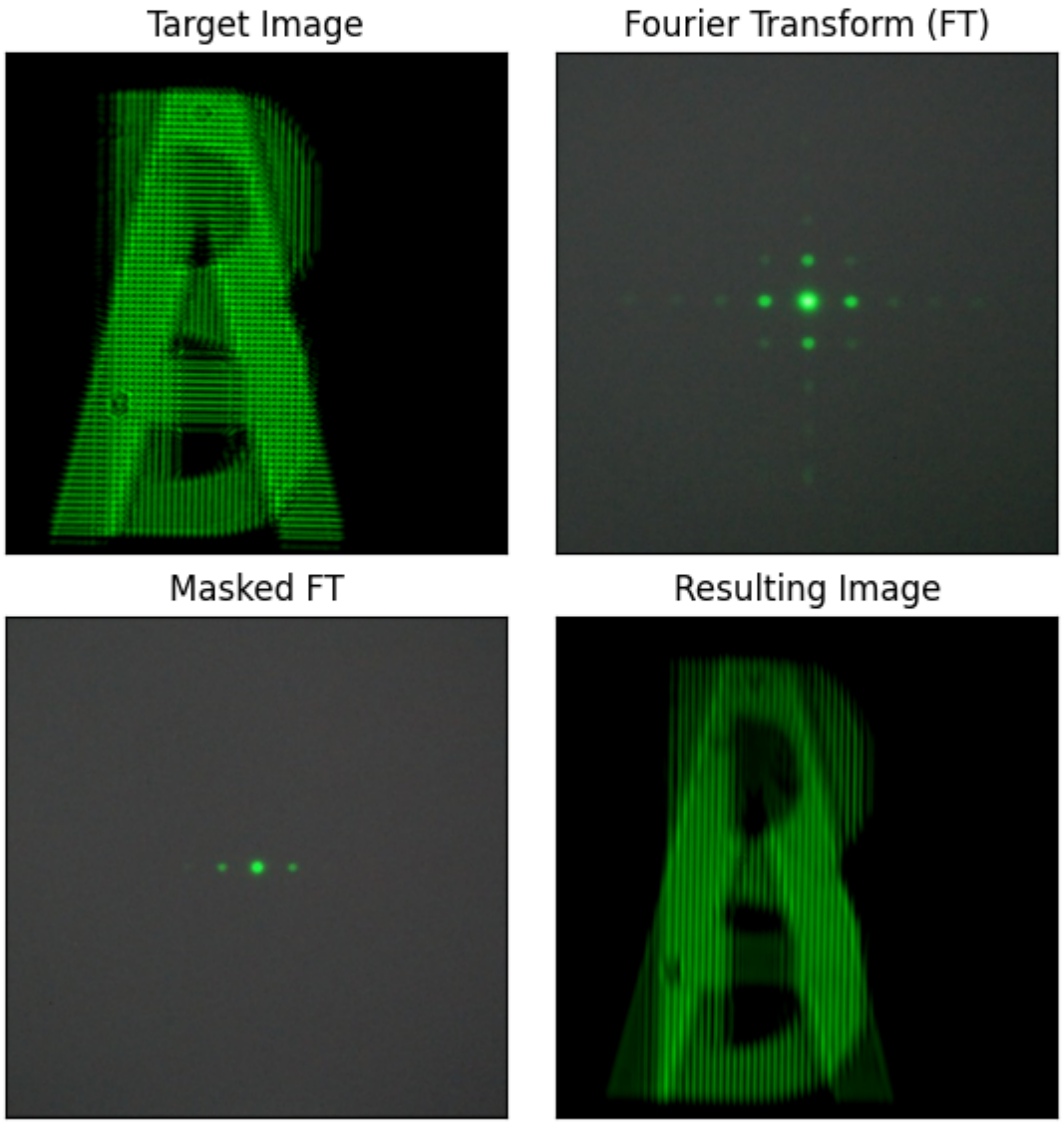
In [ ]:

```
FourierMultiplot("letters/abBefore.png", "manual/letters_ft.png", "manual/letters_a_masked.png", "letters/aAfter.png")
```



In [ ]:

```
FourierMultiplot("letters/abBefore.png", "manual/letters_ft.png", "manual/letters_b_masked.png", "letters/bAfter.png")
```



Target Image

Fourier Transform (FT)

Masked FT

Resulting Image