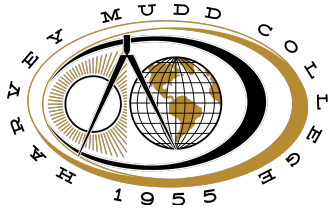
HMC Biophotonics Lab Summer 2021 Research

Fourier Ptychography

Kevin Kim



July 31, 2021

Contents

1 Concepts

1.1 Diffraction and Resolution

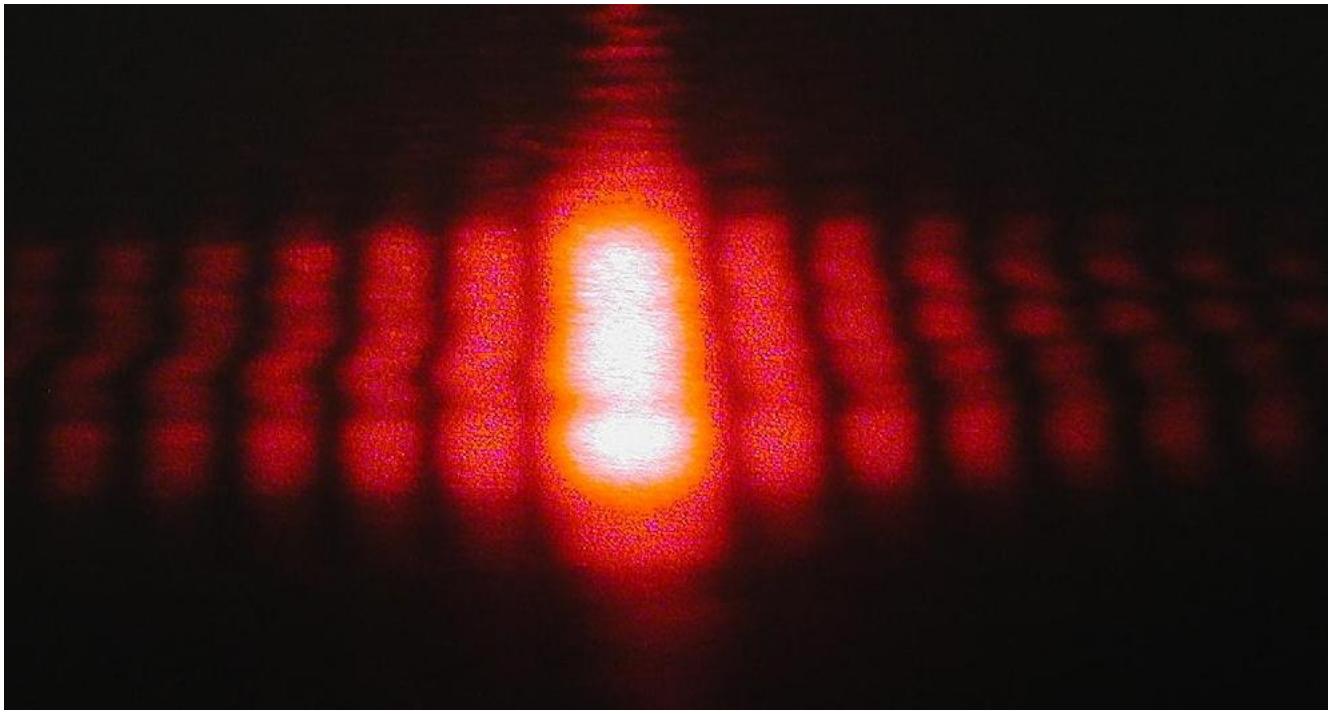


Figure 1: Diffraction pattern from light diffracted from a rectangular aperture[1]

Diffraction describes the phenomena when light, or a system of waves, bends around an object. This is due to Huygen's principle, which states that every point on a wavefront is a source of wavelets– when wavefronts hit small openings, collections of wavelets create sources of new waves, allowing the initial wave to "bend" around objects. Essentially, this allows us to treat a plane wave as a continuous array of point sources of waves. There are two main regimes of diffractions: Fresnel Diffraction and Fraunhofer Diffraction. As the former corresponds to the near field and the latter corresponds to far field (or near the focal plane), Fraunhofer Diffraction will be more relevant to our discussion. Consider a plane wave that hits a small opening like shown in the diagram below.

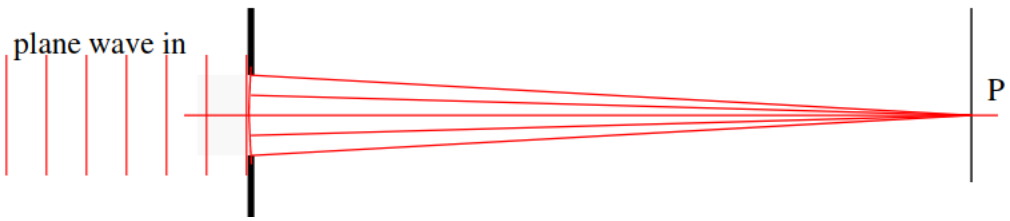


Figure 2: Diffracted light from a rectangular aperture all in phase[2]

Considering a parallel bundle of light, we notice that the rays are all in phase, meaning that all the peaks of the light waves are aligned. As such, the center most region in the diffraction pattern gives the highest intensity because the parallel light bundles start together and converge together with no phase difference, leading to constructive interference. However, when we consider parallel light bundles with different exit angles, notice that there

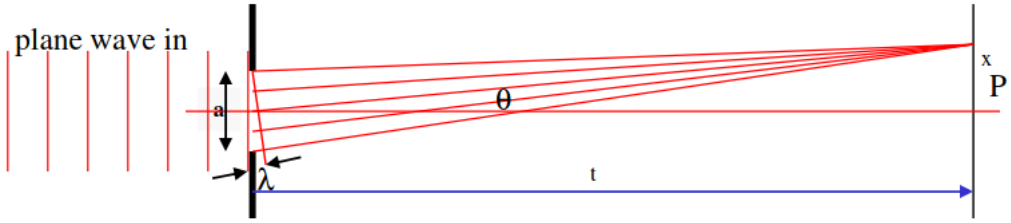


Figure 3: off axis light diffracted through an aperture [2]

is a path difference in the exit bundles, meaning that not all the light is in phase anymore. The series of light that are one wavelength out of phase creates "dark fringes" while light waves that are more in phase create light fringes, which compose a diffraction pattern. Path differences between the extreme edge rays within the parallel bundles of light can be denoted as (refer to to figure 4)

$$\Delta path = a \sin(\theta)$$

while phase differences between the extreme edge rays (δ) follow as

$$\delta = \Delta path \frac{2\pi}{\lambda}$$

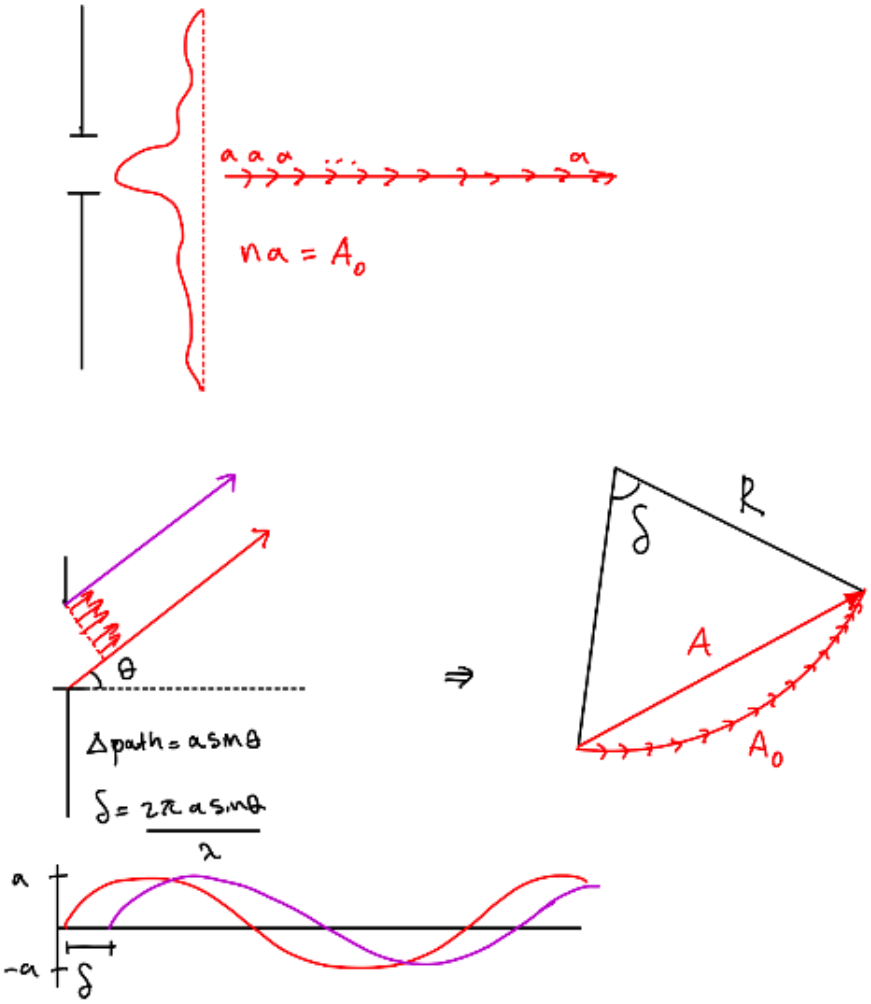


Figure 4: phasor addition of diffracted off-axis light

The intensity profile of the diffraction pattern can be examined using phasors. Let A_0 represent the amplitude of all the in-phase waves that constructively interfere. Introducing a phase delay δ, all n waves in between the extreme edge rays will have some amplitude $\frac{A_0}{n}$ and will differ from each other by phase shift $\frac{\delta}{n}$. Using a phasor diagram, adding the individual phasors representing the bundles of light that interfere is equivalent to calculating the chord of a circle with radius $\delta R = A_0$. The intensity profile of the interfering light is therefore

$$I = I_0 \frac{\sin^2(\frac{\delta}{2})}{\left[\frac{\delta}{2}\right]^2}$$

where I_0 is the intensity of the in phase light that interferes at the center of the aperture. Since intensity of the wave is the amplitude squared(A^2),

$$A = A_0 \operatorname{sinc}(\frac{\delta}{2})$$

and the phase differences between extreme edge rays δ is characterized as

$$\delta = \Delta path \frac{2\pi}{\lambda} = a \sin(\theta) \frac{2\pi}{\lambda}$$

where

$$\frac{y}{D} = \tan(\theta) \approx \sin(\theta) \approx \theta$$

$$A = A_0 \operatorname{sinc}(\frac{ay\pi}{D\lambda})$$

note that as the size of the aperture a increases, the sinc function projected onto the y axis gets thinner. As the size of the aperture a decreases however, the sinc function gets wider. If two objects are imaged through the aperture, the light scattered off the images will produce corresponding sinc functions on the sensor– the more these sinc functions overlap, the less distinguishable the images will be. Note that because we are working with two dimensions, we will see a 2d sinc function

Figure 5: An example of airy disks resulting from diffracted light from a circular aperture. If the centers of two airy disks overlap too much, then the disks are not resolved and thus the optical system fails to distinguish between the details [3]

As the aperture size increases, the width of the airy disks will get smaller and thus the ability to distinguish between airy disks that are closer to each-other will get better.

1.2 Numerical Aperture

The resolution of an optical system refers to its ability to distinguish objects in high detail. In the previous section we established that the size of the aperture is directly related its resolving ability. However, the size of the aperture is also related to the cone of acceptance angles of light emitting from the sample plane. The larger the aperture, the larger the cone of acceptance angles. It is actually the cone of acceptance angles that determine the ability the optical system has to resolve images. The larger the cone of acceptance angles, the greater the ability to resolve light with higher spatial frequencies. Mathematically, however, the NA is defined as

$$NA = n \sin(\theta)$$

in which n is the refractive index of the medium between the aperture and the source of light and θ is defined as the maximum angle of light relative to the optical axis that still reaches the aperture.

Similar to how a circular aperture diffracts light into an airy disk, or a rectangular aperture diffracts light into a "sinc" function, diffracted light off of a specimen also produce unique patterns that contain higher and lower order diffraction intensities. It is important to note that higher diffraction orders propagate at larger angles relative to the 0th order diffraction order. As the NA only accepts up to a certain angle of light, higher diffraction orders may not be imaged by the system, and thus loses out on a certain degree of information.

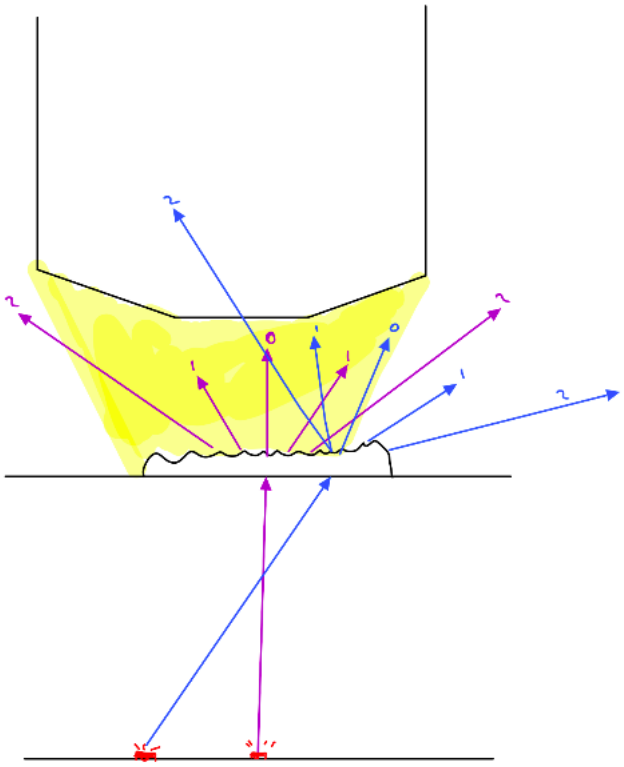


Figure 6: An example of the diffraction orders emitting from the sample. The NA, due to its limited range of acceptance angles, can only collect some of these diffraction orders, limiting the ability to collect information about the source. Illuminating the sample at different angles, however, allows the NA to collect diffraction orders that may have otherwise been neglected, which is one of the core premises of FPM.

1.3 Fourier Transforms and Series

Fourier Transforms are mathematical transforms that can convert mathematical functions with time or space domains into corresponding mathematical functions that have temporal or spatial frequency domains. The explicit mathematical form of the Fourier Transform is

given below:

$$X(f) = \int_{-\infty}^{+\infty} x(t)e^{-j2\pi ft}dt$$

The intuition behind the Fourier Transform comes from the concept of Fourier Series. As $\cos(\frac{n\pi x}{p})$ and $\sin(\frac{n\pi x}{p})$ are orthogonal functions because their inner products are 0, we can project any function into a basis of $\cos(\frac{n\pi x}{p})$ and $\sin(\frac{n\pi x}{p})$ functions with differing amplitudes (called fourier coefficients) and frequencies. The Fourier Series of a function can is represented as a sum of complex exponentials below, in which the Fourier coefficients are c_n and the period T is $\frac{2\pi}{\omega_0}$.

Notice as $T \to \infty$, $\omega_0 \to 0$

$$x(t) = \sum_{n=-\infty}^{+\infty} c_n e^{jn\omega_0 t}$$

where fourier coefficients c_n are defined as:

$$c_n = \frac{1}{T}\int_{n=-\infty}^{+\infty} x(t)e^{-jn\omega_0 t}dt$$

multiplying both sides by T, we obtain

$$Tc_n = \int_{n=-\infty}^{+\infty} x(t)e^{-jn\omega_0 t}dt$$

as Tc_n represent the amplitudes of the component sinusoids, we can now write Tc_n as a function of frequency by letting $\omega = n\omega_0$ as $n\omega_0$ can take on any value (because n ranges from $\pm\infty$)

$$X(\omega) = \int_{n=-\infty}^{+\infty} x(t)e^{-j\omega t}dt$$

which is defined as the forward fourier transform. To derive the inverse fourier transform, we again start with the fourier series on an infinite interval:

$$x(t) = \sum_{n=-\infty}^{+\infty} c_n e^{jn\omega_0 t}$$

multiplying and dividing the right hand side by T, we obtain:

$$x(t) = \sum_{n=-\infty}^{+\infty} Tc_n e^{jn\omega_0 t}\frac{1}{T}$$

after substituting $T = \frac{2\pi}{\omega_0}$,

$$x(t) = \sum_{n=-\infty}^{+\infty} Tc_n e^{jn\omega_0 t}\frac{\omega_0}{2\pi}$$

again, letting

$$\omega = n\omega_0$$

and as $T \to \infty \omega_0 \to d\omega$

$$x(t) = \frac{1}{2\pi} \sum_{n=-\infty}^{+\infty} X(\omega) e^{j\omega} d\omega$$

which yields the inverse fourier transform.
Fourier analysis can most easily be understood through the domain of sound– consider a chord that is the summation of 3 different notes. Each note corresponds to a single sinusoid with a frequency that corresponds to its pitch. The superposition of those sound waves composes the chord. Taking the fourier transform of that chord would give us three distinct delta functions, each corresponding to the distinct frequency of the constituent notes and with an amplitude corresponding to the volume of the notes.

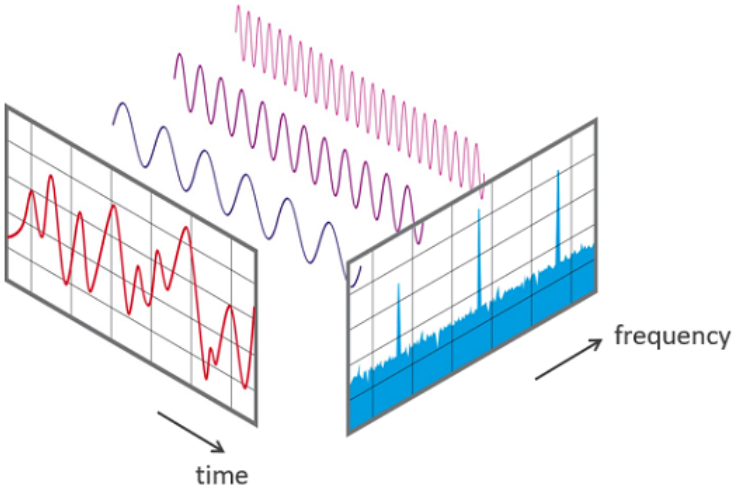


Figure 7: A visualization of a signal on the time domain, its constituent sinusoids, and the frequency of those constituent sinusoids relayed onto the fourier transform [4]

1.4 Spatial Frequencies and the 2D Fourier Transform

Fourier Transforms are not limited to shifting between the time and frequency domains. Signals that have space-dependent amplitudes can also be fourier transformed to analyze their spatial frequency components. As such, any signal or function on the space domain can be fourier transformed. This includes images, which can be thought of as 2 dimensional functions with a spatial domain that corresponds to intensities. We typically analyze the spatial frequencies of images by looking for oscillating components in a particular direction.

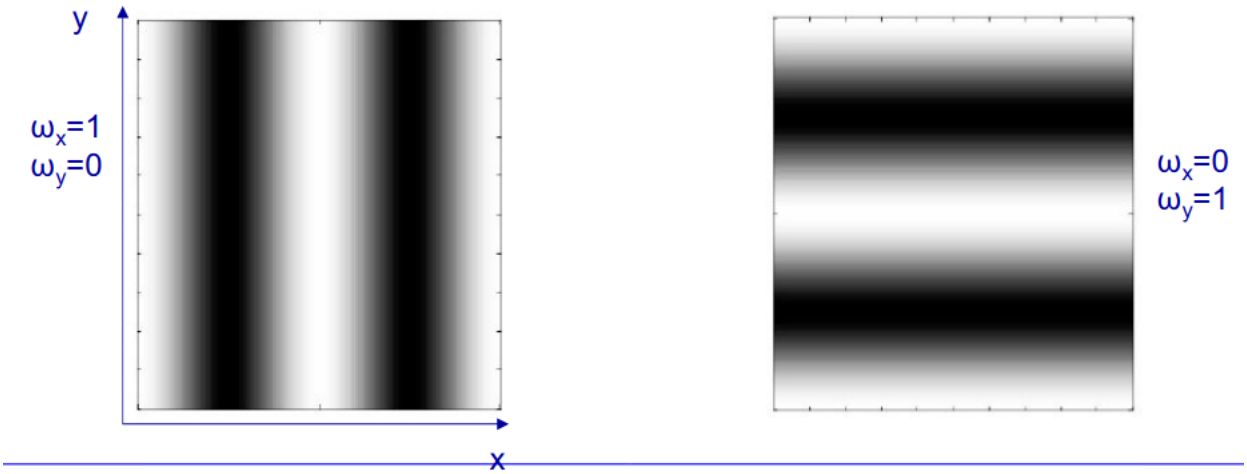


Figure 8: The image on the left is said to have a nonzero horizontal spatial frequency, as the fringes oscillate between black and white in the horizontal direction while the image on the right has a nonzero vertical spatial frequency, as fringes oscillate intensities in the vertical direction [5]

Taking the 2 dimensional Fourier transform of an image gives insight towards the spatial frequency components of the images. Similar to taking the 1-D fourier transform, in fourier space (another term for frequency domain) intensities represent fourier coefficients and positions represent spatial frequencies. As such, intensities gathered close to the origin represent the contribution of low spatial frequencies and intensities further away represent the contributions of higher spatial frequencies.

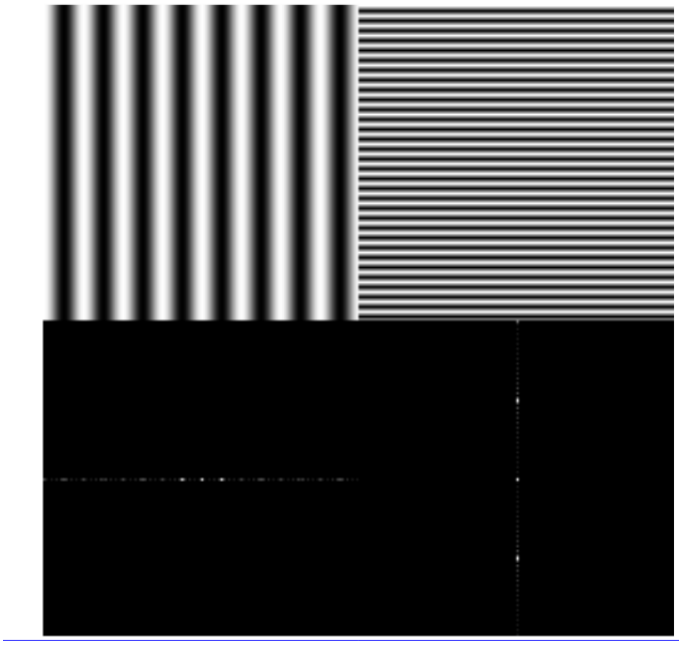


Figure 9: The fourier transforms corresponding to the horizontal and vertical fringes are shown below. As the horizontal fringes exhibit a horizontal spatial frequency, the fourier transform only shows values on the x axis, while the fourier transform of the vertical fringe pattern shows values on the y axis, denoting vertical frequencies [5]

An important concept is that the Fourier Transform does not lose any information– that is, all the information that encodes a signal or an image is preserved within the fourier transform. Simply inverse fourier transforming will bring back the original artifact. A more rigorous proof of this is given by *Parseval's Theorem* [6], which states that the fourier transform conserves the square of both the function and its transform. This is an extension of the conversation of energy principle, but will not be discussed further in this report.

1.5 Masks and Coherent Transfer Function

Given the fourier transform of a signal, we can manipulate the frequency content of the signal by filtering out specific frequencies. Inverse fourier transforming will give a different sinusoid lacking the filtered high frequency components. A sound that has been put through a "low pass filter" (a filter that blocks low frequencies) will lack high *pitches*, in the spatial sense, blocking out higher spatial frequencies will produce an image with a decreased *resolution*. Therefore, an image that has been put through a "low pass filter" will lack higher spatial frequency content. This is a fundamental concept behind the Numerical Aperture. The Numerical Aperture of an imaging system acts as a low passband for spatial frequencies because it only accepts a certain range of angles of light which correspond with spatial frequencies. The more the angle of light deviates from the optical axis (z-axis), the higher the spatial frequency the light will carry in the x-y directions. This can be seen using simple geometry.

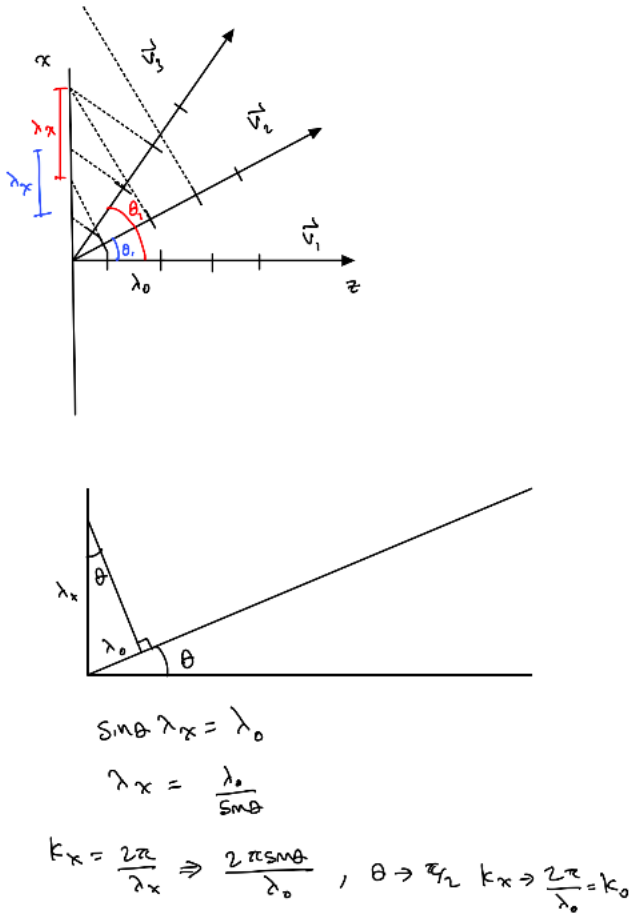


Figure 10: Geometrical depiction of the dependence of spatial frequency content on scattered angles of light

Given the diffracted light off the object is coherent, each wave will exhibit a constant wavelength in the direction of propagation. This is equivalent to saying that the spatial frequency in the direction of propagation is also constant for all waves ($\frac{2\pi}{\lambda_0}$). Consider a spherical bundle of coherent light emitting from an object propagating along the optical axis (z) with horizontal and vertical components in x and y. The light wave at further angles relative to the optical axis will have smaller components of wavelengths in the x and y directions, and thus, larger spatial frequency along the x and y directions.

$$\lambda_x = \frac{\lambda_0}{\sin(\theta)}$$

$$k_x = \frac{2\pi}{\lambda_x} = \frac{2\pi}{\sin(\theta)\lambda_0}$$

$$k_x \propto \frac{1}{\sin(\theta)} : 0 < \theta < \frac{\pi}{2}$$

where θ is the angle between the x axis and z (optical) axis.
The above math can be re-applied to k_y given that θ is the angle between the y and z (optical) axis.

Therefore, the NA's limited range of acceptance angles only accepts light within a certain range of spatial frequencies. In a signals and systems perspective however, the aperture can be described as a transfer function as it modulates the range of input frequencies (filtering out low frequencies). If a *coherent* light field $A_{input}(x, y)$ is imaged through an aperture function $h(x, y)$ the output field $A_{output}(x, y)$ can be described as

$$A_{output}(x, y) = h(x, y) * A_{input}(x, y)$$

to be more exact, $h(x, y)$ represents the *point spread function*, which represents the 3d diffraction pattern of a point source through an aperture.

Moreover, $*$ denotes a convolution.

Mathematically speaking, the convolution of two functions f and g is defined as [7]:

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau)g(t - \tau)d\tau$$

Intuitively, the convolution between two functions can be thought of as "drag and smear" between two functions, describing the summations of the products of the functions across an interval.

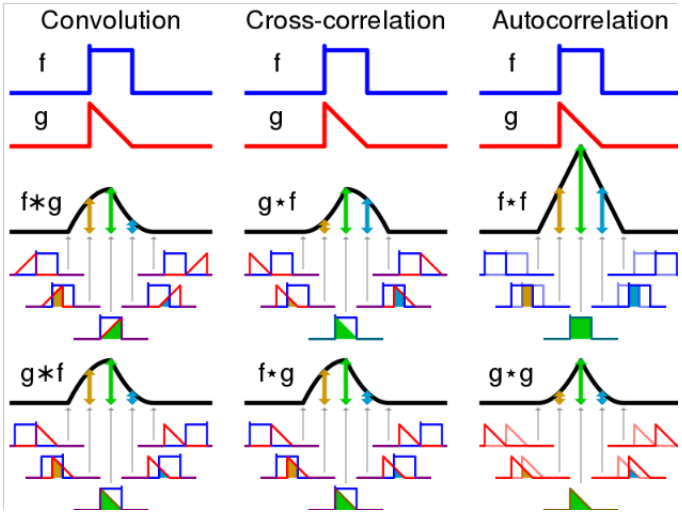


Figure 11: Visual of the convolution, cross correlation, and Autocorrelation [7]

By taking advantage of the Convolution Theorem, which states the convolution of two functions is equivalent to the inverse fourier transform of the product of the fourier transforms of the two functions, we can calculate the output field resulting from the convolution

$$\mathcal{F}\{A_{output}\} = \mathcal{F}\{h(x, y\}\mathcal{F}\{A_{input}(x, y)\}$$

or,

$$G_{output}(k_x, k_y) = H(k_x, k_y)G_{input}(k_x, k_y)$$

where $H(k_x, k_y)$ represents the coherent transfer function, which is a fourier transform of the point spread function [8].

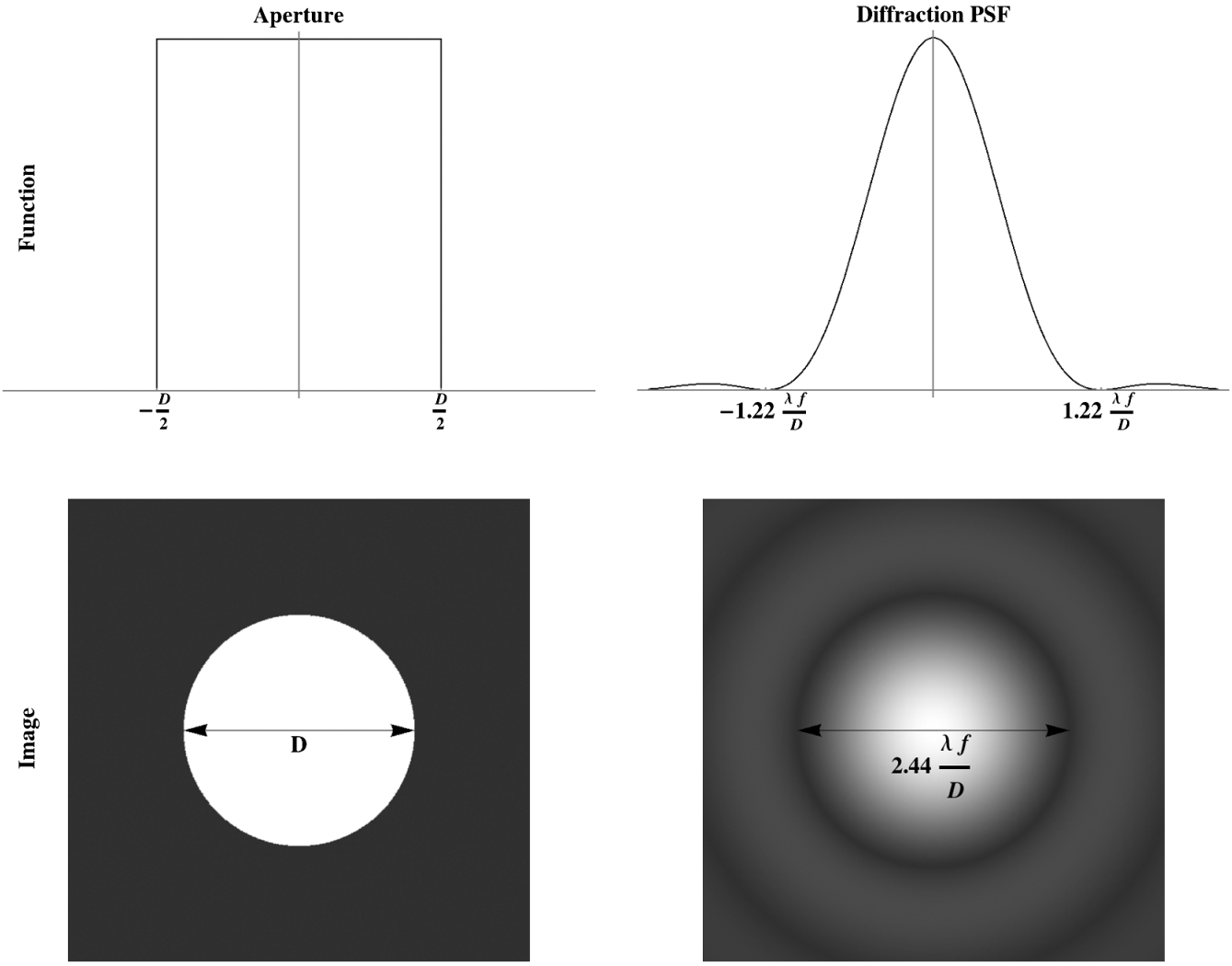


Figure 12: The CTF (image on left) is a fourier transform of the PSF (image on right) [9]

This significant of this concept is that by simply multiplying the fourier transform of an input field by the CTF (which can be represented as a circular mask function) we can arrive at the fourier spectrum of the output field.

1.6 Space Bandwidth Product

Optical engineers use a concept termed the SBP (space bandwidth product) to determine the total number of resolvable pixels in an optical system. Conventional microscope platforms

have SBP's on the order of megapixels. Essentially, the SBP can be thought of as the product between the resolution of an optical system and the field of view the optical system provides– the greater the resolution or FOV, the higher the SBP. However, the SBP for optical systems are fixed, and therefore tradeoffs between resolution and field of view must be made when imaging. A high SBP is useful in, for instance, imaging lots of cells or microorganisms in high detail.

1.7 Increasing the SBP through physical means

Conventional methods of increasing the SBP of a microscope consist of practices that aim to increase either the resolution or FOV of the existing optical system. An example of increasing the FOV through artificial means could include imaging a wider range of the sample through mechanized methods, or by simply using a larger lens (which would actually increase the resolution as well). However, mechanized methods are subject to imprecisions in motion tracking, alignment, and control, which pose a problem within the ultra-precise regime of optics. Moreover, as aberrations within lenses scale with size, simply using a larger lens pose the problem of poorer image detail, unless one seeks out lenses that have been optically refined, which are in turn extremely expensive, posing a monetary issue. On the other hand, increasing the resolution of an optical system through physical means may require the use of higher NA lens or the use of oil immersion microscopy, both of which are more expensive and suboptimal.

2 Fourier Ptychographic Microscopy

As noted in the section above, increasing the SBP through physical means proves to be expensive and subject to imprecision. Fourier Ptychographic Microscopy (FPM), a method of computational imaging, solves this issue by computationally increasing the resolution of an optical system past its physical limits. It does this so by computationally expanding the numerical aperture (NA) of the optical system.

In a traditional optical system, high spatial frequency information is uncaptured due to the low-pass filtering effect of the NA in the fourier domain. In the physical sense, the extreme angles of scattered light that do not fall within the cone of acceptance angles defined by the NA are therefore lost. FPM seeks to construct a higher-resolution image that includes the high spatial frequency content by taking many low resolution images with the high spatial frequency information shifted to fit within the NA bandpass, then stitching the images back to their appropriate positions in fourier space.

The physical setup of an FPM system is composed of a traditional microscope setup with an objective lens and an eyepiece with the exception of the illumination source as an LED array. The LED array, by illuminating the sample from several different angles, can effectively shift the spatial frequency content from the sample that enters the NA passband.

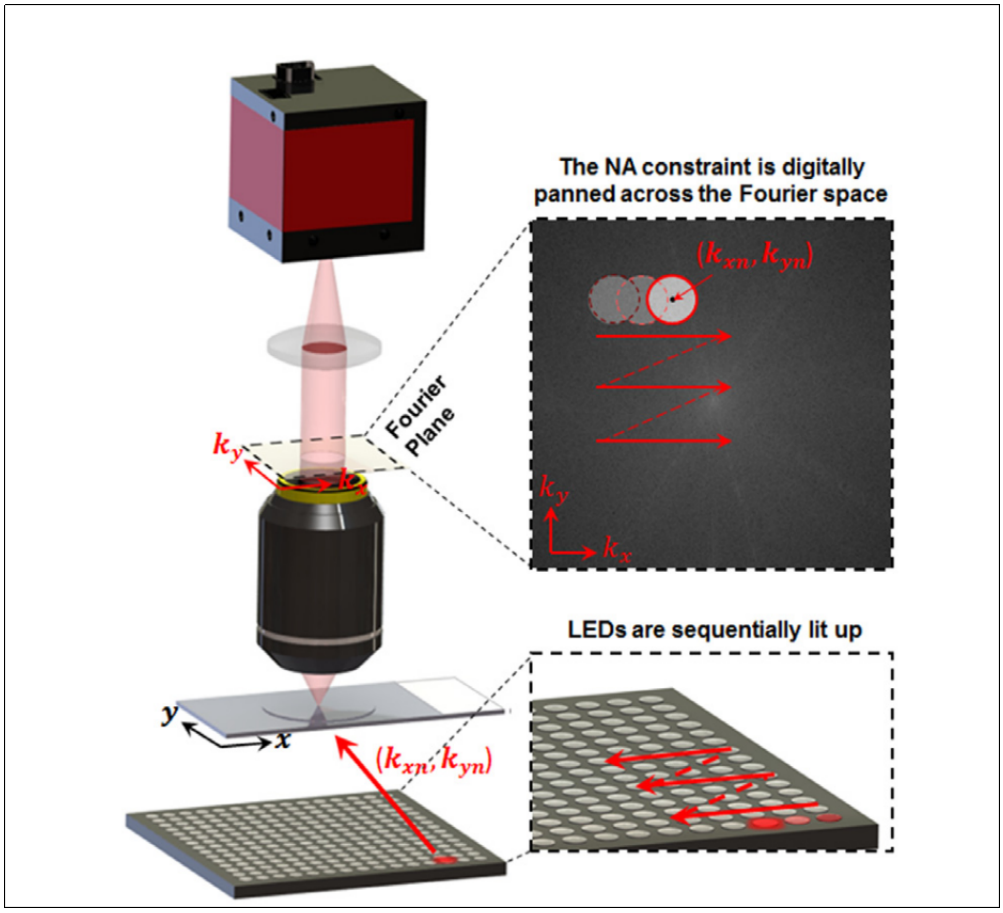


Figure 13: The physical setup in FPM uses an LED array that can illuminate the sample from different angles and allow for higher spatial frequency content to enter the NA[10]

Assuming we have n LED's, there will be n images that will need to be taken, in which each image describes the sample lit up from a single LED. Each LED acts as a point source of light and emits spherical waves, but we treat light as plane waves when they reach the sample given that the sample is sufficiently far away from the light source. Note that because FPM is a coherent imaging modality, in which the illumination light is uniform in wavelength, we can assign spatial frequency components to the light that hits the sample (if the light was spatially incoherent then we could not assume the light would have constant magnitude in spatial frequency).

The plane wave equation for light is as follows:

$$\begin{aligned} f(x,y,z) &= ae^{j(k_x x + k_y y + k_z z - \omega t)} \\ &= ae^{j(\vec{k}\cdot\vec{r} - \omega t)} \end{aligned}$$

where $\vec{k} = \hat{x}k_x + \hat{y}k_y + \hat{z}k_z$ is the wave vector, and
$\vec{r} = \hat{x}x + \hat{y}y + \hat{z}z$ is the cartesian coordinate vector

We can say that the plane wave propagates along the direction of $\vec{k}$. Since the magnitude of $\vec{k}$ is known, we can figure out the components of $\vec{k}$ by geometrically projecting the

wavevector onto the respective xyz axis.

However to understand how a change in illumination angle causes a shift in the fourier spectrum of the image, say we change the spatial frequency of one of the components of the wave by adding a phase factor Δk_x.

$$g'(x) = g(x)e^{i\Delta k_x x}$$

$$F[g'(x)] = F[g(x)e^{i\Delta k_x x}] = \int_{-\infty}^{\infty} e^{i\Delta k_x x} g(x) e^{-ik_x x} dx$$

$$= \int_{-\infty}^{\infty} g(x) e^{-i(k_x - \Delta k_x)x} dx = G(k_x - \Delta k_x)$$

Therefore, any change in the kx or ky components of the wave vector corresponds to a translating in the horizontal or vertical direction of the frequency spectrum of the wave vector. By illuminating the sample from different angles, we are effectively changing the kx/ky components of the wavevector and it is this that allows us to "shift" different parts of the fourier spectrum of the sample so that they fall into the low-pass filter (NA).

The basic premise of the FPM reconstruction is as follows

1. Gather n low resolution images, where image is illuminated by one of n LED on the LED matrix

2. Initialize a high-resolution image consisting of random values (or an interpolated scaled up version of one of the low-resolution image) with amplitude $\sqrt{I_h}$ and phase ϕ_h ($\sqrt{I_h}e^{j\phi_h}$)

3. Generate a low resolution estimate $\sqrt{I_l}e^{j\phi_l}$ by taking a circular subregion of the high-res fourier spectrum corresponding to a certain range of spatial frequencies dictated by the current LED illumination angle and performing inverse Fourier Transform

4. Replace the low-resolution estimate's amplitude with the intensity values of the actual low-resolution image taken with the corresponding LED $\sqrt{I_m}$. We now have $\sqrt{I_m}e^{j\phi_l}$.

5. Replace the current subregion in the high-resolution fourier spectrum with the fourier transform of the obtained $\sqrt{I_m}e^{j\phi_l}$.

6. Repeat steps 3-5 for all LEDs, eventually "filling in" a significant portion of the once completely estimated fourier domain and thus computationally widening the NA of the imaging system.

2.1 FPM Simulation Code

The FPM simulation code below consists of two main features:

1. Simulating a set of low resolution images each corresponding to a different LED illumination angle

2. Performing the FPM high-resolution intensity and phase reconstruction algorithm

While originally written in MATLAB [8], the code has been transcribed to python3 with extensive use of the numpy library.

```
from numpy.fft import fft, ifft, fft2, ifft2, ifftshift, fftshift
from matplotlib import colors
from skimage import data
import math
import matplotlib.pyplot as plt
import numpy as np

#creating the complex high resolution image with amplitude and phase
objectAmplitude = np.double(data.camera())
phase = np.double(data.brick())
phase = np.pi * phase / np.max(phase)
obj = objectAmplitude * np.exp(1j * phase)
fig, ax = plt.subplots(1,2)

amplitude_img = ax[0].imshow(np.abs(obj))
phase_img = ax[1].imshow(np.angle(obj))
ax[0].set_title('Sample Amplitude')
ax[1].set_title('Sample Phase')
ax[0].set_xlabel('x')
ax[0].set_ylabel('y', rotation = 0)
ax[1].set_xlabel('x')
ax[1].set_ylabel('y', rotation = 0)
```

Listing 1: Declaring imports and initializing the high resolution image

The above section of code initializes the high resolution input object with amplitude and phase.

```
#define LED array specifications
arraysize = 15 #number of LED's on one axis
LEDgap = 4     #4mm between adj LED's
LEDheight = 90 #90mm between LED and sample

# 1-D x and y position vectors denoting LED light point sources
xloc = np.linspace(-(arraysize//2), arraysize//2, arraysize)*LEDpitch_mm
yloc = np.linspace(-(arraysize//2), arraysize//2, arraysize)*LEDpitch_mm

#2x2 coordinate matrices for x and y
#ex. X[0] represents all the x values at y=0
#ex. Y[0] represents all the y values at x=0
#more information here: https://www.geeksforgeeks.org/numpy-meshgrid-
   ↪ function/

X,Y = np.meshgrid(xloc,yloc)
X = X.flatten()
Y = Y.flatten()

#kx_relative = -X/np.sqrt(X**2 + Y**2 + LEDheight**2)
#set up simulated Kx, Ky wave vectors (these are correlated directly with
   ↪ the NAShift_X and NAShift_Y)
NAShift_X = -X/np.sqrt(X**2 + Y**2 + LEDheight**2)
```

```
22 NAShift_Y = -Y/np.sqrt(X**2 + Y**2 + LEDheight**2)
```

Listing 2: Python example

We first begin by defining some parameters of the optical setup. In lines 1-4, we give the physical dimensions of the LED matrix in order to obtain the X and Y coordinate matrices of the point sources of light in lines 7-17.

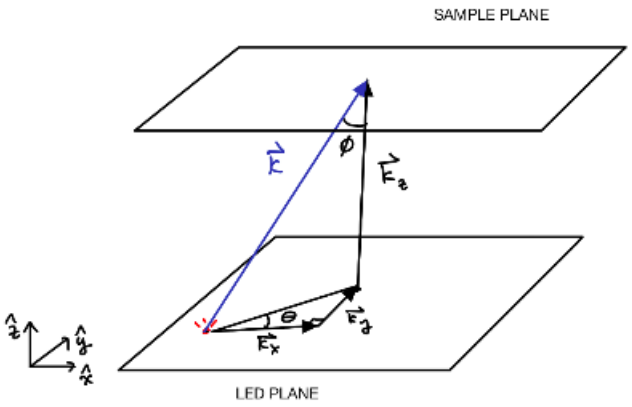


Figure 14: Shows the vector decomposition of the wavevector $\vec{k}$. Solving for the magnitudes of components $\vec{k_x}, \vec{k_y}$ relative to $|\vec{k}| = k_0$ is done in lines 21-22 in the cartesian domain

```
1  waveLength = .63e-6
2  k0 = 2*np.pi/waveLength
3  #spsize = sensor pixel size / magnification
4  #this is the physical size that a pixel in the low-res image corresponds
       ↪ to
5  spsize = 2.75e-6
6   #how much larger the hi-res image will be
7  scaling_factor = 4
8  #hi-res pixel size (smaller than the sensor pixel size)
9  psize = spsize/scaling_factor
10 NA= 0.08
11
12 #image size of the high resolution object
13 [m,n] = obj.shape
14
15 #image size of the low res
16 m1 = m//(scaling_factor)
17 n1 = n//(scaling_factor)
18
19 #image sequence of 225 low-resolution images corresponding to LED
20 imSeqLowRes = np.zeros([m1,n1, arraysize**2])  illuminations.
21
```

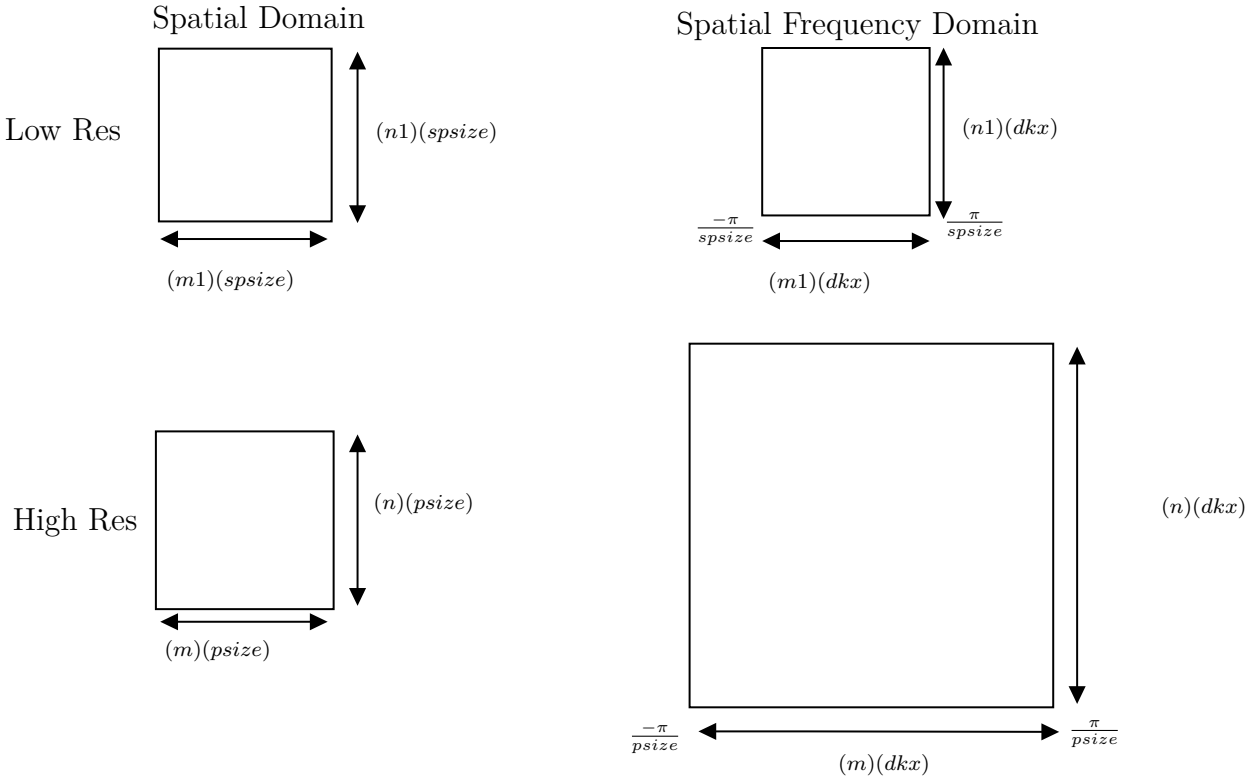
```
#kx, ky vectors
kx = k0 * NAShift_X
ky = k0 * NAShift_Y

#pixel size in fourier space
dkx = 2*np.pi/(psize * n)
dky = 2*np.pi/(psize * m)

#highest spatial frequency that can pass through NA (higher frequencies
   ↪ are filtered out)
cutoffFrequency = NA*k0

#nyquist frequency
kmax= np.pi/spsize

#pixel coordinates in fourier space of low res image
kx_lowres = np.linspace(-kmax, kmax, n1,endpoint=False)
ky_lowres = np.linspace(-kmax, kmax, m1,endpoint=False)
kxm, kym = np.meshgrid(kx_lowres, ky_lowres)

#defining the NA bandpass (or the CTF)
CTF = ((kxm**2 + kym **2) < cutoffFrequency ** 2)
objectFT = fftshift(fft2(obj))

for i in range(arraysize**2):
    #1/dkx represent num pixels per unit frequency
    kxc = np.int((n)/2 + kx[i]/dkx)
    kyc = np.int((m)/2 + ky[i]/dky)
    kyl = np.int(kyc - (m1)/2)
    kyh = np.int(kyc + (m1)/2)
    kxl = np.int(kxc - (n1)/2)
    kxh = np.int(kxc + (n1)/2)
    imSeqLowFT_mask = ((scaling_factor) ** 2 * objectFT[kyl:kyh, kxl:kxh])
   ↪  * CTF
    imSeqLowRes[:,:,i] = np.abs(ifft2(ifftshift(imSeqLowFT_mask)))
```

Listing 3: Python example

For the sake of clarity, a table has been prepared to keep track of the variables and their respective units (note: HR means high resolution and LR means low resolution):

Quantity	Meaning	Dimensions
kmax	nyquist limit from sampling	$\frac{rad}{mm}$
spsize	pixel size, LR image	mm
psize	pixel size, HR image	mm
n,m	dimensions of HR image	px
n1,m1	dimensions of LR image	px
NAShift_X,NAShift_Y	wave vector components in cartesian plane	unitless
kx,ky	wave vector spatial frequency components	$[k0] = \frac{rad}{mm}$
dkx, dky	pixel size in fourier space	$\frac{[2\pi]}{[psize][n]} = \frac{(rad)}{(mm)(px)}$
cutoffFrequency	max frequency allowed by NA	$[\text{NA}][\text{k0}] = \frac{[NA][2\pi]}{[\lambda]} = \frac{rad}{mm}$
kxc, kyc	pixel coordinate of circle centers in HR k-space	$\frac{[kx]}{[dkx]} = px$

To grasp the intuition behind physical units and pixel coordinates, let us consider 4 figures representing a low res image, high res image, and their corresponding frequency spectra.



In the spatial domain, the physical size (mm) of the low res image and the high res image are the same. However, the high res image includes more pixels, thus exhibiting a higher resolution (defined as pixels/mm). Note that in the space domain, the physical pixel size decreases to account for the shift from low to high resolutions. In the fourier spectrum however, the physical dimensions of the images vary. The high resolution fourier spectrum has a larger physical range, which makes sense due to the expanded range of spatial frequencies in a high resolution image. By the same line of logic, the low resolution fourier spectrum exhibits a smaller range of frequencies, and therefore a smaller physical width. It is now intuitive that we must process "subregions" in high res fourier space with the captured

low resolution images for the physical dimensions to work out.

1. LR image
    - pixel size (in mm/pix): spsize
    - num pixels on axis 0: n1
    - length: $(n1)(spsize) = l$
    - physical range: $[\frac{-l}{2}, \frac{l}{2}] = [\frac{-spsize*n1}{2}, \frac{spsize*n1}{2}]$
    - index array: $[\frac{-spsize*n1}{2} : n1 : \frac{spsize*n1}{2}]$
2. HR image
    - pixel size (in mm/pix): psize
    - num pixels on axis 0: n
    - length: $(n)(psize) = l$
    - physical range: $[\frac{-l}{2}, \frac{l}{2}] = [\frac{-psize*n}{2}, \frac{psize*n}{2}]$
    - index array: $[\frac{-psize*n}{2} : n : \frac{psize*n}{2}]$
3. LR fourier image
    - pixel size (in freq/pix): $\frac{2\pi}{n*spsize}$
    - length:$\frac{2\pi}{spsize}$
    - physical range: $[\frac{-\pi}{spsize}, \frac{\pi}{spsize}]$
    - index array (indices are physical units): $[\frac{-\pi}{spsize} : n1 : \frac{\pi}{spsize}] \leftrightarrow kx\_lowres, ky\_lowres$
4. HR fourier image
    - pixel size (in freq/pix): $\frac{2\pi}{n*psize}$
    - length:$\frac{2\pi}{psize}$
    - num pixels on axis 0: n
    - physical range: $[\frac{-\pi}{psize}, \frac{\pi}{psize}]$
    - index array (indices are referenced by frequency): $[\frac{-\pi}{psize} : n : \frac{\pi}{psize}]$
    - conversion from frequency to pixel: $\frac{1}{pixelsize}$

The above code creates a set of 225 low resolution images Now that we have our simulated set of 225 low resolution images, we can perform FPM to obtain a single high resolution image.

```
#INPUTS:
#iterations:(int) listing how many iterations of the reconstruction to run
#scaling_factor: (int) the ratio between size of high res image and low res image
#kx,ky: (1x225 array of float) magnitude of spatial frequency wave vector in x,y directions. These define the "circle centers" in the fourier domain.
#dkx,dky: (float) pixel size in fourier space. Acts as a conversion factor from physical units to pixel coordinates
#seq: (1D array of int) index array giving the indices of low resolution images to iterate over in the reconstruction algorithm. First index corresponds to the centermost LED and indices spiral out
#imseq: (3D array of images stacked on the axis=2) low resolution images

#OUTPUTS:
#objectRecover: (2d array) high resolution image (complex valued)
#objectRecoverFT: (2d array) high resolution fourier transform of image
def phaseRetrieval(iterations, scaling_factor, dkx, dky, CTF, imseq, kx, ky):
    numimg = imseq.shape[2]
    seq = gseq(numimg)
    #image size of the high resolution object
    m = imseq.shape[0] * scaling_factor
    n = imseq.shape[0] * scaling_factor

    #image size of low res reconstruction
    m1 = imseq.shape[0]
    n1 = imseq.shape[0]

    #initializing the high res image estimate stored in objectRecover
    objectRecover=np.zeros((m,n))
    objectRecoverFT = fftshift(fft2(objectRecover))
    for i in range(iterations):
        print("iteration: ", i)
        for j in range(numimg):
            #kxc,kyc define the center of the subregion we will be substituting in fourier space

            kxc = np.round(n/2 +kx[seq[j]]/dkx)
            kyc = np.round(m/2 + kx[seq[j]]/dky)

            #kyl,kyh, kxl,kxh define the edges of the subregion in pixel coordinates
            kyl = np.round(kyc - (m1)/2); kyh = np.round(kyc + (m1)/2)
            kxl = np.round(kxc - (n1)/2); kxh=np.round(kxc + (n1)/2)
            kxl = int(kxl)
            kyl = int(kyl)
            kxh = int(kxh)
            kyh = int(kyh)

            #taking the mask of the subregion and storing within lowResFT
            lowResFT = (scaling_factor)**2 * objectRecoverFT[kyl:kyh, kxl:kxh] * CTF

```

```
            #inverse fourier transform low resolution fourier spectrum to
    ↪ get the complex image
            im_lowRes = ifft2(ifftshift(lowResFT));

            #subsitute in the amplitude by the actual image (imseq[:,:,seq
    ↪ [j]]) but keep the phase from the estimate
            im_lowRes = (1/scaling_factor)**2 * imseq[:,:,j] * np.exp(1j *
    ↪  np.angle(im_lowRes))

            #fourier transform back
            lowResFT = fftshift(fft2(im_lowRes)) * CTF

            #substitute back into the high resolution back into its
    ↪ original place
            objectRecoverFT[kyl:kyh,kxl:kxh] = (1-CTF) * objectRecoverFT[
    ↪ kyl:kyh,kxl:kxh] + lowResFT

    objectRecover = ifft2(ifftshift(objectRecoverFT))
    return (objectRecover,objectRecoverFT)
```

Listing 4: Python example

3 Errors and Calibration

The main errors in FPM come from two domains: aberrations and misalignment. Aberrations stem from optical inconsistencies in the imaging system, especially in the lens. This in turn causes imperfections in the image that must be corrected for computationally. Unknown aberrations in the lens can be corrected for using Embedded Pupil Function Recovery algorithm (EPRY), which proves to be a fertile ground for future work. Misalignment, especially with LED positions and therefore calculated kx, ky positions in Fourier space can pose signficant challenges to the FPM reconstruction algorithm. This is because the low resolution images may not be substituted in their correct positions in fourier space. One solution to calibrate LED positions and correct for kx, ky values is to identify the physical kx, ky components that manifest as the the position of the circle centers in fourier space of the low-resolution images [11].

A K-Means clustering algorithm that distinguishes between brightfield and darkfield images has been developed by analyzing the DC component of images. It is in the brightfield images that the circle contours can be seen in fourier space. Moreover, preprocessing the fourier spectrum of images by dividing out the average mean, placing a binary step filter, then convolving with a guassian kernel with $\sigma = 2$ has also been implemented in python3.

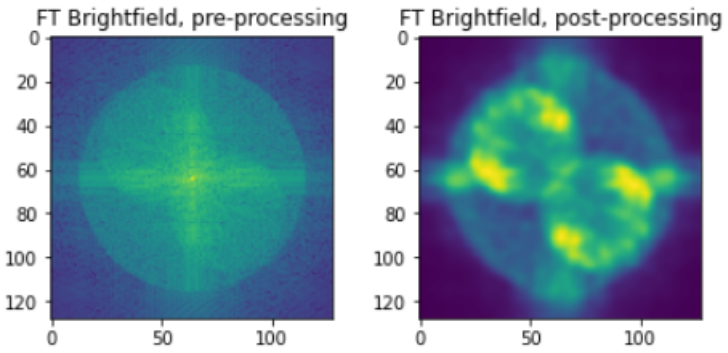


Figure 15: Fourier Spectrum of brightfield image on Caltech's FPM dataset. Processing the image by the means described above can help us clearly distinguish the twin pupils in fourier space and pinpoint circle centers

```
1
2  import numpy as np
3  from numpy.fft import fft, ifft, fft2, ifft2, ifftshift, fftshift
4  import matplotlib.pyplot as plt
5  from matplotlib import colors
6  from scipy.ndimage.filters import gaussian_filter
7  """
8  kMeans2Cluster: performs a K-means cluster on a 1 dimensional array into 2
       ↪  clusters
9
10 input:
11     X: a 1 dimensional array
12 output:
13     a 1 dimensional array consisting of 1's and 2's that delineate
       ↪ clusters
14     in which indices of 1's and 2's correspond to elements within X.
15 """
16 def kMeans2Cluster(X):
17     start = 0
18     end = len(X)-1
19     centroid_1 = np.random.choice(X)
20     centroid_2 = np.random.choice(X)
21     while(centroid_2 == centroid_1):
22         centroid_2 = np.random.choice(X)
23     newcentroid_1, newcentroid_2 = calcCentroid(X, centroid_1,centroid_2)
24     while (newcentroid_1 != centroid_1 and newcentroid_2 != centroid_2):
25         centroid_1 = newcentroid_1
26         centroid_2 = newcentroid_2
27         newcentroid_1,newcentroid_2 = calcCentroid(X, centroid_1,
       ↪ centroid_2)
28     Y = np.zeros(len(X))
29     for i in range(len(X)):
30         dist_c1 = abs(X[i]-centroid_1)
31         dist_c2 = abs(X[i] -centroid_2)
32         if dist_c1 < dist_c2:
33             Y[i] = 0
```

```
        else:
            Y[i] = 1
    return Y,newcentroid_1,newcentroid_2;

"""
helper function that calculates new centroid based off of previous
   ↪ centroids

X: input dataset
centroid_1: previous centroid 1
centroid_2: previous centroid 2
"""

def calcCentroid(X, centroid_1, centroid_2):
    y1 = []
    y2 = []
    for i in range(len(X)):
        dist_c1 = abs(X[i]-centroid_1)
        dist_c2 = abs(X[i] -centroid_2)
        if dist_c1 < dist_c2:
            y1.append(X[i])
        else:
            y2.append(X[i])
    c1 = np.mean(y1)
    c2 = np.mean(y2)
    return c1, c2
```

Listing 5: 1-Dimensional K Means Clustering Algorithm

```

from numpy.fft import fft, ifft, fft2, ifft2, ifftshift, fftshift
import math
import numpy.matlib
import matplotlib.pyplot as plt
from matplotlib import colors
import numpy as np
from scipy.ndimage.filters import gaussian_filter
import cv2

#I: images (intensity) 250x250x293. Interpret as imSeqLowRes
#xI: 2-d array of the X pixel indices (from meshgrid)
#yI: 2-d array of the Y pixel indices (from meshgrid)
#XYmid: tuple containing middle indices of image
#radP: radius of the NA (in pixels)
#sigmaG: sigma=2 within gaussian filtering

"""Program steps
    1) convert images into intensity valued fourier spectrum
    2) take the mean across all spectrum images-- this results in a "mean
   ↪ spectrum"
    3) take the mean value outside the 2NA support in the mean spectrum
    4) from the mean spectrum, floor out the noise outside the 2NA support
   ↪  (any pixels outside 2NA support with values less than 3*mean Image
    5)  divide out the modified mean spectrum from all the FT images
```

```
    6)  from those images, convolve them with Gaussian Kernel std.dev 2
    ↪ pixels"""

def calFI(I,xI, yI,XYmid, radP, sigmaG):
    FI = fftshift(fft2(I,axes=[1,0]),axes=[1,0]) #FT the image set
    avgFI = np.mean(np.abs(FI),2) #take the mean across all images
    w_2NA=np.sqrt((xI-XYmid[0])**2 + (yI-XYmid[1])**2)>=2*radP; #Outside 2
    ↪ NA support
    mT = np.mean(avgFI[w_2NA]) #the mean value outside the 2NA support
    avgFI2 = np.where(avgFI < 3*mT, 3*mT, avgFI)#if values are less than
    ↪ 3*mT then floor them to the 3*mT (essentially getting rid of noise
    ↪ outside the 2NA support).
    FIdiv = FI/np.tile(avgFI2[:,:,np.newaxis], [1,1, FI.shape[2]])

    FIdivG = cv2.GaussianBlur(np.abs(FIdiv),ksize=(0,0),sigmaX=2,
    ↪ borderType=cv2.BORDER_REPLICATE)#convolving with the Gaussian Kernel
    ↪  with std.dev = 2 pixels
    return FIdiv, FIdivG, FI, w_2NA
```

Listing 6: Preprocessing Code

4 Acknowledgements

I would like to sincerely thank our lab supervisor Professor Josh Brake for his unending support, enthusiasm, and guidance throughout this project. I would also like to thank my labmates Sathvika Anand, George Wang, and Kaanthi Pandhigunta for their open-minded collaboration and hard work in the lab.

References

[1] "Diffraction image." https://secure.math.ubc.ca/~cass/courses/m309-03a/m309-projects/krzak/. Accessed: 2021-7-28.

[2] "Diffraction diagram." http://roorda.vision.berkeley.edu/VS203BWebsite/Labs/VS203B_LAB8.pdf. Accessed: 2021-7-28.

[3] "Resolving diagram." http://labman.phys.utk.edu/phys222core/modules/m9/resolving_power.htm. Accessed: 2021-7-28.

[4] "fourier transform image." https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft. Accessed: 2021-7-27.

[5] "spatialfrequency, fourier transform." https://svi.nl/FourierTransform. Accessed: 2021-7-30.

[6] "Parseval's theorem." https://en.wikipedia.org/wiki/Parseval's_theorem. Accessed: 2021-7-27.

[7] "convolution." https://en.wikipedia.org/wiki/Convolution. Accessed: 2021-7-27.

[8] G. Zheng, *Fourier Ptychographic Imaging: A Matlab Tutorial*. Morgan & Claypool Publisher, 2016.

[9] "ctfatf." https://www.spiedigitallibrary.org/ContentImages/Journals/OPEGAR/53/8/083103/FigureImages/OE_53_8_083103_f007.png. Accessed: 2021-7-30.

[10] C. Y. Guoan Zheng, Roarke Horstmeyer, "Wide-field, high-resolution fourier ptychographic microscopy," *Nature Photonics*, vol. 7, pp. 739–745, 2013.

[11] L. W. Regina Eckert, Zachary F. Phillips, "Efficient illumination angle self-calibration in fourier ptychography," *Applied Optics*, vol. 57, pp. 5434–5442, 2018.