ECE5554 – Computer Vision
Lecture 4a – Fourier Transforms

Creed Jones, PhD

Course Updates

- HW assignment 1 is due TONIGHT at 11:59 PM
- HW2 is posted – due next Wednesday, June 20
- Quiz 2 is TOMORROW
  - 6 PM Tuesday to 3 AM Wednesday, Eastern time
  - covers lectures 3 and 4
  - ten questions, twenty minutes
    - questions are a bit simpler than last time

Today's Objectives

- The Fourier Transform
  - The concept
  - The math
- The 2D Fourier Transform
- Discrete Fourier Transforms
- Basis Functions
- 2D FFT of images
- numpy fft2()
  - shifting results
- example of fingerprint enhancement
- example of smoothing in the Fourier domain

Transforms are used to yield alternative representations of functions (say, $f(t)$) in an alternative domain ($F(\tau)$), where the alternative representation has some desired properties

- Useful transforms should be unique, invertible and practically computable
- The transform domain may be a more complicated representation
  - Fourier transforms will, in general, transform from the real number domain to complex numbers
  - The "kernel trick" used in Support Vector Machines transforms from a lower-dimensional space to a much higher – perhaps infinite(!) – dimensional space
- In many cases the transform domain has a useful interpretation

Why might we go to the trouble of using transforms?
- easier calculation of key operations
- more compact representation
- useful interpretation

- Important operations can be done more efficiently
  - Example: convolution in the time domain multiplication in the Fourier domain
  - (also in the Laplace domain)

- A function that would take *n* samples to represent to sufficient accuracy can often be represented in much fewer than *n* data points in a transform domain
  - Many periodic signals can be represented by a small number of Fourier coefficients
  - Wavelet transformation is the key step in many data compression schemes

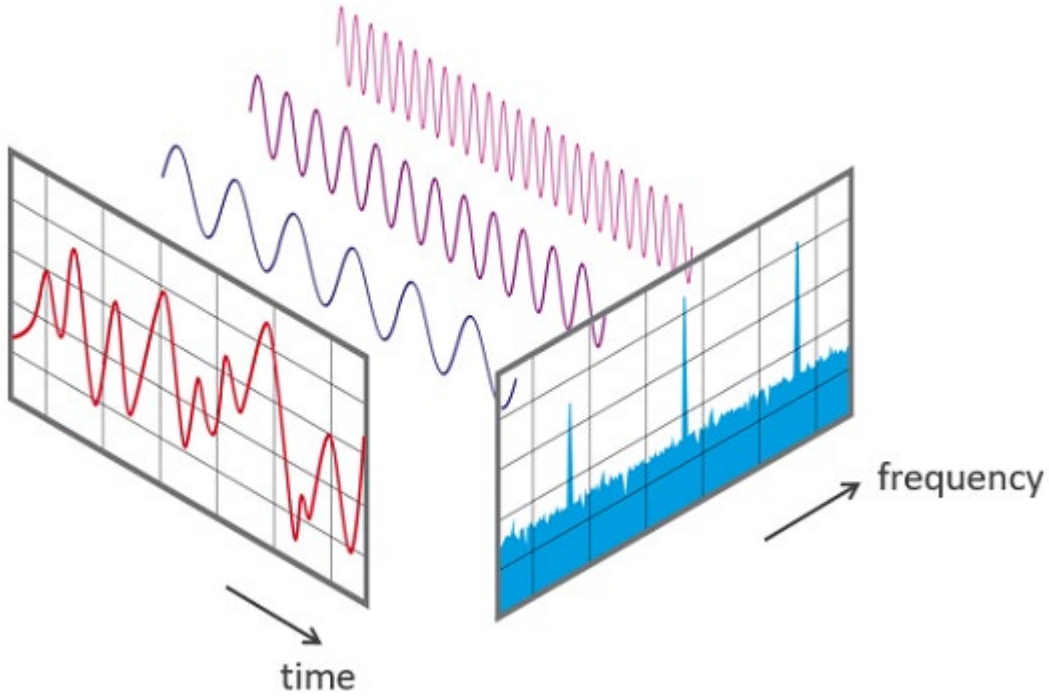
- Transform domain representations often reveal interesting behavior
  - Fourier coefficients relate to temporal (or spatial) frequency

A one-dimensional Fourier transform of a time-based function reveals the *temporal frequencies* that make up the function

$$F(\omega) = \int_{-\infty}^{\infty} f(t)\, e^{-j\omega t} dt$$

- The exponential is a (complex) sinusoid
- The family of sinusoids is orthogonal, so for any ω the product will select activity at that *frequency*
- The transform is invertible…

$$f(t) = \int_{-\infty}^{\infty} F(\omega)\, e^{j\omega t} d\omega$$



Sinusoidal Orthogonality

- m,n: integer, $T_c=1/f_0$

$$\int_0^{T_c} \cos(2\pi m f_0 t)\cdot\cos(2\pi n f_0 t)dt = \begin{cases} \frac{T_c}{2} & (m=n) \\ 0 & (m\neq n) \end{cases} \longrightarrow \text{Orthogonal}$$

$$\int_0^{T_c} \sin(2\pi m f_0 t)\cdot\sin(2\pi n f_0 t)dt = \begin{cases} \frac{T_c}{2} & (m=n) \\ 0 & (m\neq n) \end{cases} \longrightarrow \text{Orthogonal}$$

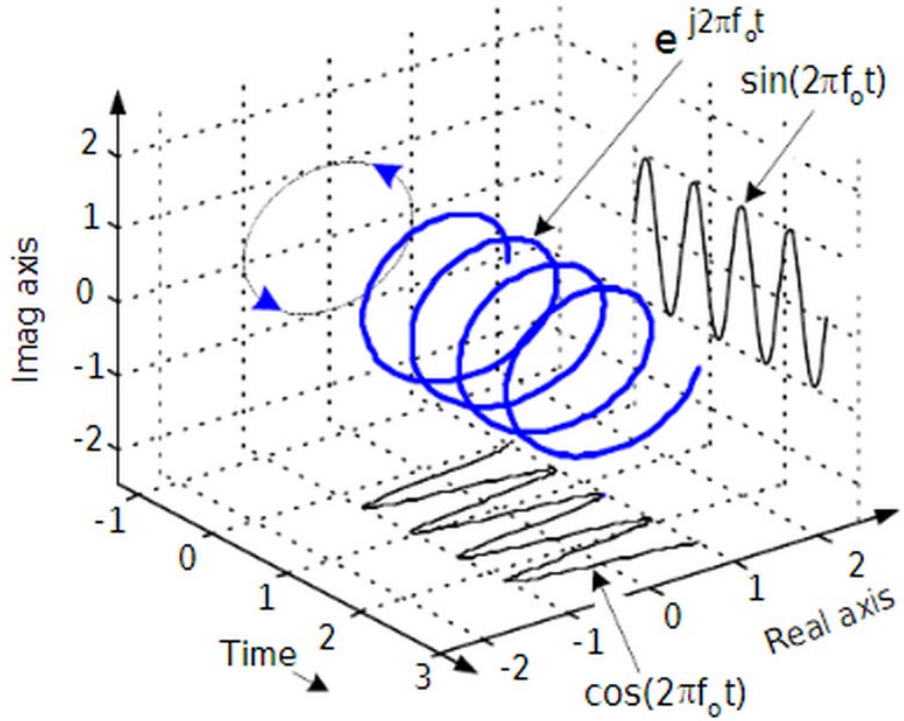
$$\int_0^{T_c} \cos(2\pi m f_0 t)\cdot\sin(2\pi n f_0 t)dt = 0 \longrightarrow \text{Orthogonal}$$

Markus Ahonen

Look closely at the continuous Fourier transform

$$F(\omega) = \int_{-\infty}^{\infty} f(t)\, e^{-j\omega t} dt$$

- t is the domain variable of the initial function
  - Often thought of as time
- j is the complex root $j = \sqrt{-1}$
- $e^{-j\omega t}$ is the complex exponential
- Euler's equation tells us that
  $$e^{-j\omega t} = \cos(\omega t) - j\sin(\omega t)$$
- We integrate the product of the function with a pair of sinusoids, out of phase, where one is imaginary
- To compute a single value of $F(\omega)$ we integrate across the entire range of t



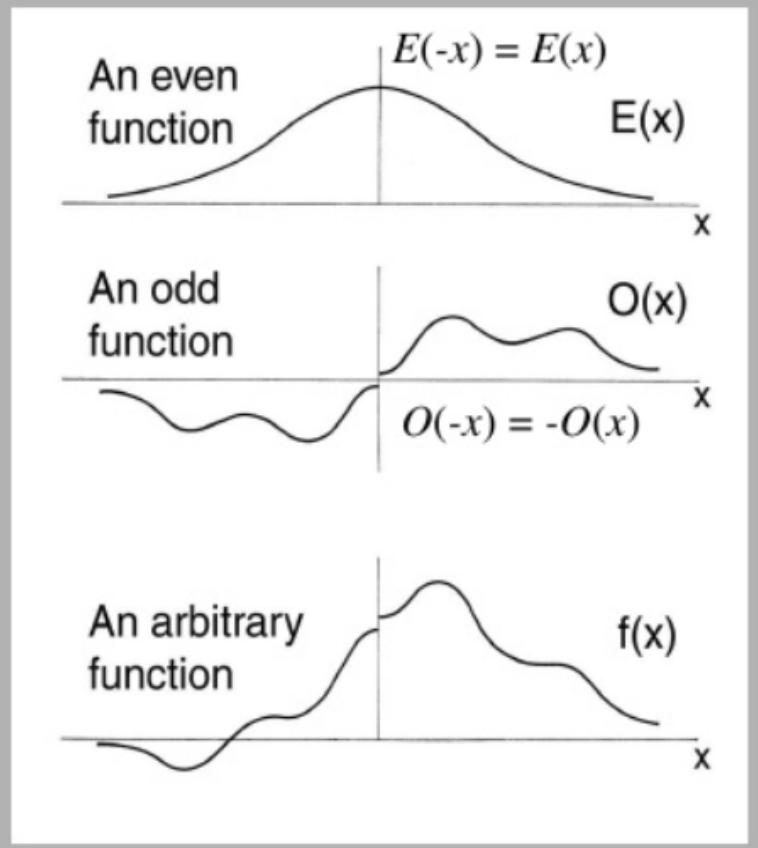
In general, the Fourier transform operates on a complex function (t) and generates a complex function of one variable (ω)

$$F(\omega) = \int_{-\infty}^{\infty} f(t)\, e^{-j\omega t} dt$$

- If the function $f(t)$ is real-valued everywhere (all imaginary parts zero), the transform $F(\omega)$ will generally still have complex values (non-zero imaginary part)

- Specifically:
  - *Even-valued real* functions (symmetric about the Y-axis) have purely real transforms
  - *Odd-valued real* functions (anti-symmetric about the Y axis) have purely imaginary transforms
  - Most real functions are a sum of each of these

Any function can be written as the sum of an even and an odd function



$$E(x) \equiv [f(x) + f(-x)]/2$$

$$O(x) \equiv [f(x) - f(-x)]/2$$

$$\Downarrow$$

$$f(x) = E(x) + O(x)$$

Even functions: $f(x) = f(-x)$

Odd functions: $f(x) = -f(-x)$

Hermitian: $f(x) = f^*(-x)$

Input Domain	Frequency Domain
Real, even	Real, even
Real, odd	Imaginary, odd
Real, no symmetry	Hermitian
Imaginary, even	Imaginary, even
Imaginary, odd	Real, odd
Imaginary, no symmetry	Anti-Hermitian
Hermitian	Real, no symmetry
Anti-Hermitian	Imaginary, no symmetry
Complex, even	Complex, even
Complex, odd	Complex, odd
Complex, no symmetry	Complex, no symmetry

If the function is not time-varying but rather a function of a spatial dimension x, then the same formulation applies – we are determining the spatial frequency behavior of $f(x)$

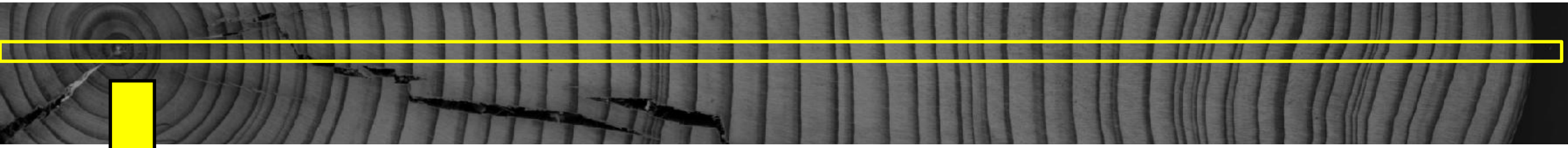
$$F(\omega) = \int_{-\infty}^{\infty} f(x)\, e^{-j\omega x} dx$$

- Spatial frequencies relate to the spacing between values in the signal
- Since the exponential term is complex, Fourier transforms of real-valued spatial signals generally have real and imaginary components, just like time signals
  - We often examine the magnitude: $\|F(\omega)\|$
- The spatial Fourier transform is invertible…

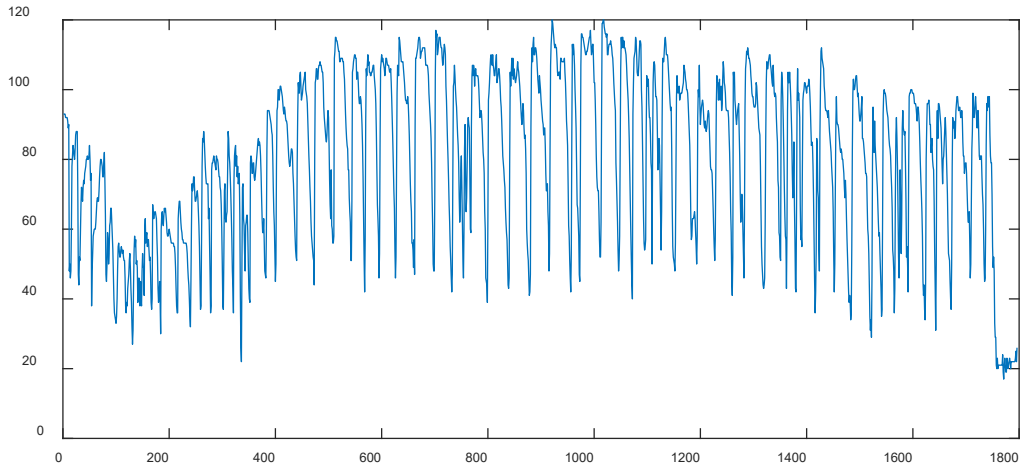
$$f(x) = \int_{-\infty}^{\infty} F(\omega)\, e^{j\omega x} d\omega$$

Consider a two-dimensional signal – an image of a cross-section of a tree trunk (showing the rings)

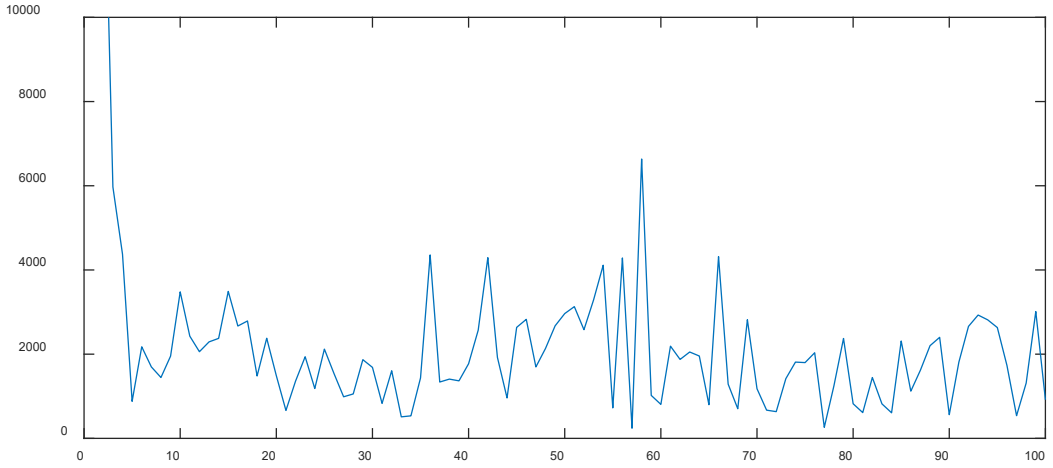
By averaging a small slice of the image, we obtain a one-dimensional spatial signal – and can compute its spatial Fourier transform



Intensity plot vs. X



$\|F(\omega)\|$



The advantages of the Fourier transform are its useful properties – below – and its interpretation as a frequency-based representation of the original signal

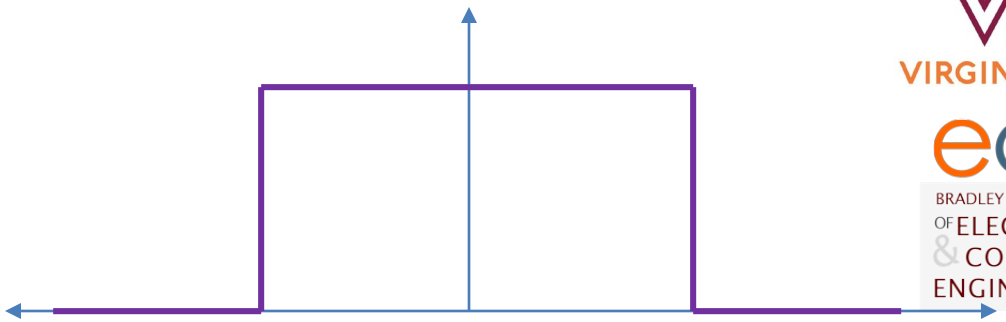
Property	Signal		Transform
superposition	$f_1(x) + f_2(x)$		$F_1(\omega) + F_2(\omega)$
shift	$f(x - x_0)$		$F(\omega)e^{-j\omega x_0}$
reversal	$f(-x)$		$F^*(\omega)$
convolution	$f(x) * h(x)$		$F(\omega)H(\omega)$
correlation	$f(x) \otimes h(x)$		$F(\omega)H^*(\omega)$
multiplication	$f(x)h(x)$		$F(\omega) * H(\omega)$
differentiation	$f'(x)$		$j\omega F(\omega)$
domain scaling	$f(ax)$		$1/aF(\omega/a)$
real images	$f(x) = f^*(x)$	$\Leftrightarrow$	$F(\omega) = F(-\omega)$
Parseval's Theorem	$\sum_x [f(x)]^2$	$=$	$\sum_\omega [F(\omega)]^2$

Table 3.1 in Szeliski.

Some useful properties of Fourier transforms. The original transform pair is:

$$F(\omega) = \mathcal{F}\{f(x)\} = \int_{-\infty}^{\infty} F(x)\, e^{-j\omega x} dx$$

Consider a real function that has the value 1 from –k to k and is zero everywhere else

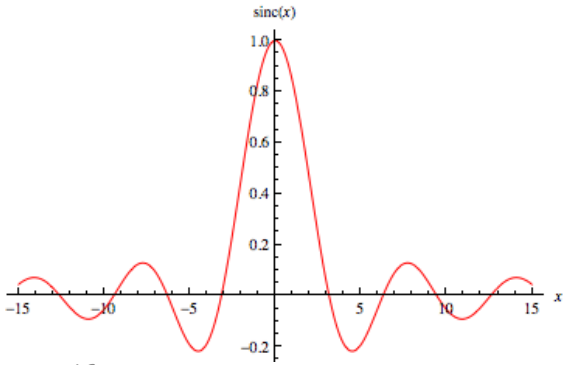


$$f(x) = \begin{cases} 1, -k \le x \le k \\ 0, elsewhere \end{cases} \qquad F(\omega) = \int_{-\infty}^{\infty} f(x)\, e^{-j\omega x} dx = \int_{-k}^{k} 1\, e^{-j\omega x} dx + \int_{elsewhere} 0\, e^{-j\omega x} dx$$

$$= \int_{-k}^{k} 1\, e^{-j\omega x} dx = \frac{-1}{j\omega} e^{-j\omega x}\Big|_{x=-k}^{x=k} = \frac{-1}{j\omega} e^{-j\omega k} - \frac{-1}{j\omega} e^{-j\omega(-k)} = \frac{1}{j\omega} e^{j\omega k} - \frac{1}{j\omega} e^{-j\omega k}$$

$$= \frac{1}{j\omega}\left(e^{j\omega k} - e^{-j\omega k}\right) = \frac{1}{j\omega}(\cos \omega k + j \sin \omega k - \cos(-\omega k) - j \sin(-\omega k)) = \frac{1}{j\omega}(2j \sin \omega k)$$

$$\boldsymbol{F(\omega) = \frac{2\, sin\, \omega k}{\omega}}$$



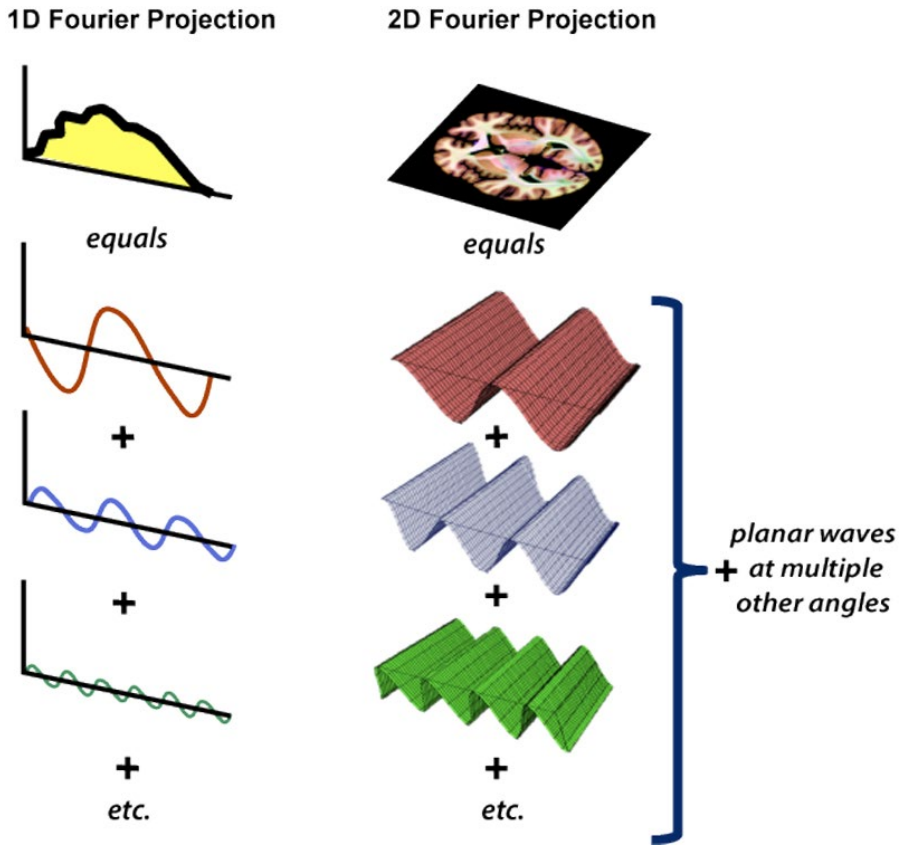
If the input is a function of two dimensions, we can expand the definition to produce the 2D Fourier transform:

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-j(ux+vy)}dx\,dy$$

- It is invertible…

$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v)e^{j(ux+vy)}du\,dv$$

- The result is still generally a complex quantity (having real and imaginary parts)
- Functions f with significant periodicity in a direction will have notable Fourier components
  - Direction is indicated in the 2D Fourier plane



For discrete signals, there are discrete formulations of both the 1D and 2D Fourier transforms

- 1D Discrete Fourier Transform:

$$F(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x) e^{-2\pi j \frac{ux}{N}}, \qquad u \in [0, \dots, N-1]$$

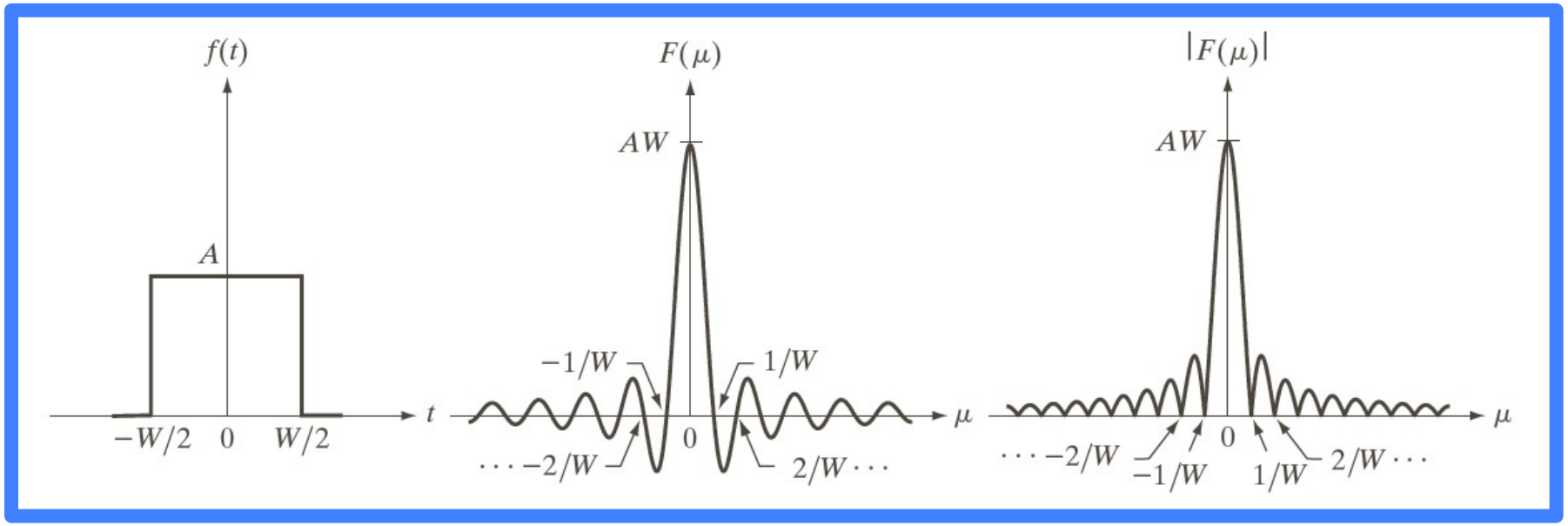
$$f(x) = \sum_{u=0}^{N-1} F(u) e^{2\pi j \frac{ux}{N}}, \qquad u \in [0, \dots, N-1]$$

- 2D Discrete Fourier Transform:

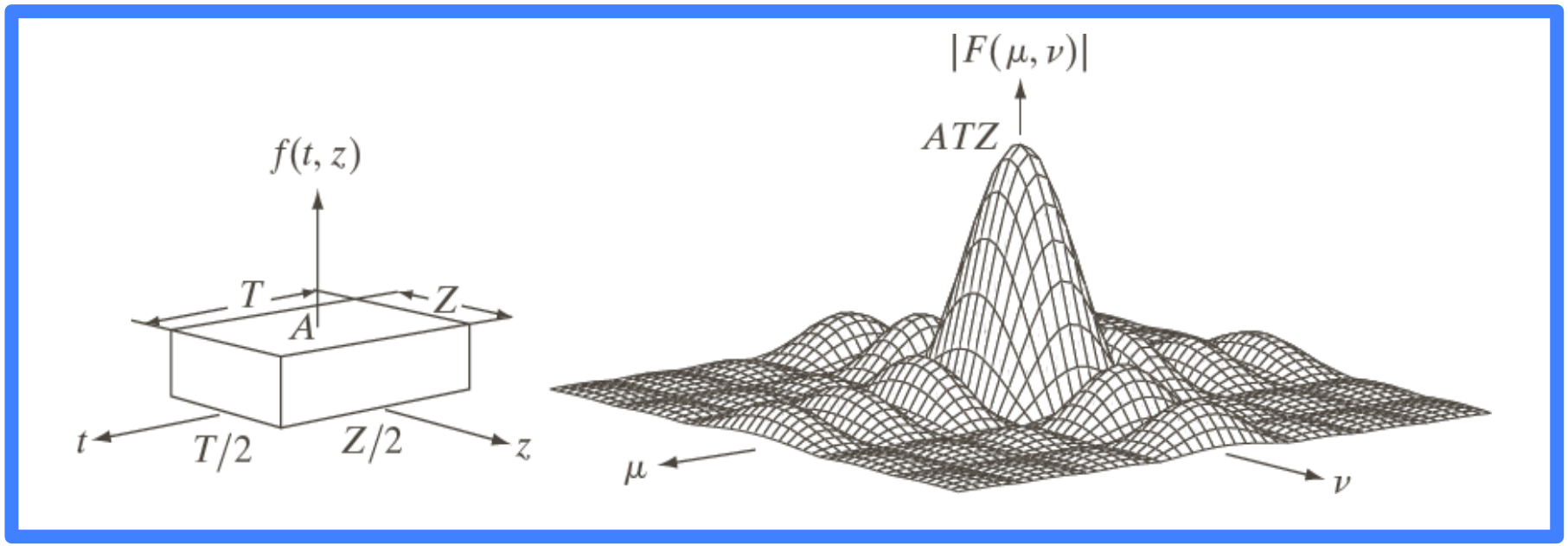
$$F(u,v) = \frac{1}{N}\frac{1}{M}\sum_{x=0}^{N-1}\sum_{y=0}^{M-1} f(x,y) e^{-2\pi j\left(\frac{ux}{N}+\frac{vy}{M}\right)},$$
$$u \in [0, \dots, N-1], v \in [0, \dots, M-1]$$

$$f(x,y) = \sum_{u=0}^{N-1}\sum_{v=0}^{M-1} F(u,v) e^{2\pi j\left(\frac{ux}{N}+\frac{vy}{M}\right)},$$
$$x \in [0, \dots, N-1], y \in [0, \dots, M-1]$$

- 1-D

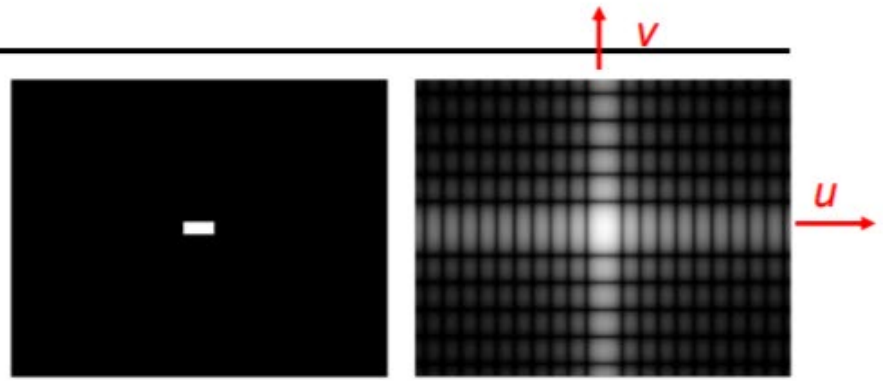

f(t)
A
−W/2
0
W/2
t
F(μ)
AW
−1/W
1/W
0
· · · −2/W
2/W · · ·
μ
|F(μ)|
AW
· · · −2/W
−1/W
0
1/W
2/W · · ·
μ

- 2-D


f(t, z)
T
Z
A
t
T/2
Z/2
z
|F(μ, ν)|
ATZ
μ
ν

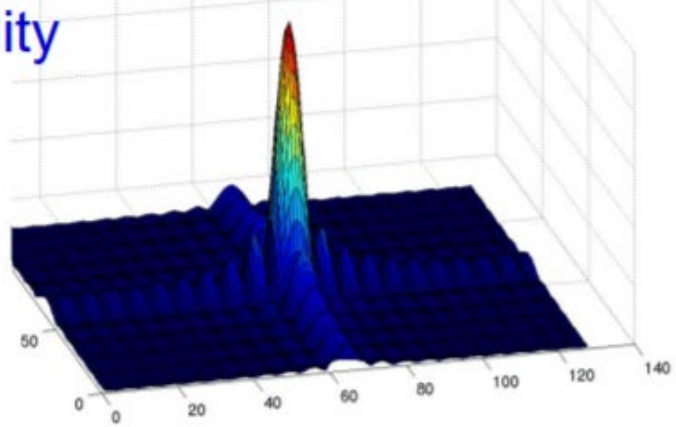
FT pair example 1

rectangle centred at origin
with sides of length X and Y



$$
\begin{aligned}
F(u,v) &= \int\int f(x,y)e^{-j2\pi(ux+vy)}dxdy, \\
&= \int_{-X/2}^{X/2} e^{-j2\pi ux}dx \int_{-Y/2}^{Y/2} e^{-j2\pi vy}dy, \quad \text{separability} \\
&= \left[\frac{e^{-j2\pi ux}}{-j2\pi u}\right]_{-X/2}^{X/2} \left[\frac{e^{-j2\pi vy}}{-j2\pi v}\right]_{-Y/2}^{Y/2}, \\
&= \frac{1}{-j2\pi u}\left[e^{-juX} - e^{juX}\right] \frac{1}{-j2\pi v}\left[e^{-jvY} - e^{jvY}\right], \\
&= XY\left[\frac{sin(\pi Xu)}{\pi Xu}\right]\left[\frac{sin(2\pi Yv)}{\pi Yv}\right] \\
&= XY\mathrm{sinc}(\pi \mathrm{Xu})\mathrm{sinc}(\pi \mathrm{Yv}).
\end{aligned}
$$

$f(x,y)$ $|F(u,v)|$



$|F(u,v)|$

A. Zissermann, Oxford Univ.

FT pair example 2

Gaussian centred on origin

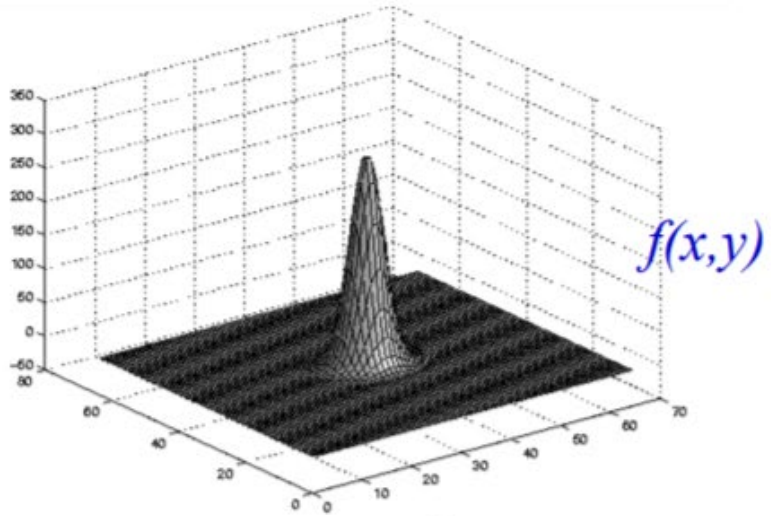
$$f(r) = \frac{1}{2\pi\sigma^2} e^{-r^2/2\sigma^2}$$

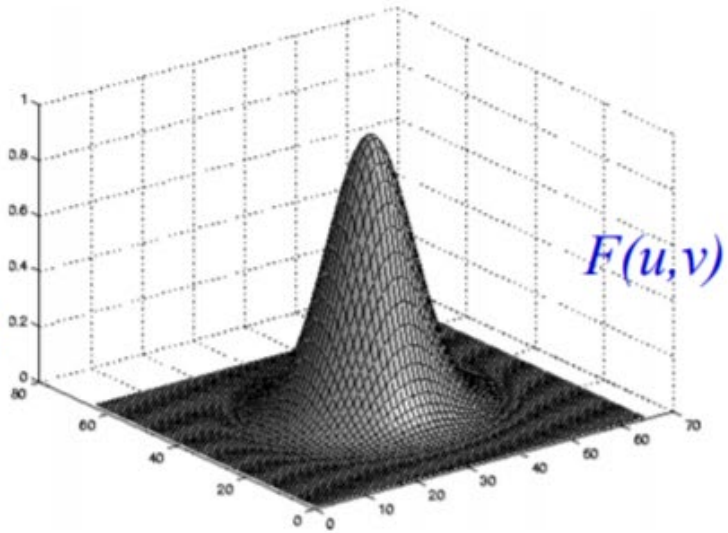
where $r^2 = x^2 + y^2$.

$$F(u,v) = F(\rho) = e^{-2\pi^2\rho^2\sigma^2}$$

where $\rho^2 = u^2 + v^2$.

- FT of a Gaussian is a Gaussian
- Note inverse scale relation



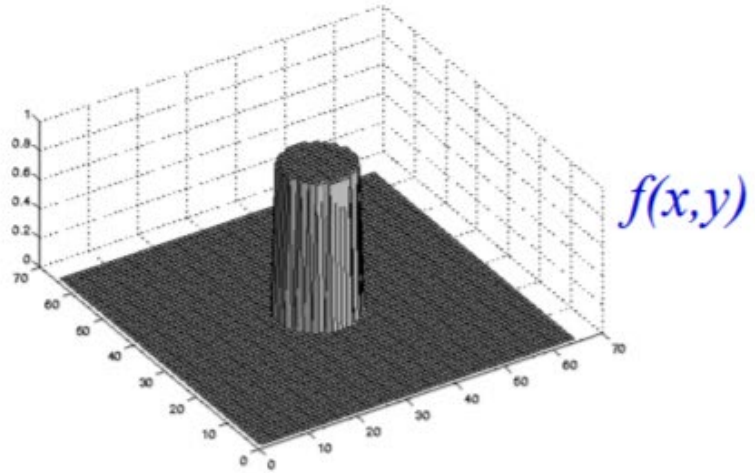


A. Zissermann, Oxford Univ.

FT pair example 3

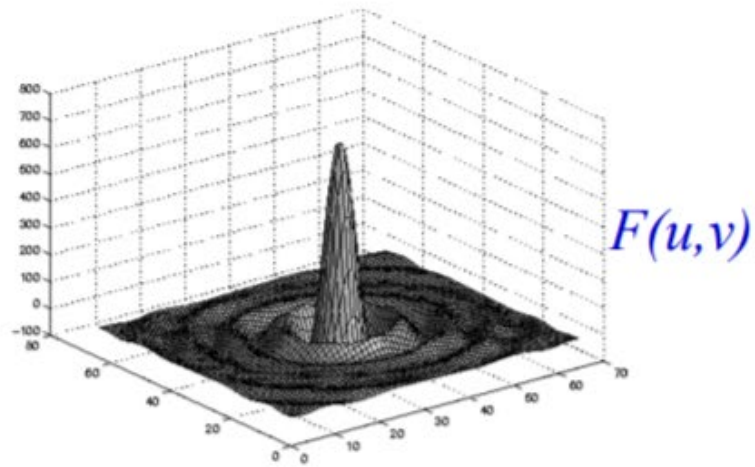
Circular disk unit height and radius *a* centred on origin

$$f(x,y) = \begin{cases} 1, & |r| < a, \\ 0, & |r| \geq a. \end{cases}$$



$$F(u,v) = F(\rho) = aJ_1(\pi a\rho)/\rho$$

where $J_1(x)$ is a Bessel function.

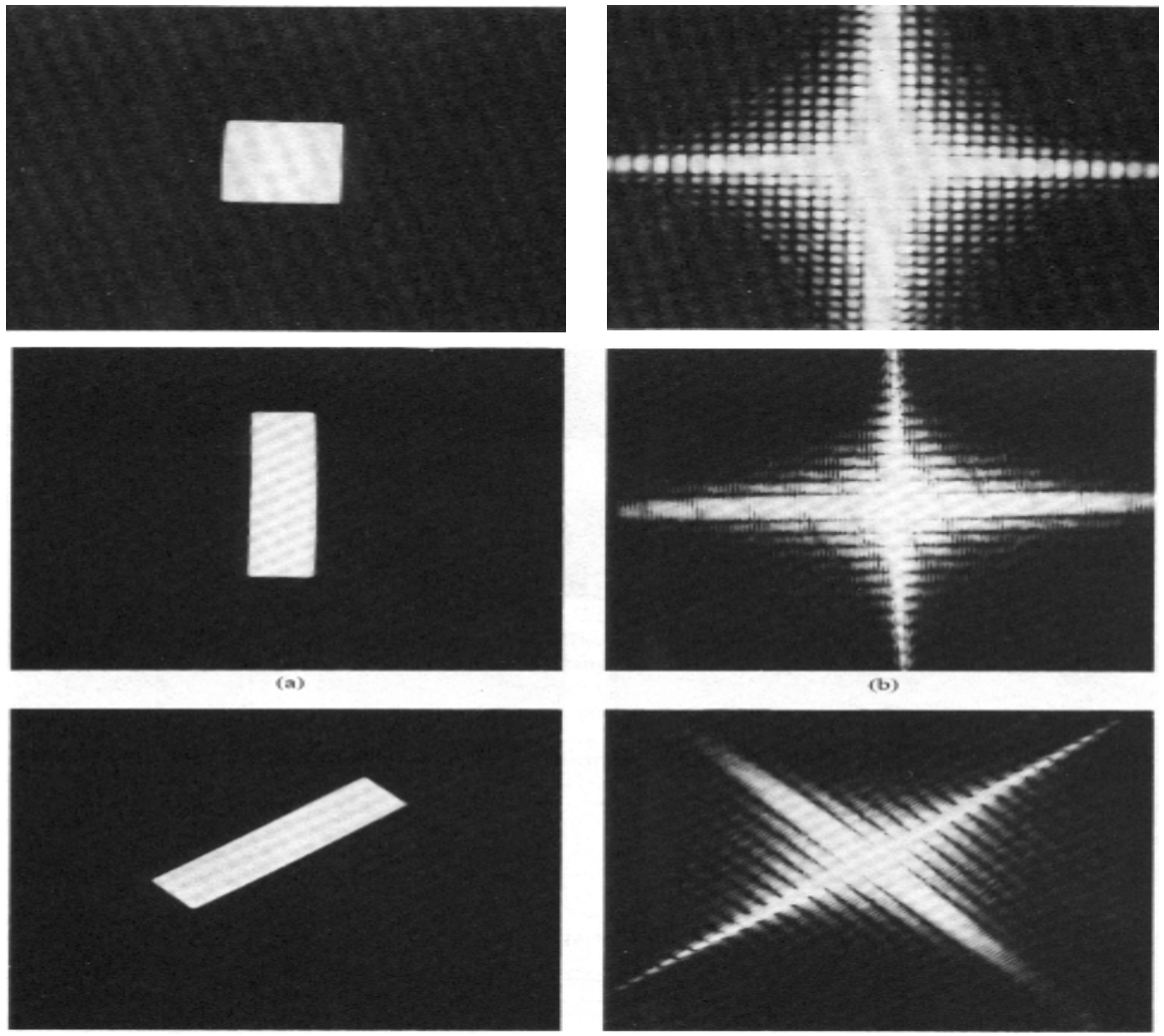


- rotational symmetry
- a '2D' version of a sinc

A. Zissermann, Oxford Univ.

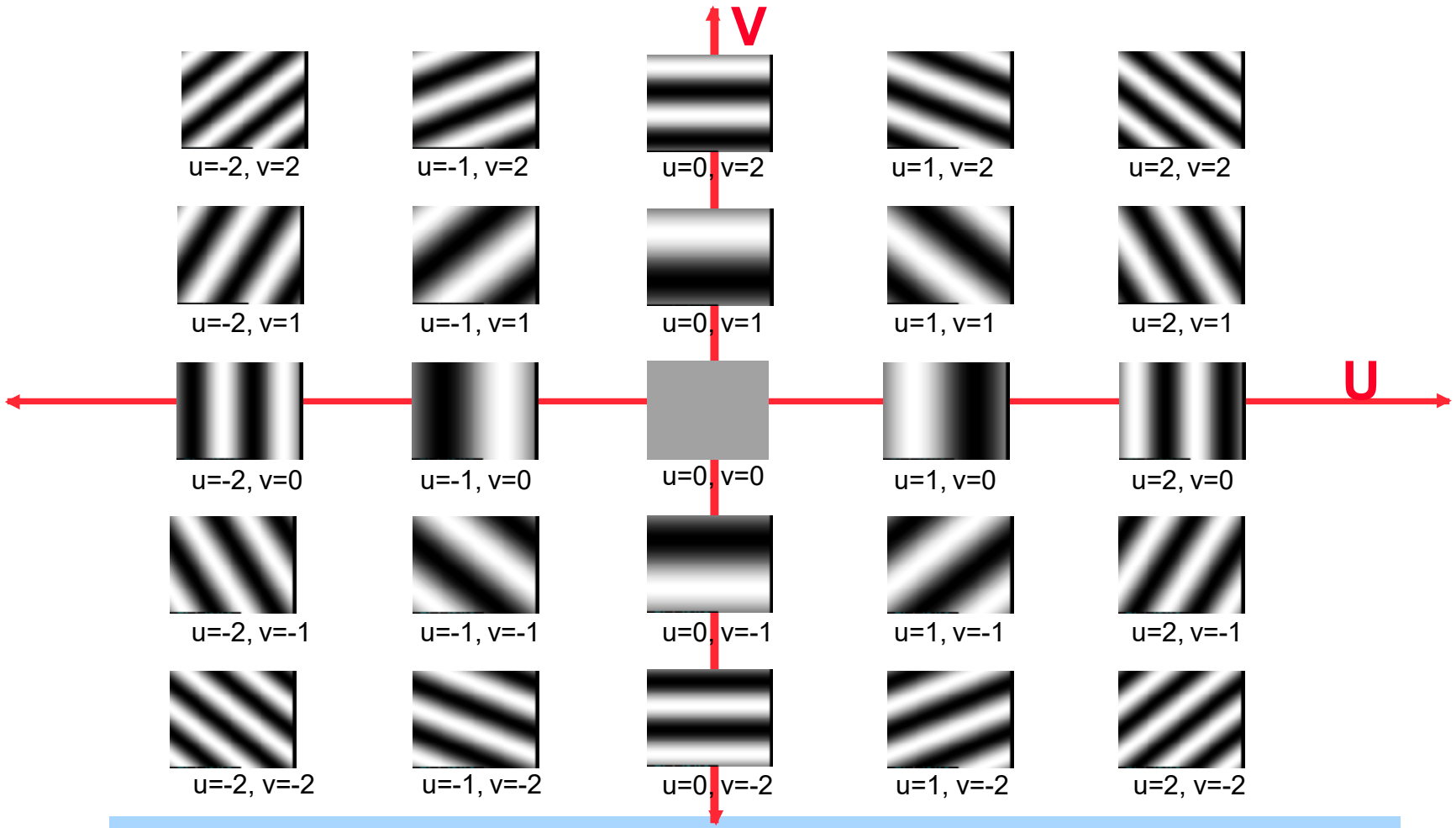
Examples of the 2D Fourier transform of spatial signals

Image Domain



Frequency Domain – Magnitude Shown

2D Basis Functions - $e^{2\pi j(ux+vy)}$



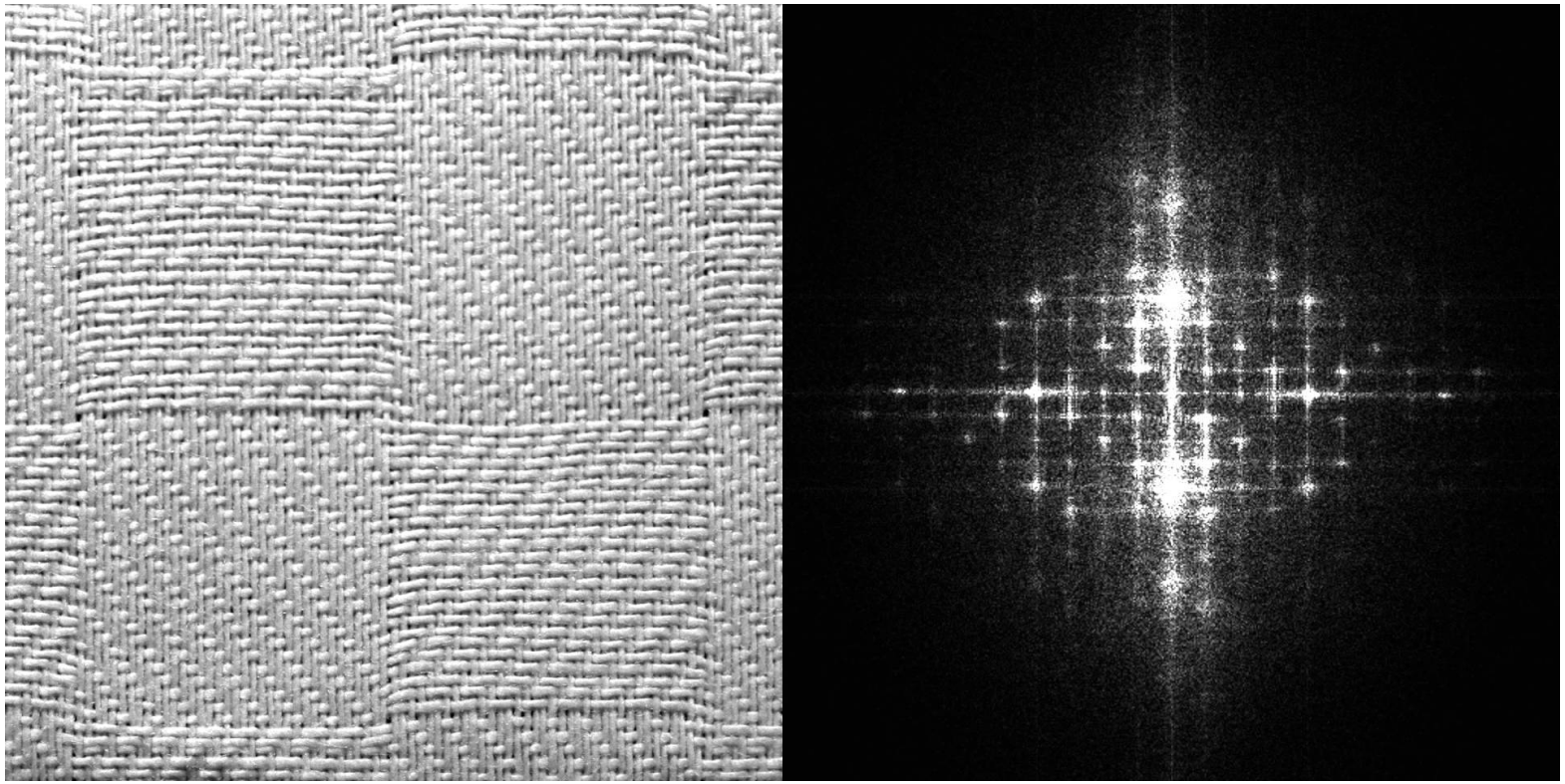
The wavelength is $1/\sqrt{u^2+v^2}$. The direction is u/v .

Those who are experts in algorithm analysis will spot the problem with the order of growth of the 2D DFT… (what's the $O(N)$ of the 2D FFT?)

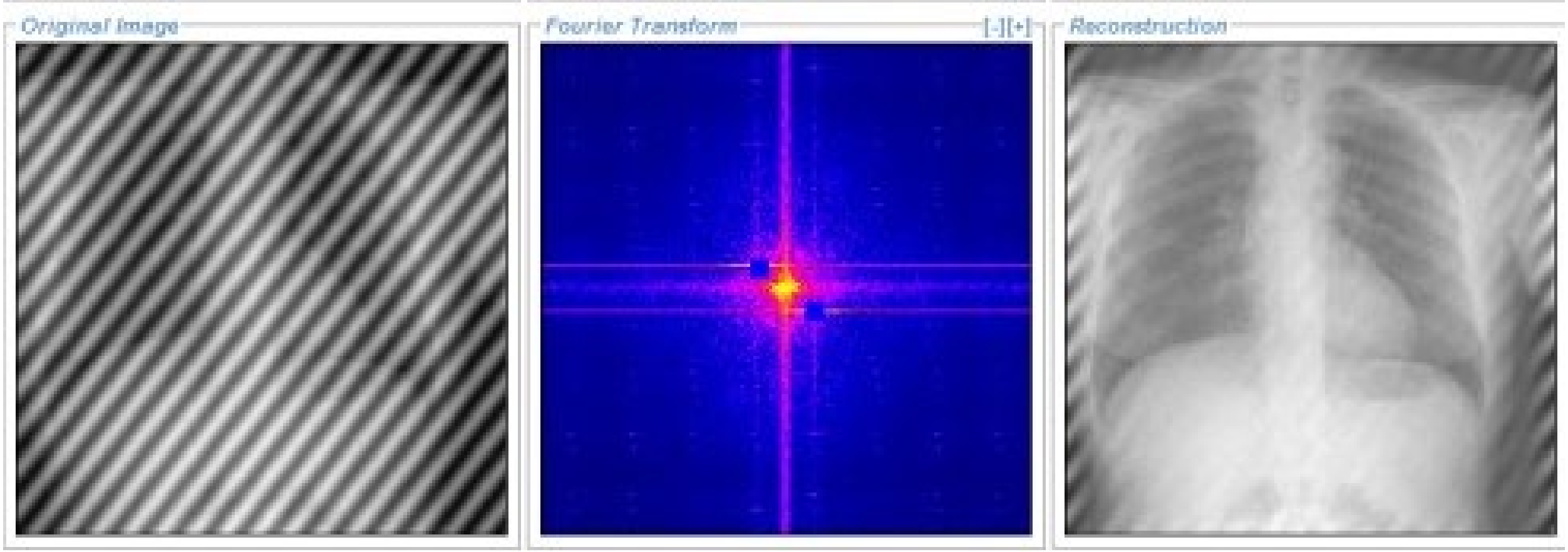
- Cooley and Tukey (1965) developed the Fast Fourier Transform algorithm – the FFT
  - Actually, Gauss knew about this technique in 1805!
  - It's been extended to two dimensions
- A one dimensional FFT is $O(N) = N \log N$
- The two dimensional FFT is $O(N, M) = NM \log NM$
- This algorithmic innovation has made audio and video processing feasible
  - Perhaps the single most important discovery in signal processing (though others would disagree with me)

The two-dimensional Fourier transform used for images reveals spatial frequencies at various orientations in the X-Y plane

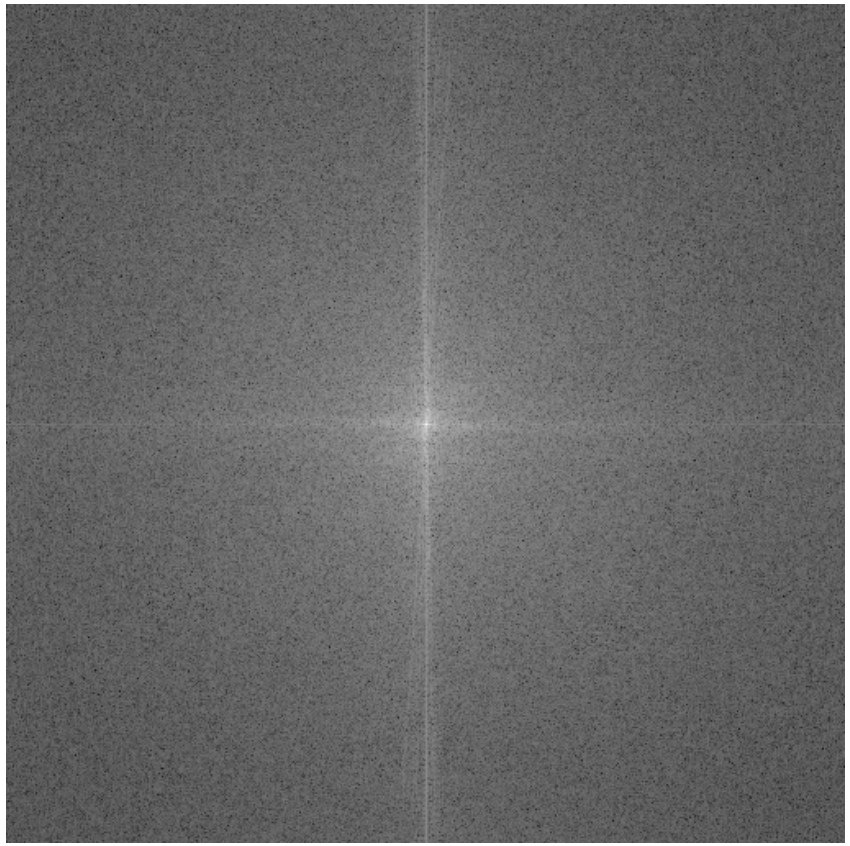
$$F(u,v) = \iint_{-\infty}^{\infty} f(x,y)\, e^{-2\pi j(ux+vy)}\, dx\, dy$$



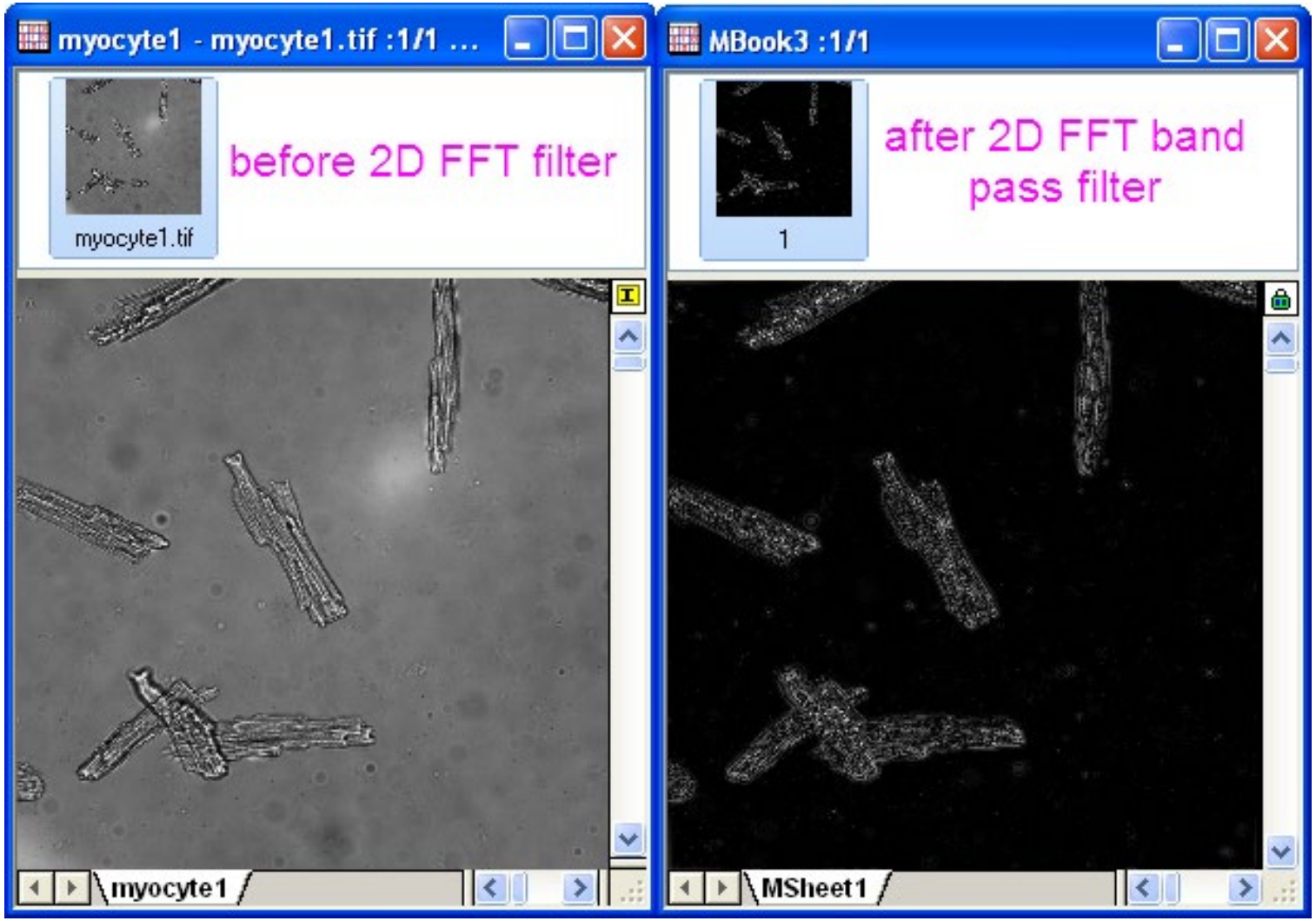
This is the magnitude of $F(u,v)$ - the colors indicate brighter areas (higher magnitude)

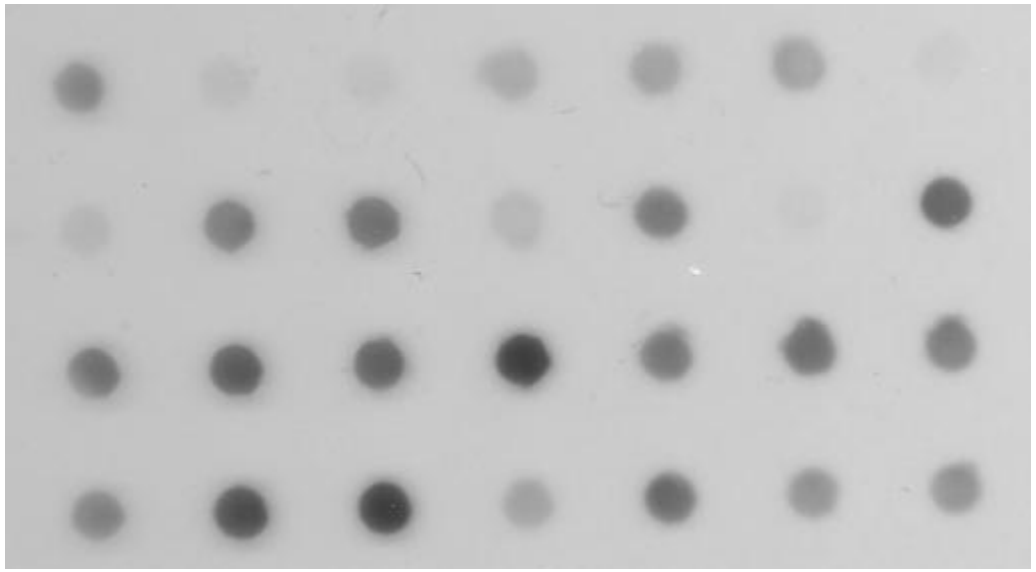


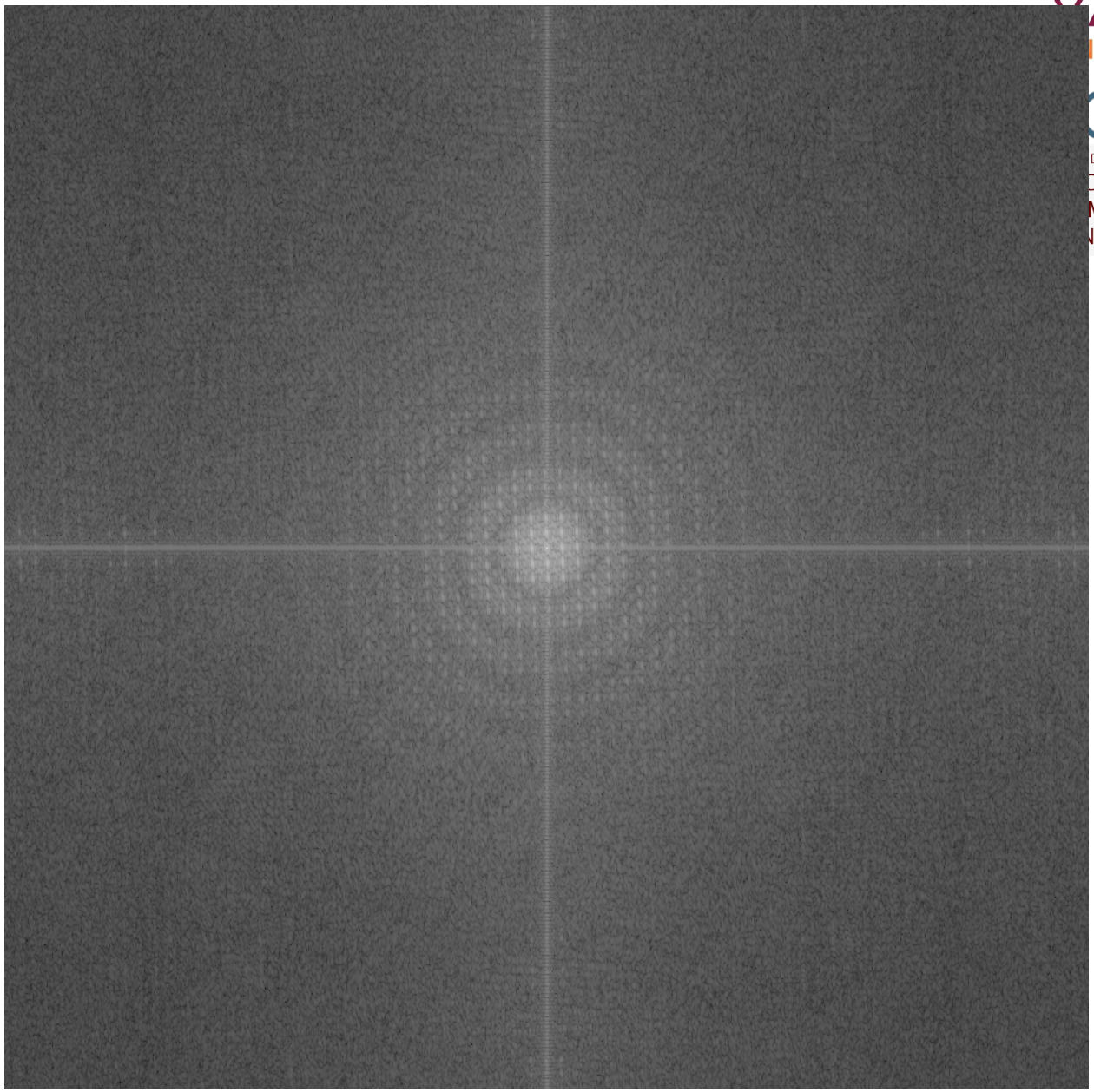
Most tools show the magnitude of the Fourier transform – there is a phase (or *direction*) as well…

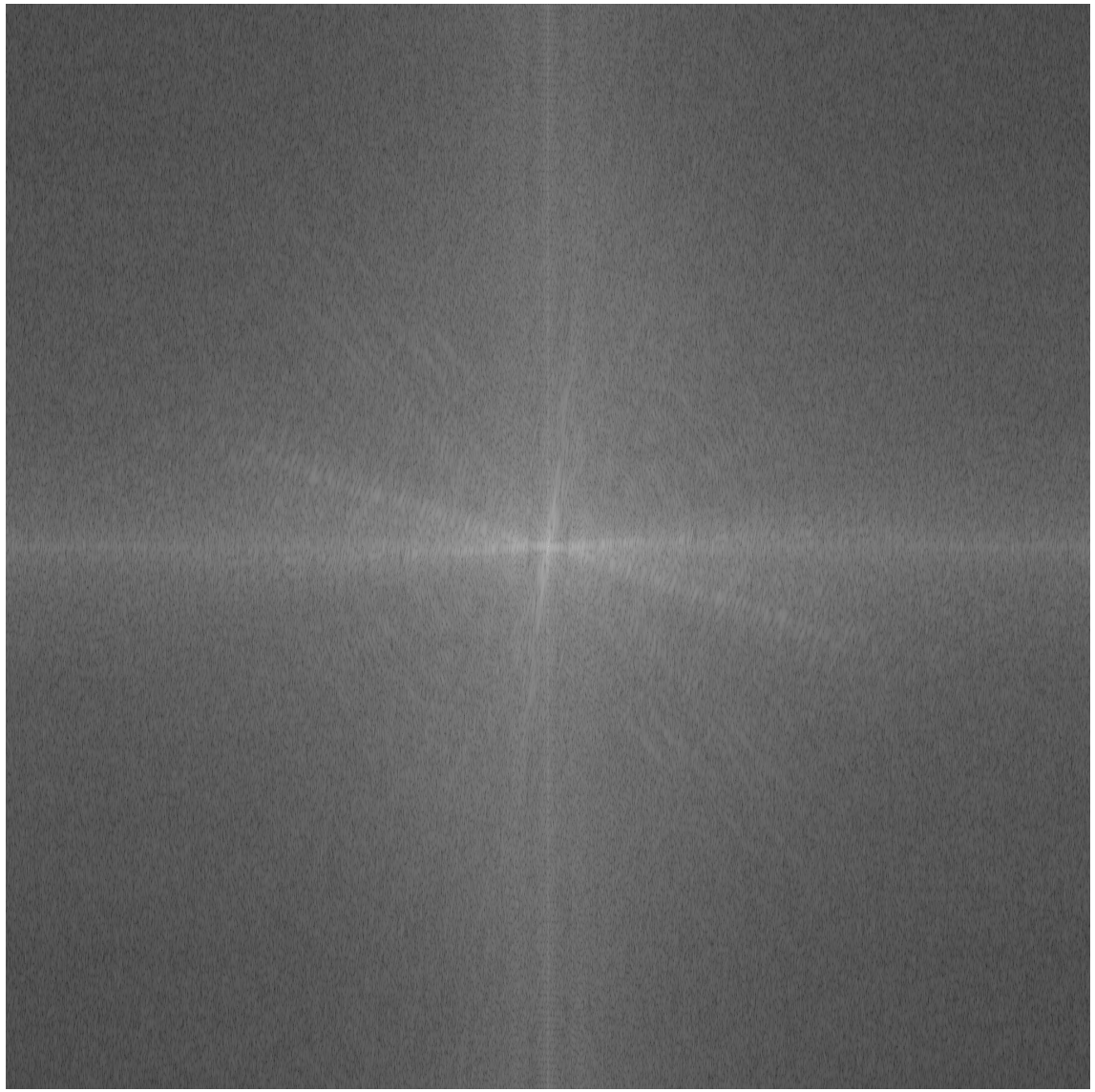


The 2D Fourier Transform has uses in compression, noise removal, filtering, image analysis and image characterization









I use the numpy function for two-dimensional Fourier transforms – numpy.fft.fft2

fft.fft2(***a, s=None, axes=(-2, -1), norm=None***)

Compute the 2-dimensional discrete Fourier Transform.

This function computes the *n*-dimensional discrete Fourier Transform over any axes in an *M*-dimensional array by means of the Fast Fourier Transform (FFT). By default, the transform is computed over the last two axes of the input array, i.e., a 2-dimensional FFT.

Parameters

aarray_like

Input array, can be complex

ssequence of ints, optional

Shape (length of each transformed axis) of the output (s[0] refers to axis 0, s[1] to axis 1, etc.). This corresponds to n for fft(x, n). Along each axis, if the given shape is smaller than that of the input, the input is cropped. If it is larger, the input is padded with zeros. if *s* is not given, the shape of the input along the axes specified by *axes* is used.

axes*sequence of ints, optional*

Axes over which to compute the FFT. If not given, the last two axes are used. A repeated index in *axes* means the transform over that axis is performed multiple times. A one-element sequence means that a one-dimensional FFT is performed.

norm*{"backward", "ortho", "forward"}, optional*

New in version 1.10.0.

Normalization mode (see **numpy.fft**). Default is "backward". Indicates which direction of the forward/backward pair of transforms is scaled and with what normalization factor.

New in version 1.20.0: The "backward", "forward" values were added.

To create the usual "centered" FFT display, use numpy.fft.fftshift() – note, you must use ifftshift() prior to applying the inverse transform

fft.fftshift(*x*, *axes=None*)[source]

Shift the zero-frequency component to the center of the spectrum.

This function swaps half-spaces for all axes listed (defaults to all). Note that y[0] is the Nyquist component only if len(x) is even.

Parameters

x *array_like*

Input array.

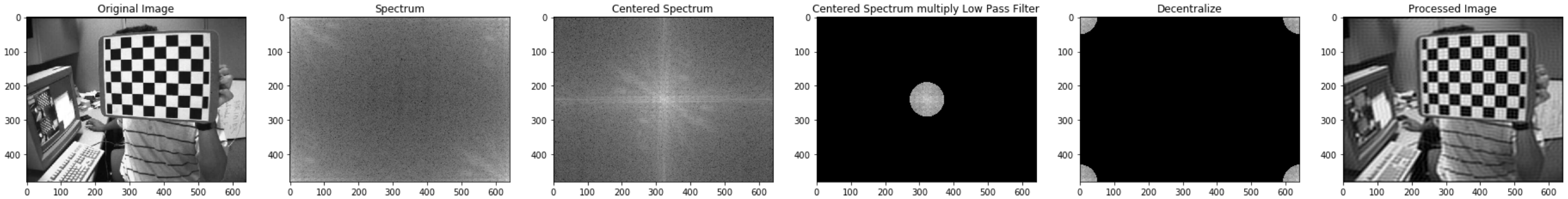
axes *int or shape tuple, optional*

Axes over which to shift. Default is None, which shifts all axes.

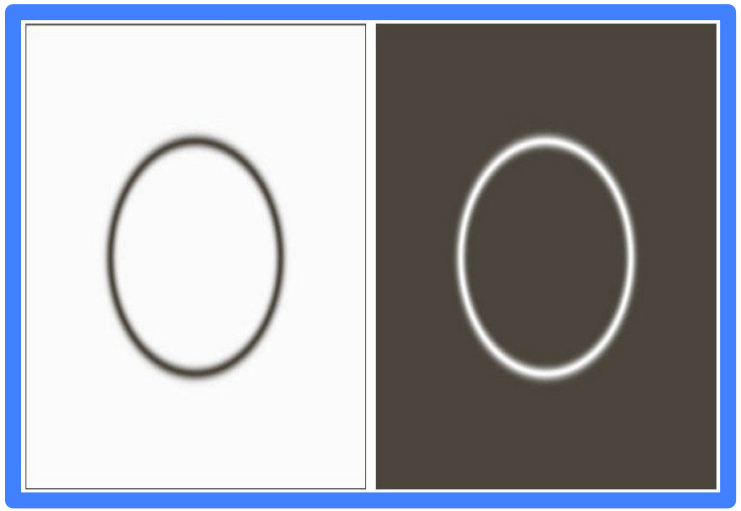
Returns

y *ndarray*

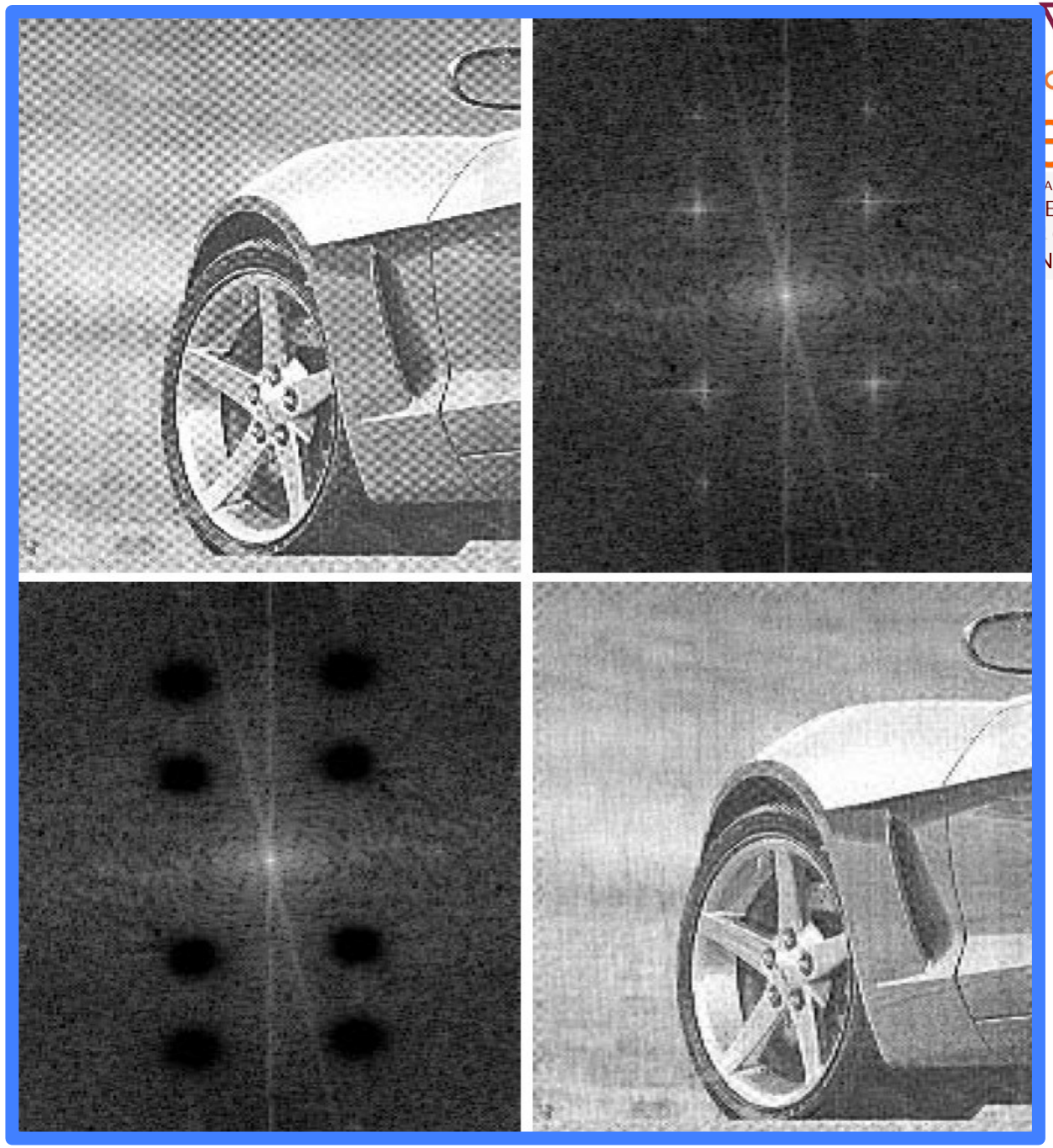
The shifted array.



Selective filtering using Bandreject and Bandpass Filters

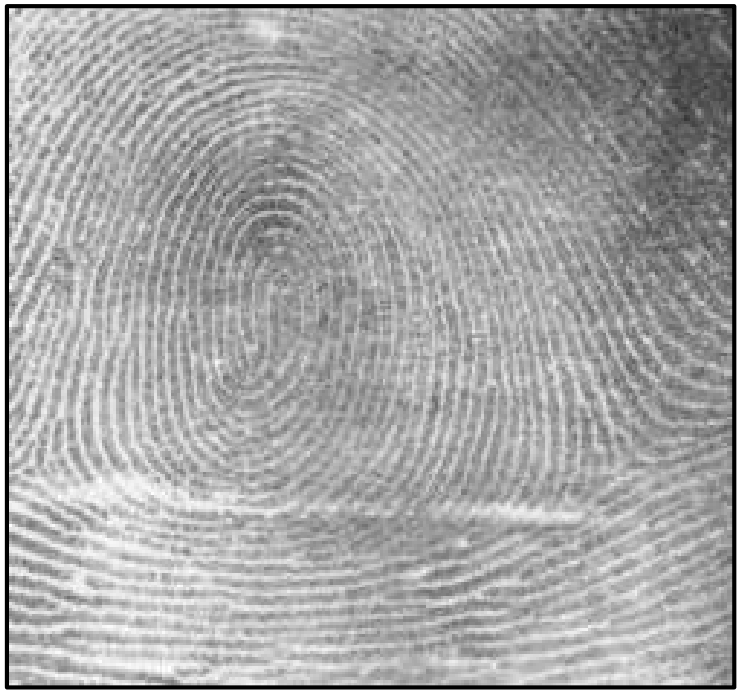


Notch filters to be applied in the Fourier domain

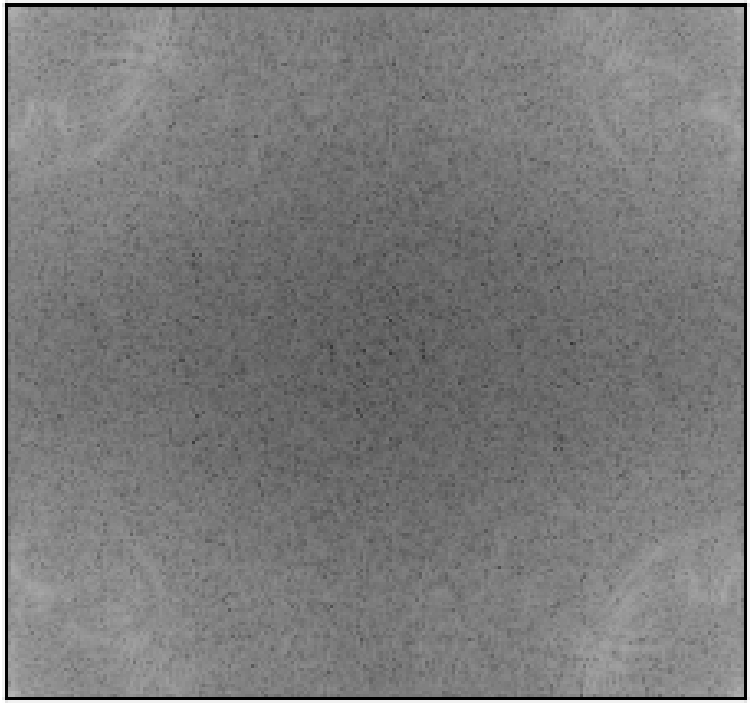


Consider an image with a large amount of certain spatial frequencies – a fingerprint

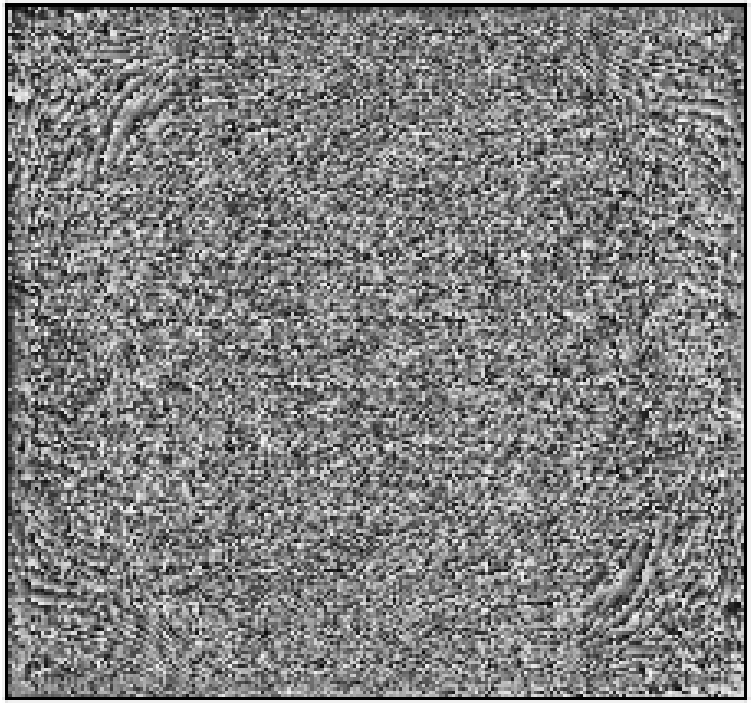
The 2D Fourier transform magnitude and phase shows the content at certain frequencies (in these plots, the Fourier domain origin is in the corner)



FFT magnitude

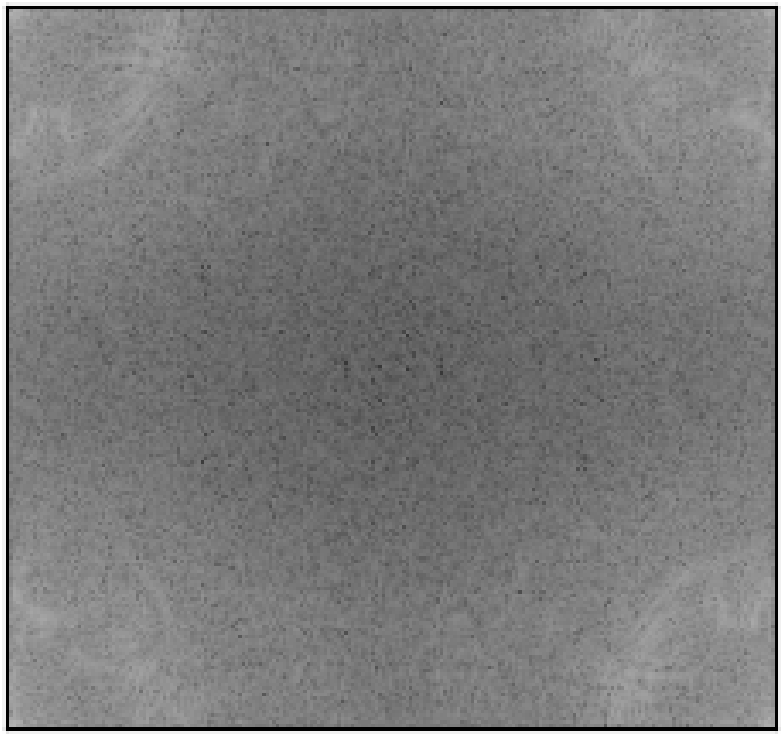


FFT phase

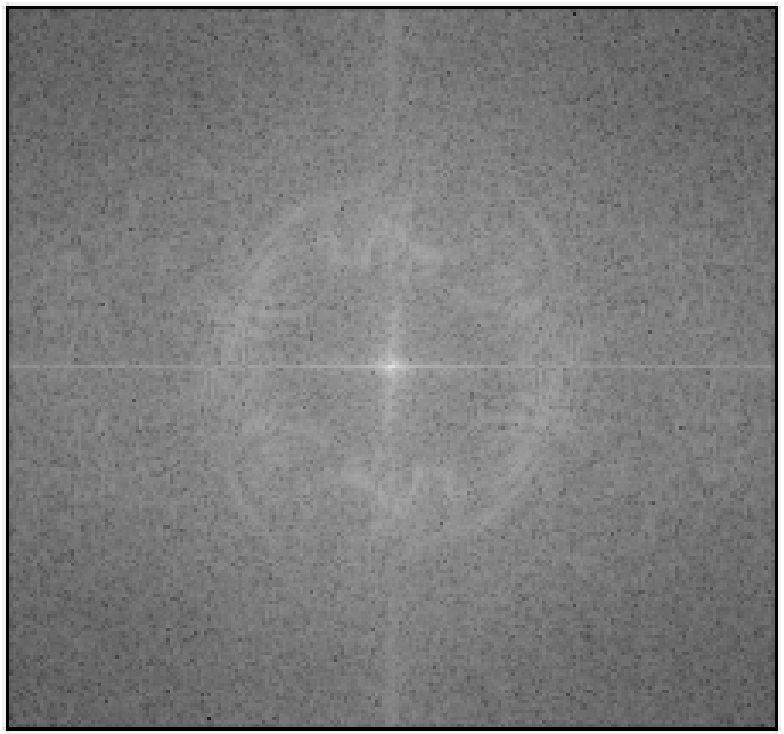


We apply fft2shift to place the Fourier origin in the center – this is the most common way to see the Fourier information

FFT magnitude

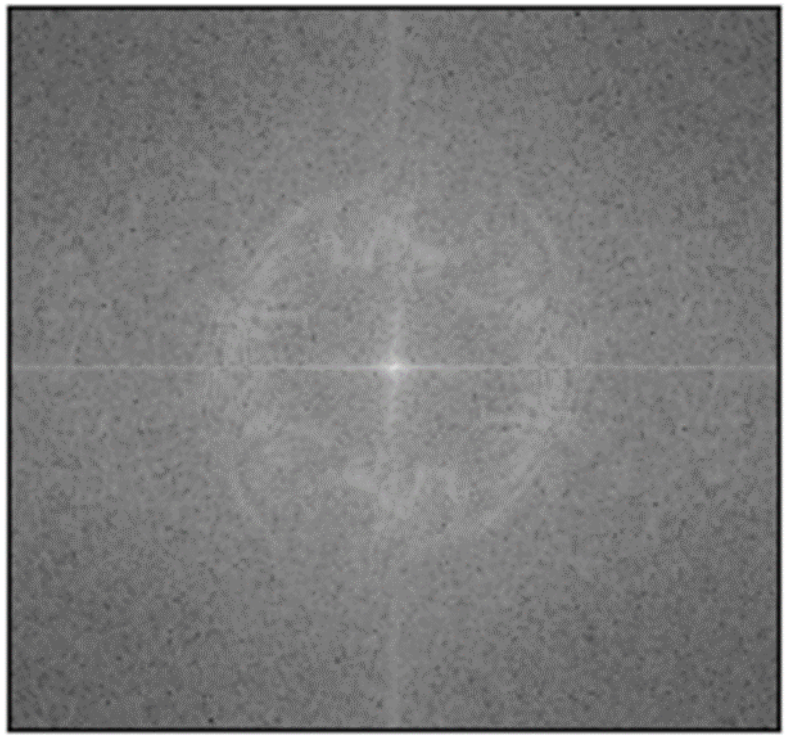


FFT shifted

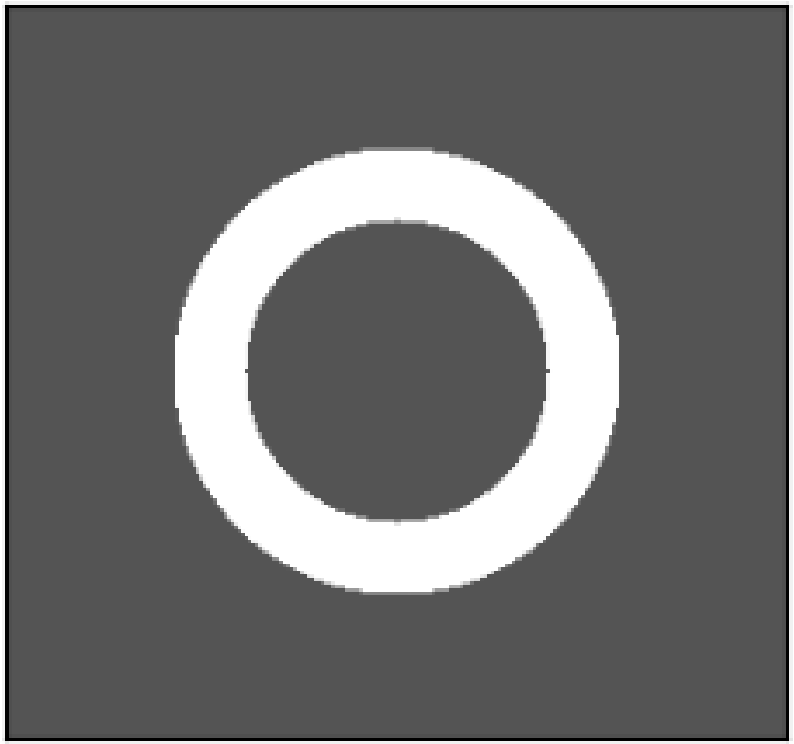


We emphasize the dominant frequencies using a mask – the mask values are 1.0 and 0.333

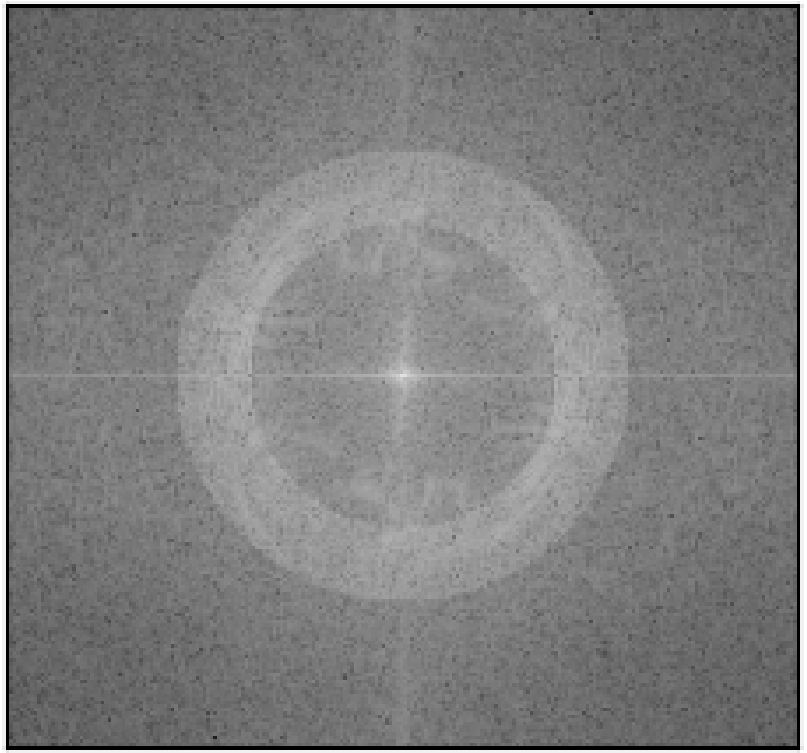
FFT shifted



mask

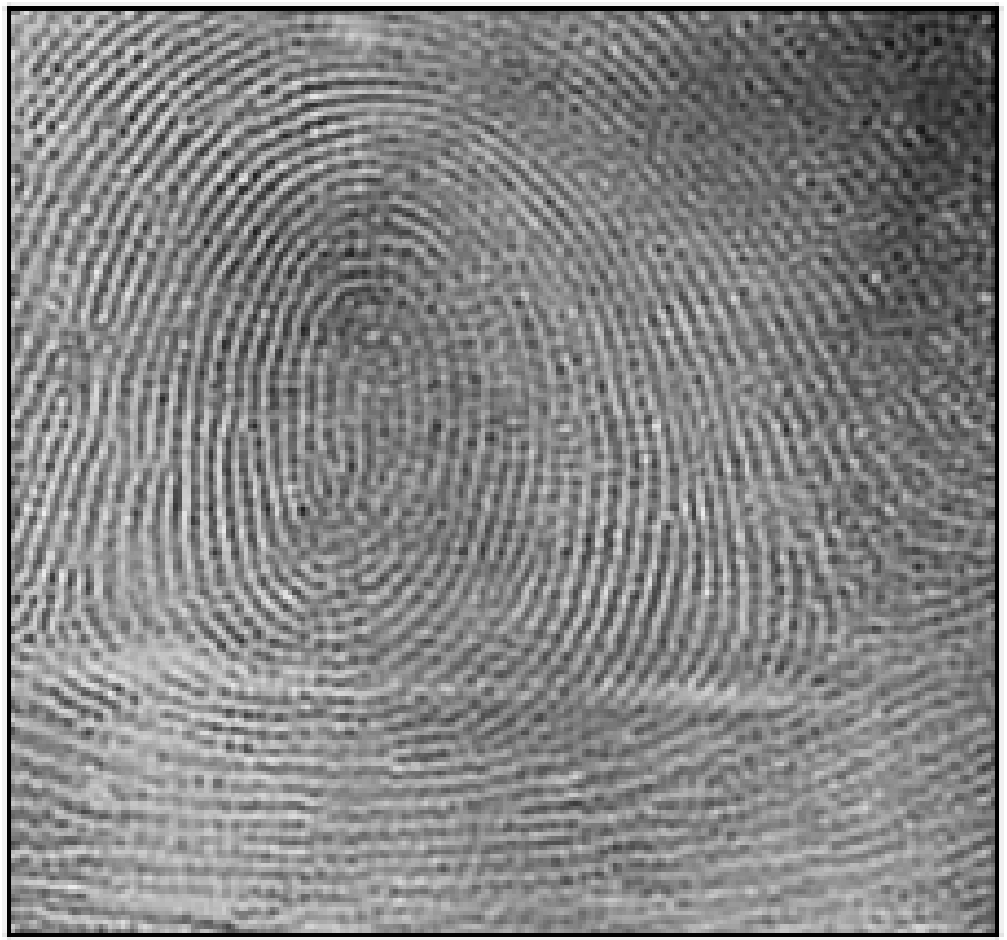


FFT mag masked



original

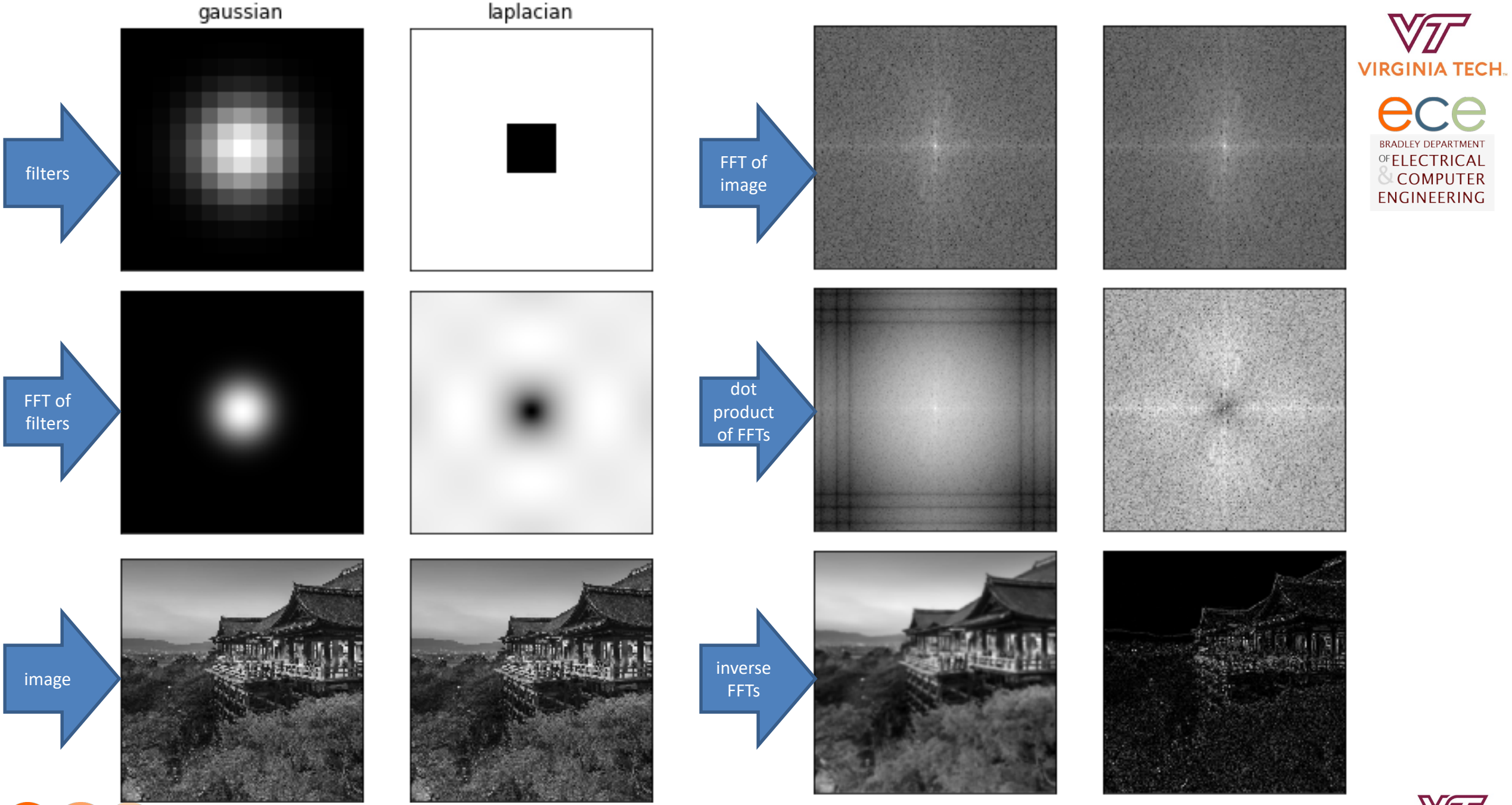
result



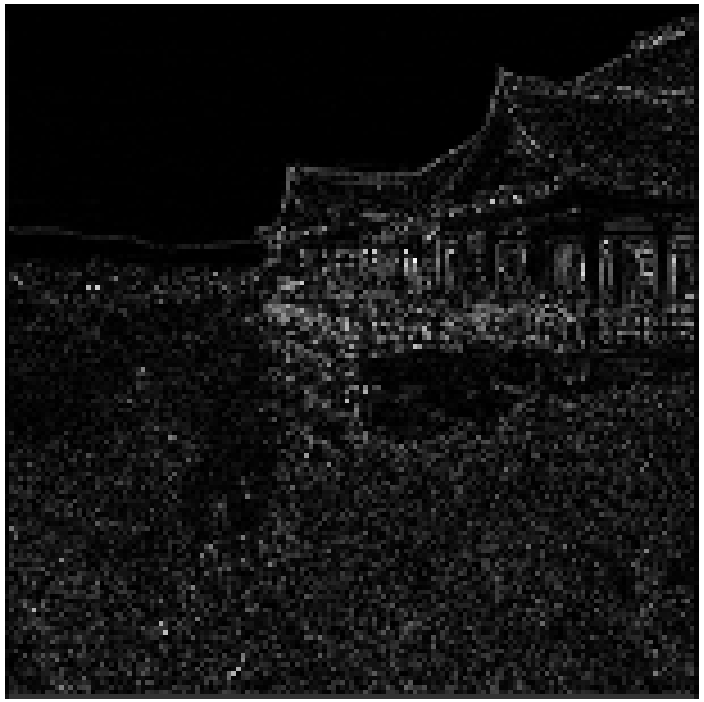
```
def fftfingerprint(img):
    plt.show()
    fftimg = np.fft.fft2(img)
    fftmag = np.log(np.abs(fftimg))
    fftphase = np.angle(fftimg)
    shifted = np.fft.fftshift(fftimg)
    shiftedmag = np.log(np.abs(shifted))

   # make a mask - there are better ways I'm sure
    mask = 0.33*np.ones_like(fftmag)
    center = np.asarray([np.int(fftmag.shape[0]/2), np.int(fftmag.shape[1]/2)])
    radii = np.asarray([np.int(35/180*fftmag.shape[1]),
                        np.int(55/180*fftmag.shape[0]) ])
    for row in range(fftmag.shape[0]):
        for col in range(fftmag.shape[1]):
            dist = np.linalg.norm(np.asarray([row, col]) - center)
            if ( (radii[0] < dist) and (dist < radii[1]) ):
                mask[row, col] = 1.0

    maskedfft = np.multiply(shifted, mask)

    res = np.fft.ifft2(np.fft.ifftshift(maskedfft))
```

Let's experiment with performing kernel filtering in the Fourier domain – Gaussian smoothing and Laplacian 2nd gradient computation

```
def showImgPair(img, names):
    for i in range(2):
        plt.subplot(1,2,i+1),plt.imshow(img[i],cmap = 'gray')
        plt.title(names[i]), plt.xticks([]), plt.yticks([])
    plt.show()

################################################################################

M=513        # creating a gaussian filter
x = cv2.getGaussianKernel(15,2)
gauss = x*x.T
laplacian=np.array([[1, 1, 1, 1, 1],    # laplacian
                    [1, 1, 1, 1, 1],
                    [1, 1, -24, 1, 1],
                    [1, 1, 1, 1, 1],
                    [1, 1, 1, 1, 1]])
padWid = np.int(M/2 - gauss.shape[0]/2)
bigGaussian = np.pad(gauss, ((padWid, padWid), (padWid, padWid)))
padWid = np.int(M/2 - laplacian.shape[0]/2)
bigLaplacian = np.pad(laplacian, ((padWid, padWid), (padWid, padWid)))
filters = [bigGaussian, bigLaplacian]
filter_name = ['gaussian','laplacian']
fft_filters = [np.fft.fft2(x) for x in filters]
fft_shift = [np.fft.fftshift(y) for y in fft_filters]
mag_spectrum = [np.log(np.abs(z)+1) for z in fft_shift]
plt.show()
img = cv2.imread("C:\\Data\\kyotosquare.png", cv2.IMREAD_GRAYSCALE)
rawSpec = np.fft.fft2(img)
spec = np.fft.fftshift(np.fft.fft2(img))
resSpec = [np.multiply(spec, fft_shift[0]), np.multiply(spec, fft_shift[1])]
dispRes = [np.fft.ifft2(resSpec[0]), np.fft.ifft2(resSpec[1])]
res = [np.fft.ifftshift(np.fft.ifft2(resSpec[0])), np.fft.ifftshift(np.fft.ifft2(resSpec[1]))]

showImgPair([gauss, laplacian], filter_name)
showImgPair(mag_spectrum, ['',''])
showImgPair([img, img], ['',''])
showImgPair(np.log(np.abs([spec, spec])), ['',''])
showImgPair(np.log(np.abs(resSpec)), ['',''])
showImgPair(np.abs(dispRes), ['',''])
showImgPair(np.abs(res), ['',''])
```


gaussian
laplacian
filters
FFT of image
FFT of filters
dot product of FFTs
image
inverse FFTs

The results are indeed the Gaussian smoothed and Laplacian filtered versions of the original image – multiplication in the Fourier domain accomplishes convolution in the spatial domain



Today's Objectives

- The Fourier Transform
  - The concept
  - The math
- The 2D Fourier Transform
- Discrete Fourier Transforms
- Basis Functions
- 2D FFT of images
- numpy fft2()
  - shifting results
- example of fingerprint enhancement
- example of smoothing in the Fourier domain