Frequency Domain OCT

by Ben Perry, Matthew Marcos, and Ege Ozemek

To begin we first import the neccessary modules and create a test phantom

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.io
from google.colab import files
from scipy.ndimage import gaussian_filter
from scipy.ndimage import gaussian_filter1d
from ipywidgets import interact, FloatSlider, IntSlider
from PIL import Image

def makecircle(Im, x,y, xloc, yloc, radius, signal):
    Im[(x-xloc)**2+(y-yloc)**2<radius**2]=signal

    return Im

def makephantom(Im,x,y,num_circles, rstart, rend, offcenter, signalobj,signalout):
    #Make outside circle of 110 width
    Im=makecircle(Im, x,y,0,0,110,signalout)
    #Make different radii for various resolution objects (all circles)
    radius=np.linspace(rstart,rend,num_circles)
    #Find angle so we can determine position for resolution objects
    angle=np.arange(0,2*np.pi,2*np.pi/num_circles)
    #Calculate where objects will appear in xlocs and ylocs array but round answer to integer
    xlocs=np.round(offcenter*np.cos(angle))
    ylocs=np.round(offcenter*np.sin(angle))
    #Make objects inside phantom using for loop and makecircle function
    for i in range(num_circles):
        Im=makecircle(Im, x,y,xlocs[i],ylocs[i],radius[i],signalobj)


    return Im
```

Optical Coherence Tomography is a very high resultion imaging system that can have resolutions between 10 µm to 1 µm. OCT uses visible light to infra-red as its E/M radiation, allowing it to be non-ionizing, require no contrast agents, and highly sensitive. However, the trade-off is that OCT has a low penetration depth between 1mm to 1cm. This makes OCT useful in areas like opthalmology, dentistry, an endoscopy.

Frequency-Domain OCT functions by measuring the frequency components of backscattered light from an object. In FD-OCT, light from a broadband source is split into two paths, a reference path and a sample path. The reference arm has a known path length, while the sample arm will have an unknown structure. The light reflected from the sample is combined with the reference path, creating an interference pattern in the frequency domain of the light signal. This interference pattern in the frequency-domain can be measured by a spectrometer. Then, by using a Fourier transform we can recover depth from the interference in the frequency-domain and return to the spatial domain.

In FD-OCT we use a Michelson interferometer to split light into the reference and sample arms.

The detected intensity is modeled as a function of wavenumber $k = \frac{2\pi}{\lambda}$

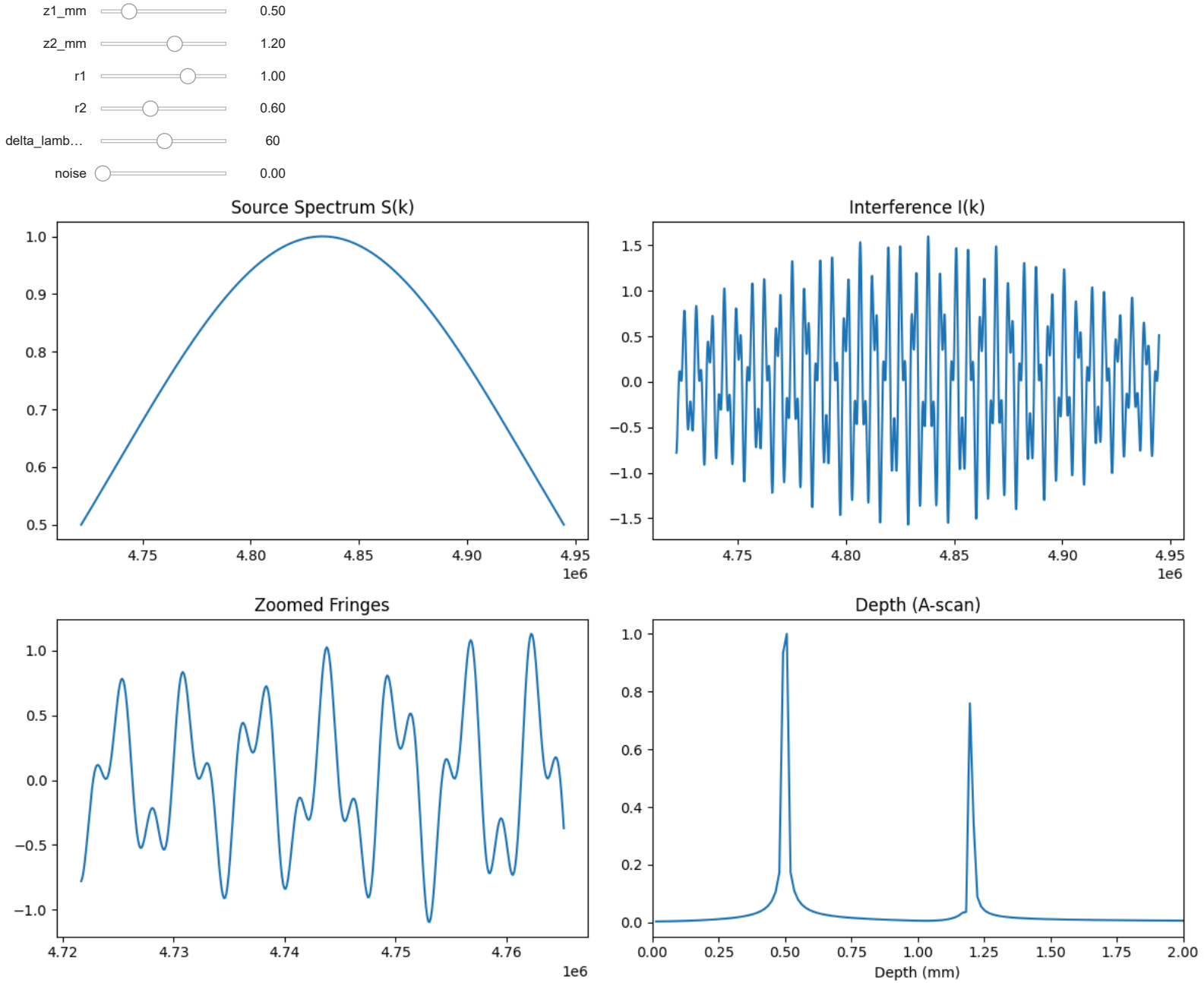
$$I(k) = S(k)[R_r + R_s + 2\sqrt{R_r R_s}\cos 2kz + \phi]$$

Where z is the depth of the reflector, $S(k)$ represents the spectrum of our source, and R_r and R_s represent our reflectivities.

Each reflector in depth creates a cosine modulation in k-space so that shallow reflector produce slow oscillations in k-space and deep reflectors produce fast oscillation in k-space.

```
def fd_oct_demo(z1_mm=0.5, z2_mm=1.2,
                r1=1.0, r2=0.6,
                delta_lambda_nm=100,
                noise=0.0):

    N = 2048 #number of samples in k-space (higher = smoother A-scan)
    lambda0 = 1300e-9 #center wavelength of light source is 1300 nm (good for tissue imaging, 800 nm for retinal imaging). Longer wavelenght = deeper penetration
    delta_lambda = delta_lambda_nm * 1e-9

    # Reflectors1
    z1 = z1_mm * 1e-3
    z2 = z2_mm * 1e-3

    # Define uniform k directly
    k0 = 2 * np.pi / lambda0
    delta_k = (2*np.pi/lambda0**2) * delta_lambda

    k = np.linspace(k0 - delta_k/2, k0 + delta_k/2, N)
    dk = k[1] - k[0]

    # Source spectrum
    sigma_k = delta_k / (2*np.sqrt(2*np.log(2)))
    S_k = np.exp(-(k - k0)**2 / (2 * sigma_k**2))

    # Interference
    I_k = S_k * (
        r1 * np.cos(2 * k * z1) +
        r2 * np.cos(2 * k * z2)
    )

    # remove DC / mean
    #eliminates the constant background intensity so that the true depth-resolved reflectors
    # are visible in the A-scan
    I_k = I_k - np.mean(I_k)

    # Add noise
    I_k += noise * np.random.randn(N)

    # FFT
    freq = np.fft.fftfreq(N, d=dk)
    A_z = np.abs(np.fft.fft(I_k))
    A_z = np.fft.fftshift(A_z)

    freq = np.fft.fftshift(freq)

    #normalize
    A_z = A_z / np.max(A_z)


    # Correct depth axis for cos(2kz)
```

```
    z = np.pi * freq

    # Keep positive depths
    mask = z > 0
    z = z[mask]
    A_z = A_z[mask]

    # Plot
    plt.figure(figsize=(12,8))

    plt.subplot(2,2,1)
    plt.plot(k, S_k)
    plt.title("Source Spectrum S(k)")

    plt.subplot(2,2,2)
    plt.plot(k, I_k)
    plt.title("Interference I(k)")

    plt.subplot(2,2,3)
    plt.plot(k[:400], I_k[:400])
    plt.title("Zoomed Fringes")

    plt.subplot(2,2,4)
    plt.plot(z*1e3, A_z)
    plt.xlim(0,2)
    plt.title("Depth (A-scan)")
    plt.xlabel("Depth (mm)")

    plt.tight_layout()
    plt.show()


interact(fd_oct_demo,
         z1_mm=FloatSlider(0.5, min=0.1, max=2, step=0.1),
         z2_mm=FloatSlider(1.2, min=0.1, max=2, step=0.1),
         r1=FloatSlider(1.0, min=0.1, max=1.5, step=0.1),
         r2=FloatSlider(0.6, min=0.1, max=1.5, step=0.1),
         delta_lambda_nm=IntSlider(60, min=20, max=100, step=10),
         noise=FloatSlider(0.0, min=0, max=1, step=0.05));
```



What the sliders do:

z1_mm, z2_mm:

→ move peaks left/right in the A-scan → also change fringe frequency in k-space

r1, r2:

→ change peak height

delta_lambda_nm:

→ changes source bandwidth → affects axial resolution (peak width) (Larger bandwidth Δλ → shorter coherence length → better axial resolution.)

noise:

→ adds randomness to interference signal → degrades SNR

Simplified OCT Image Simulation (`simulate_oct`)

This function generates an OCT-like image from a 2D phantom using image-processing approximations rather than a full physics-based OCT model.

The phantom is first normalized, then processed through several steps to mimic key characteristics of OCT images:

- resolution limitations (blurring),
- speckle noise,
- signal attenuation with depth (rolloff),
- and log-compressed intensity display.

These steps approximate how OCT images appear in practice, producing approximate texture and contrast. However, unlike the FD-OCT simulation, this method does not model spectral interference or Fourier-based depth reconstruction.

```
def simulate_oct(phantom,
                 axial_res=3,
                 lateral_res=3,
                 speckle_strength=0.4,
                 rolloff=2.0,
                 dynamic_range=40):

    #converts the phantom to decimal values and normalizes it so the brightest value is about 1
    phantom = phantom.astype(float)
    phantom /= np.max(phantom) + 1e-8

    # Resolution blur
    #blurs the image. axial_res blurs vertically/depth-wise, and lateral_res blurs horizontally. Bigger values mean worse resolution.
    blurred = gaussian_filter(phantom, sigma=(axial_res, lateral_res))

    # Speckle
    #adds grainy multiplicative noise, meant to mimic real OCT speckle.
    speckle = 1 + speckle_strength * np.random.randn(*phantom.shape)
    speckle_img = blurred * speckle

    # Depth rolloff
    #makes deeper parts of the image dimmer. Higher rolloff means signal disappears faster with depth.
    depth = np.linspace(0,1,phantom.shape[0])
```

```
    rolloff_curve = np.exp(-rolloff * depth)[:,None]
    rolled = speckle_img * rolloff_curve

    # Log compression
    #converts image to a decibel/log display, like many imaging systems do. dynamic_range controls how many low-intensity values are still visible.
    log_img = 20*np.log10(np.abs(rolled)+1e-6)
    log_img -= np.max(log_img)
    log_img = np.clip(log_img, -dynamic_range, 0)

    return log_img
```

Phantom Generation and Simplified OCT Visualization

This creates a synthetic phantom and visualizes how it appears under a simplified OCT imaging model.

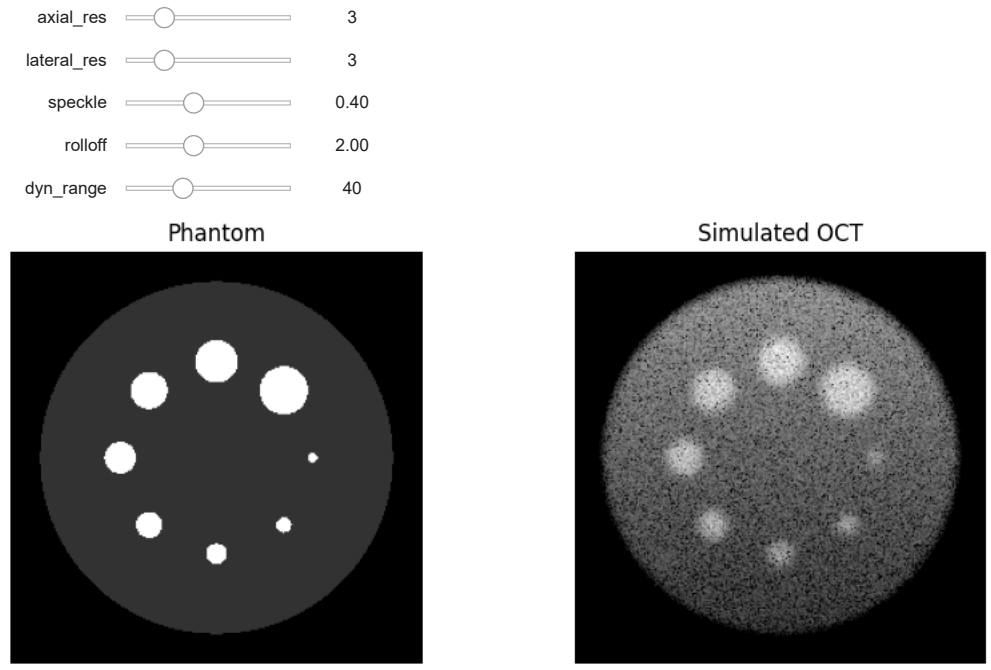
A 2D grid is first generated to define spatial coordinates. Using this grid, a circular phantom is constructed with smaller embedded circular structures of varying sizes. These features serve as reflectivity targets to evaluate image resolution and contrast.

The `phantom_view` function displays the original phantom alongside its corresponding OCT-like image generated by the `simulate_oct()` function. This allows for direct comparison between the true structure and its simulated imaging appearance.

The interactive sliders enable real-time updates to the simulated OCT image, making it easier to observe how changes in imaging conditions affect the visibility and clarity of structures within the phantom.

```
# Grid
#makes a 256 × 256 coordinate grid
N = 256
x = np.linspace(-128,128,N)
y = np.linspace(-128,128,N)
X, Y = np.meshgrid(x,y)

#creates our usual circular phantom with smaller bright circles inside it
base_phantom = makephantom(np.zeros((N,N)), X, Y,
                           num_circles=8,
                           rstart=3,
                           rend=15,
                           offcenter=60,
                           signalobj=1,
                           signalout=0.2)

def phantom_view(axial_res, lateral_res, speckle, rolloff, dyn_range):

    #takes the phantom and runs it through the simplified OCT-looking simulator (simulate_oct())
    oct_img = simulate_oct(base_phantom,
                           axial_res=axial_res,
                           lateral_res=lateral_res,
                           speckle_strength=speckle,
                           rolloff=rolloff,
                           dynamic_range=dyn_range)

    plt.figure(figsize=(10,4))

    #shows the original phantom
    plt.subplot(1,2,1)
    plt.imshow(base_phantom, cmap='gray')
```

```
    plt.title("Phantom")
    plt.axis('off')


    #shows the blurred/noisy/attenuated version labeled “Simulated OCT"
    plt.subplot(1,2,2)
    plt.imshow(oct_img, cmap='gray')
    plt.title("Simulated OCT")
    plt.axis('off')

    plt.show()


#creates sliders for:
# axial_res: vertical/depth blur
# lateral_res: horizontal blur
# speckle: grainy coherent noise
# rolloff: signal loss with depth
# dyn_range: displayed contrast range
interact(phantom_view,
         axial_res=IntSlider(value=3, min=1, max=10, step=1),
         lateral_res=IntSlider(value=3, min=1, max=10, step=1),
         speckle=FloatSlider(value=0.4, min=0, max=1, step=0.05),
         rolloff=FloatSlider(value=2.0, min=0, max=5, step=0.1),
         dyn_range=IntSlider(value=40, min=20, max=80, step=5));
```



Slider Function: creates sliders for:

- axial_res: vertical/depth blur

- lateral_res: horizontal blur
- speckle: grainy coherent noise
- rolloff: signal loss with depth
- dyn_range: displayed contrast range

Physics-Based FD-OCT B-scan Simulation

This function converts a 2D phantom into a simulated FD-OCT B-scan. The phantom is treated as a reflectivity map, where each column represents one depth profile that can be reconstructed as an A-scan.

First, the phantom is normalized so its values range from 0 to 1. The function then defines the OCT light source using a center wavelength of 1300 nm (infrared here but can be adjusted) and a chosen bandwidth, Δλ. The wavelength bandwidth is converted into a wavenumber bandwidth, Δk, so the spectral interference signal can be sampled uniformly in k-space.

For each column of the phantom, we treat each pixel as a small reflector at a specific depth. Each reflector contributes a cosine interference term, and all reflector contributions are summed to create the total spectral interference signal, (I(k)), for that A-scan.

The source spectrum, (S(k)), is applied to model the finite bandwidth of the OCT light source. The DC/background component is removed, and optional noise is added to simulate measurement noise.

Next, a Fourier transform is applied to convert the spectral interference signal from k-space into depth space. This produces one A-scan. Repeating this process for every column and stacking the A-scans creates a full B-scan image.

Finally, the B-scan is normalized, slightly smoothed laterally to reduce streaking artifacts, and log-compressed into decibel scale to mimic the display style of real OCT images.

```
def fd_oct_bscan_from_phantom(phantom, delta_lambda_nm=100, noise=0.0, dz_um=5):
    phantom = phantom.astype(float)
    phantom = phantom / (np.max(phantom) + 1e-8)

    Nz, Nx = phantom.shape

    # FD-OCT source / k-space setup
    N_k = 2048
    lambda0 = 1300e-9
    delta_lambda = delta_lambda_nm * 1e-9

    k0 = 2 * np.pi / lambda0
    delta_k = (2*np.pi/lambda0**2) * delta_lambda
    k = np.linspace(k0 - delta_k/2, k0 + delta_k/2, N_k)
    dk = k[1] - k[0]

    sigma_k = delta_k / (2*np.sqrt(2*np.log(2)))
    S_k = np.exp(-(k - k0)**2 / (2 * sigma_k**2))

    # Depth locations for phantom rows
    dz = dz_um * 1e-6
    z_phantom = np.arange(Nz) * dz

    # FFT depth axis
    freq = np.fft.fftfreq(N_k, d=dk)
```

```
    freq = np.fft.fftshift(freq)
    z_fft = np.pi * freq

    mask = z_fft > 0
    z_pos = z_fft[mask]

    bscan = []

    for col in range(Nx):
        reflectivity = phantom[:, col]

        # Sum contributions from all depths in this column
        I_k = np.zeros_like(k)

        for i in range(Nz):
            I_k += reflectivity[i] * np.cos(2 * k * z_phantom[i])

        I_k = S_k * I_k

        # remove DC/background and add noise
        I_k = I_k - np.mean(I_k)
        I_k += noise * np.random.randn(N_k)

        # FFT to recover A-scan
        A_z = np.abs(np.fft.fft(I_k))
        A_z = np.fft.fftshift(A_z)
        A_z = A_z[mask]

        bscan.append(A_z)

    bscan = np.array(bscan).T

    # normalize
    bscan = bscan / (np.max(bscan) + 1e-8)

    # Slight lateral smoothing to reduce vertical streaks
    bscan = gaussian_filter(bscan, sigma=(0, 1))

    # Log compression for OCT-style display
    bscan_log = 20*np.log10(bscan + 1e-6)
    bscan_log -= np.max(bscan_log)

    return z_pos, bscan_log
```

Comparison of Simplified and Physics-Based OCT Simulations

This cell compares three images to highlight the difference between a simplified OCT model and a physics-based FD-OCT reconstruction.

- **Input Phantom:**
  The original reflectivity map used as the ground truth. This represents the object being imaged.
- **Simplified OCT-Like Image:**
  Generated using the `simulate_oct()` function, which applies image processing steps such as blurring, speckle noise, depth rolloff,

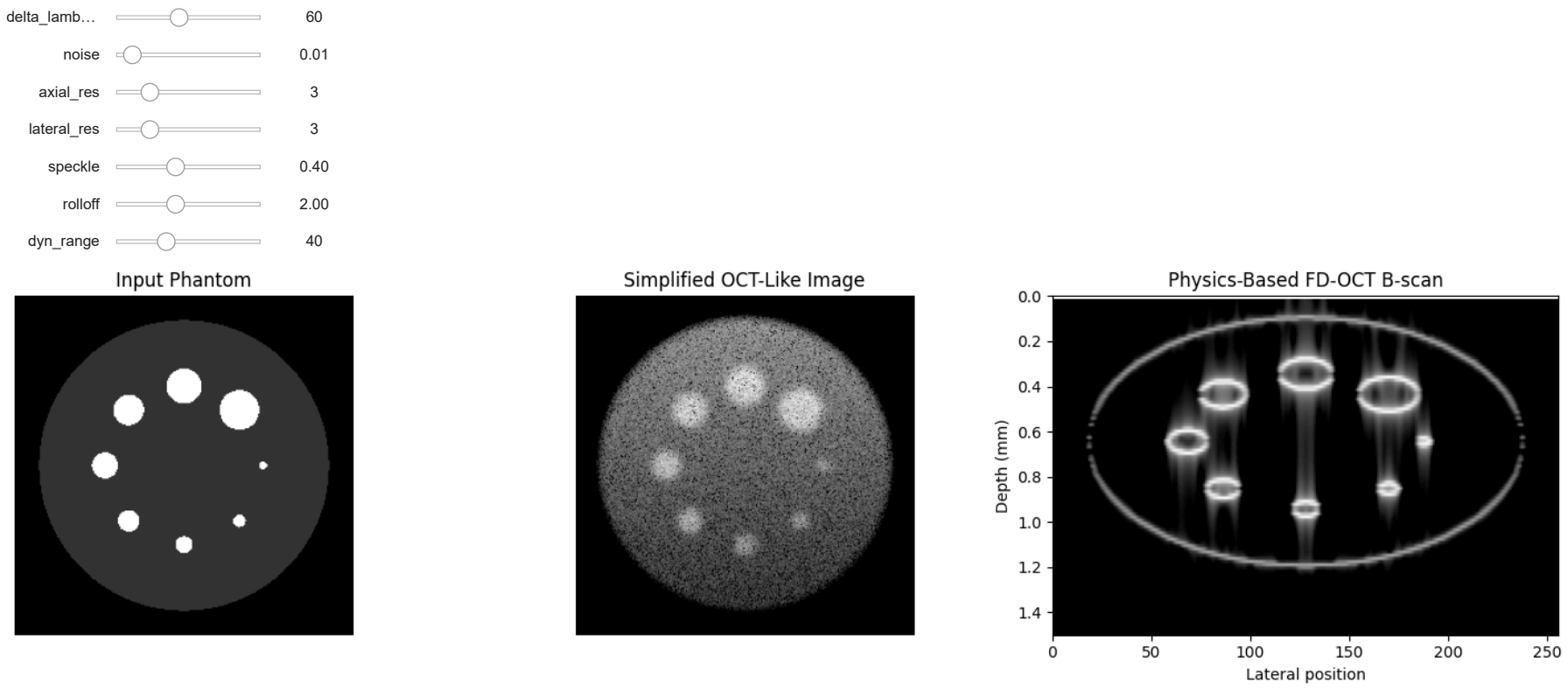
and log compression. This produces an OCT-like appearance but does not model the underlying physics of OCT signal formation.

- **Physics-Based FD-OCT B-scan:**
  Generated using `fd_oct_bscan_from_phantom()`, which simulates spectral interference in k-space and reconstructs depth using a Fourier transform for each A-scan. The resulting image is displayed in decibel scale with a fixed dynamic range to mimic real OCT visualization.

The depth axis is scaled in millimeters and flipped so that shallow regions appear at the top and deeper regions at the bottom, consistent with standard OCT images. The fixed display range (vmin = -35 dB, vmax = 0 dB) ensures consistent contrast and suppresses low-intensity noise.

```
    def compare_oct_models(delta_lambda_nm, noise, axial_res, lateral_res, speckle, rolloff, dyn_range):
        z_fd, fd_bscan = fd_oct_bscan_from_phantom(
            base_phantom,
            delta_lambda_nm=delta_lambda_nm,
            noise=noise,
            dz_um=5
        )

        simple_oct = simulate_oct(
            base_phantom,
            axial_res=axial_res,
            lateral_res=lateral_res,
            speckle_strength=speckle,
            rolloff=rolloff,
            dynamic_range=dyn_range
        )

        plt.figure(figsize=(15,4))

        plt.subplot(1,3,1)
        plt.imshow(base_phantom, cmap='gray')
        plt.title("Input Phantom")
        plt.axis('off')

        plt.subplot(1,3,2)
        plt.imshow(simple_oct, cmap='gray', vmin=-dyn_range, vmax=0)
        plt.title("Simplified OCT-Like Image")
        plt.axis('off')

        plt.subplot(1,3,3)
        plt.imshow(fd_bscan, cmap='gray', aspect='auto',
                   extent=[0, base_phantom.shape[1], z_fd[-1]*1e3, z_fd[0]*1e3],
                   vmin=-35, vmax=0)
        plt.ylim(1.5, 0)
        plt.title("Physics-Based FD-OCT B-scan")
        plt.xlabel("Lateral position")
        plt.ylabel("Depth (mm)")

        plt.tight_layout()
        plt.show()


    interact(compare_oct_models,
             delta_lambda_nm=IntSlider(value=60, min=30, max=100, step=10),
             noise=FloatSlider(value=0.01, min=0, max=0.1, step=0.01),
```

```
        axial_res=IntSlider(value=3, min=1, max=10, step=1),
        lateral_res=IntSlider(value=3, min=1, max=10, step=1),
        speckle=FloatSlider(value=0.4, min=0, max=1, step=0.05),
        rolloff=FloatSlider(value=2.0, min=0, max=5, step=0.1),
        dyn_range=IntSlider(value=40, min=20, max=80, step=5));
```



Slider Effects: Physics-Based vs. Simplified OCT

- **Δλ (Source Bandwidth):**
  Controls axial resolution in the physics-based FD-OCT model. Larger Δλ produces sharper, thinner features in depth (better resolution).
- **Noise (FD-OCT):**
  Adds randomness to the interference signal, reducing signal-to-noise ratio and making the B-scan appear noisier.
- **Axial Resolution (Simplified Model):**
  Controls vertical (depth) blurring in the image. Higher values make features appear more spread out in depth.
- **Lateral Resolution (Simplified Model):**
  Controls horizontal blurring. Higher values reduce sharpness across the image.

- **Speckle Strength (Simplified Model):**
  Adds grainy multiplicative noise, mimicking coherent imaging artifacts.
- **Rolloff (Simplified Model):**
  Simulates signal attenuation with depth. Higher values make deeper regions darker.
- **Dynamic Range (Simplified Model):**
  Controls contrast in the log-compressed image. Lower values increase contrast but may hide weaker signals.

These sliders highlight the difference between the two models: the physics-based FD-OCT simulation is primarily governed by fundamental parameters like bandwidth and noise, while the simplified model uses image-processing parameters to approximate OCT appearance.

Interpretation of Physics-Based FD-OCT B-scan Output

Outer ring is clearly defined: The boundary of the phantom appears bright because it represents a strong change in reflectivity (refractive index), which produces a high OCT signal.

Inner circular structures are visible: The smaller circles are reconstructed at the correct lateral positions and depths, confirming that the FD-OCT model is accurately capturing spatial information.

Background appears dark: Weak signals are suppressed after log compression, resulting in a darker background. This mimics real OCT systems, where only sufficiently strong reflections are displayed.

Vertical streaks are present but reduced: These arise because each column is reconstructed independently as an A-scan. After applying smoothing and log compression, these streaks are less prominent and more consistent with realistic OCT imaging artifacts.

Overall, the physics-based FD-OCT simulation (fd_oct_bscan_from_phantom()) produces edge-enhanced reflectivity patterns because OCT detects refractive index changes. Unlike the simplified model, which applies image filters, this model reconstructs depth by simulating spectral interference and performing a Fourier transform for each A-scan.

Quantitative Measurement of Axial Resolution

This code measures axial resolution using a simplified FD-OCT simulation of a single reflector.

The function `measure_axial_resolution()` generates a spectral interference signal for one reflector at a fixed depth. A Gaussian source spectrum is applied, and the signal is converted from k-space to depth using a Fourier transform, producing an A-scan.

To quantify resolution, the FWHM of the A-scan peak is calculated. This represents how “spread out” a point reflector appears in depth and is used as a measure of axial resolution.

The function is evaluated over a range of source bandwidths (Δλ), and the resulting FWHM values are stored. A small amount of smoothing is applied to reduce numerical noise from discrete sampling.

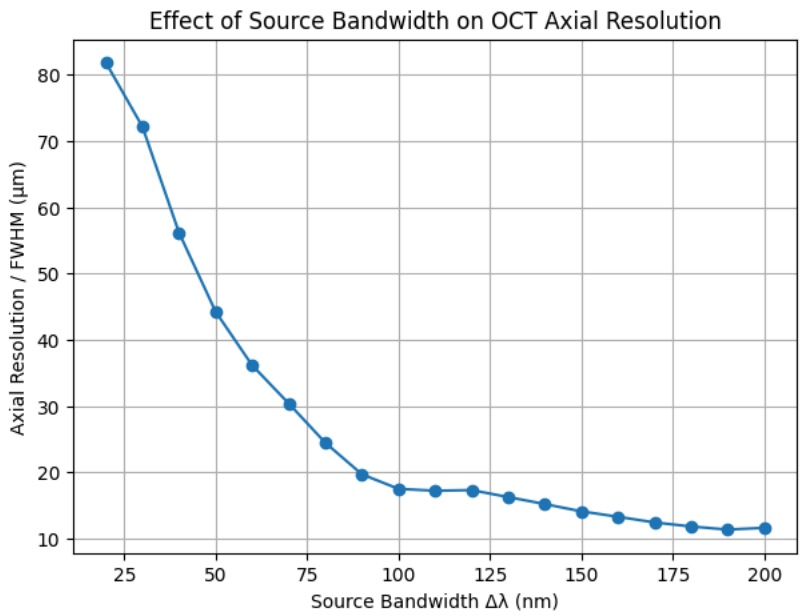
Finally, the plot shows axial resolution (FWHM) as a function of bandwidth, allowing for a quantitative comparison of how source bandwidth affects OCT performance.

```
def measure_axial_resolution(delta_lambda_nm, z1_mm=0.8, r1=1.0):
    N = 2048
```

```
    lambda0 = 1300e-9
    delta_lambda = delta_lambda_nm * 1e-9

    k0 = 2 * np.pi / lambda0
    delta_k = (2*np.pi/lambda0**2) * delta_lambda
    k = np.linspace(k0 - delta_k/2, k0 + delta_k/2, N)
    dk = k[1] - k[0]

    sigma_k = delta_k / (2*np.sqrt(2*np.log(2)))
    S_k = np.exp(-(k - k0)**2 / (2 * sigma_k**2))

    z1 = z1_mm * 1e-3

    I_k = S_k * r1 * np.cos(2 * k * z1)
    I_k -= np.mean(I_k)

    A_z = np.abs(np.fft.fft(I_k))
    A_z = np.fft.fftshift(A_z)

    freq = np.fft.fftfreq(N, d=dk)
    freq = np.fft.fftshift(freq)

    z = np.pi * freq
    mask = z > 0
    z = z[mask]
    A_z = A_z[mask]

    A_z = A_z / np.max(A_z)

    # Find FWHM around main peak
    peak_idx = np.argmax(A_z)
    half_max = 0.5

    left_idx = peak_idx
    while left_idx > 0 and A_z[left_idx] > half_max:
        left_idx -= 1

    right_idx = peak_idx
    while right_idx < len(A_z)-1 and A_z[right_idx] > half_max:
        right_idx += 1

    fwhm_m = z[right_idx] - z[left_idx]
    return fwhm_m * 1e6  # micrometers
```

```
bandwidths = np.arange(20, 210, 10)
resolutions = []

for bw in bandwidths:
    res = measure_axial_resolution(bw)
    resolutions.append(res)

resolutions_smooth = gaussian_filter1d(resolutions, sigma=1)

plt.figure(figsize=(7,5))
plt.plot(bandwidths, resolutions_smooth, marker='o')
plt.xlabel("Source Bandwidth Δλ (nm)")
plt.ylabel("Axial Resolution / FWHM (μm)")
plt.title("Effect of Source Bandwidth on OCT Axial Resolution")
```

```
plt.grid(True)
plt.show()
```



Effect of Source Bandwidth on Axial Resolution

The plot shows the relationship between source bandwidth and axial resolution, measured using the full width half max (FWHM) of the reconstructed A-scan peak.

As the bandwidth increases, the axial resolution improves (FWHM decreases). This occurs because a broader bandwidth corresponds to a shorter coherence length, allowing the system to better distinguish closely spaced reflectors in depth.

The curve shows a rapid improvement at lower bandwidths, followed by diminishing returns at higher bandwidths, consistent with OCT reality.

| Qualitatitve display of quantitative axial resolution analysis |

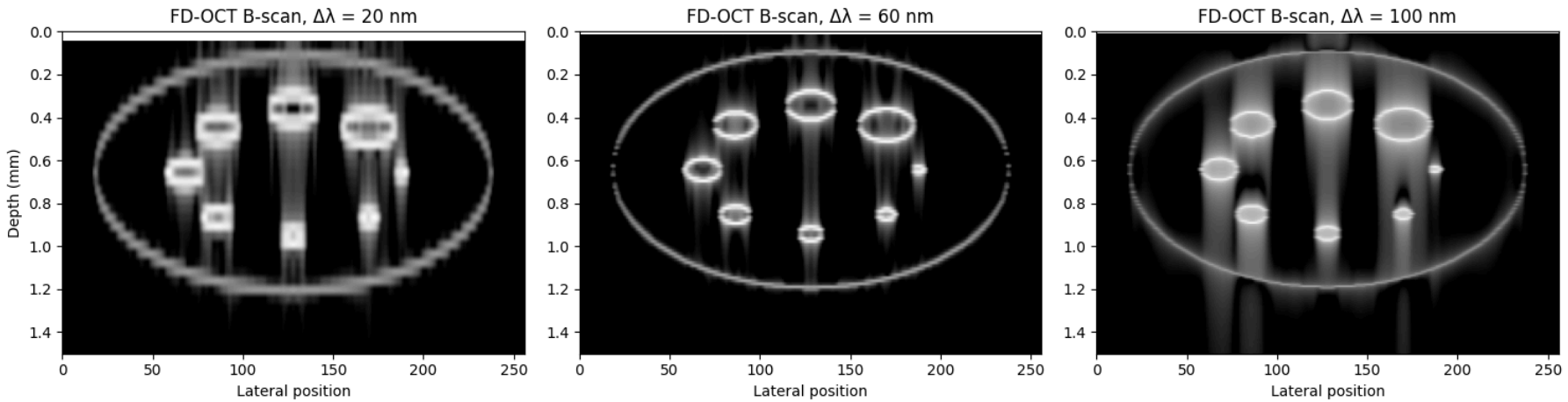
Generating B-scans at Different Source Bandwidths

This code generates FD-OCT B-scans for multiple source bandwidths (Δλ = 20 nm, 60 nm, and 100 nm) using the same phantom. Each B-scan is reconstructed using the physics-based FD-OCT model.

All images are displayed using the same depth range and intensity scaling (vmin = -35 dB, vmax = 0 dB) to ensure a fair comparison. This allows differences in image appearance to be attributed solely to changes in bandwidth rather than display settings.

The resulting images are shown side-by-side to visually compare how bandwidth affects OCT image quality.

```
bandwidths_to_show = [20, 60, 100]

plt.figure(figsize=(15,4))

for i, bw in enumerate(bandwidths_to_show):
    z_fd, fd_bscan = fd_oct_bscan_from_phantom(
        base_phantom,
        delta_lambda_nm=bw,
        noise=0.01,
        dz_um=5
    )

    plt.subplot(1, 3, i+1)
    plt.imshow(fd_bscan, cmap='gray', aspect='auto',
               extent=[0, base_phantom.shape[1], z_fd[-1]*1e3, z_fd[0]*1e3],
               vmin=-35, vmax=0)
    plt.title(f"FD-OCT B-scan, Δλ = {bw} nm")
    plt.xlabel("Lateral position")
    if i == 0:
        plt.ylabel("Depth (mm)")
    plt.ylim(1.5, 0)

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



Visual Effect of Source Bandwidth on OCT B-scans

These B-scans show how source bandwidth affects axial resolution. At lower bandwidth, the reflectors appear more blurred in depth because the coherence length is longer. As bandwidth increases, the reflector boundaries become sharper, indicating improved axial resolution. This visually supports the quantitative FWHM analysis.

The intermediate bandwidth appears visually cleaner (less streak artifacts) because it balances resolution and artifact visibility. At higher bandwidths, improved resolution comes with increased sidelobes and FFT-related artifacts, which are amplified by log compression.

Uploading and Simulating Custom Images

Upload your own image and visualize how it would appear under a simplified OCT imaging model.

- **Image Upload:**
  The `upload_image()` function allows you to select an image file, converts it to grayscale, and normalizes its intensity values. The uploaded image is stored and flagged for use in later processing.
- **Image Selection:**
  The viewer function checks whether an image has been uploaded. If so, it uses the uploaded image as the input phantom. Otherwise, it defaults to the synthetic phantom used earlier in the notebook.
- **OCT Simulation:**
  The selected image is passed into the `simulate_oct()` function, which applies resolution blur, speckle noise, depth rolloff, and log compression to generate an OCT-like image.
- **Interactive Controls:**
  Sliders allow the user to adjust axial resolution, lateral resolution, speckle strength, rolloff, and dynamic range in real time, making it possible to explore how these parameters affect image quality.

The resulting output displays the original input image alongside the simulated OCT image for direct comparison.

```
# STORAGE FOR IMAGE
user_image = {"img": None}
use_upload = {"flag": False}

# UPLOAD FUNCTION
def upload_image():
    uploaded = files.upload()
    filename = list(uploaded.keys())[0]

    img = Image.open(filename).convert('L')
    img = np.array(img).astype(float)

    # normalize
    img /= np.max(img)

    user_image["img"] = img
    use_upload["flag"] = True

    print("Image uploaded successfully!")


# UPDATED VIEWER FUNCTION
def oct_image_view(axial_res, lateral_res, speckle, rolloff, dyn_range):

    #either uses uploaded image or defaults to the synthetic phantom
    if use_upload["flag"] and user_image["img"] is not None:
        phantom = user_image["img"]
        title_left = "Uploaded Image (Input)"
    else:
        phantom = base_phantom
        title_left = "Synthetic Phantom"
```

```
        title_left = "Synthetic Phantom"

    #still uses the simplified image-quality simulator (so same as cell above unless image uploaded)
    oct_img = simulate_oct(phantom,
                           axial_res=axial_res,
                           lateral_res=lateral_res,
                           speckle_strength=speckle,
                           rolloff=rolloff,
                           dynamic_range=dyn_range)

    plt.figure(figsize=(10,4))

    # Input image
    plt.subplot(1,2,1)
    plt.imshow(phantom, cmap='gray')
    plt.title(title_left)
    plt.axis('off')

    # OCT output
    plt.subplot(1,2,2)
    plt.imshow(oct_img, cmap='gray')
    plt.title("Simulated OCT Output")
    plt.axis('off')

    plt.show()



upload_image()



# sliders
interact(oct_image_view,
         axial_res=IntSlider(value=3, min=1, max=10, step=1),
         lateral_res=IntSlider(value=3, min=1, max=10, step=1),
         speckle=FloatSlider(value=0.4, min=0, max=1, step=0.05),
         rolloff=FloatSlider(value=2.0, min=0, max=5, step=0.1),
         dyn_range=IntSlider(value=40, min=20, max=80, step=5));
```

Choose Files OCT.png

OCT.png(image/png) - 605281 bytes, last modified: 4/24/2026 - 100% done

```
Saving OCT.png to OCT (1).png
Image uploaded successfully!
```

axial_res 3

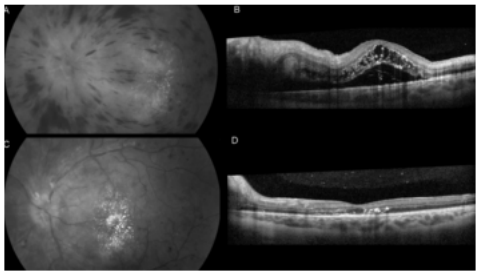
lateral_res 3

speckle 0.40

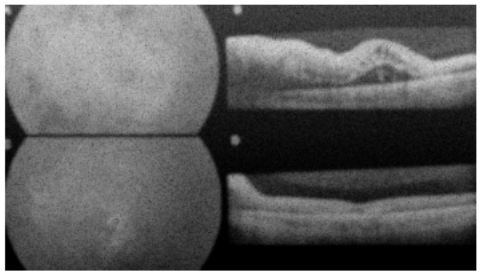
rolloff 2.00

dyn_range 40

Uploaded Image (Input)



Simulated OCT Output



Effect of Image Quality Parameters on Simulated OCT Output

- **Axial Resolution:**
  Controls vertical (depth) blurring. Higher values reduce axial resolution, causing features to appear more spread out in depth.
- **Lateral Resolution:**
  Controls horizontal blurring across the image. Increasing this value reduces lateral resolution, making structures appear less sharp side-to-side.
- **Speckle Strength:**
  Adds multiplicative noise to simulate OCT speckle. Higher values increase graininess and reduce image clarity.
- **Rolloff:**
  Simulates signal attenuation with depth. Larger values cause deeper regions of the image to appear darker, mimicking reduced signal strength in real OCT systems.
- **Dynamic Range:**
  Controls how intensities are displayed after log compression. Lower values increase contrast but may hide weaker signals, while higher values reveal more detail in low-intensity regions.

SPACER FOR PROPER PRINTING

SPACER FOR PROPER PRINTING

SPACER FOR PROPER PRINTING

SPACER FOR PROPER PRINTING

OCT SIMULATOR

Infrared to visible-light optical coherence tomography for ocular imaging

Matt Marcos, Ege Ozemek, Ben Perry

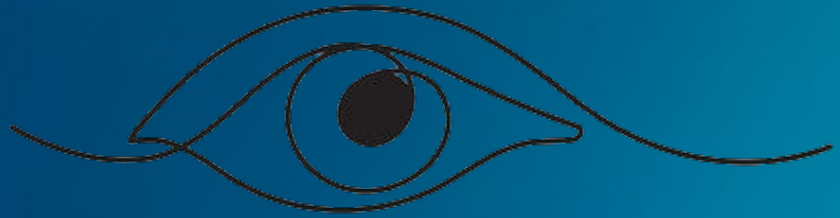
Introduction

Optical Coherence Tomography (OCT) is a noninvasive, depth-resolved imaging modality often used for the eye. OCT uses non-ionizing, shorter wavelength E/M radiation (visible light to infrared range), which allows it to provide higher axial resolution (10 µm - 1µm) and enhanced contrast for retinal layers. However, the trade-off is less penetration (1mm - 1cm).

Clinical and research applications include fine structural imaging, functional contrast, and early disease detection in ophthalmology, dentistry, and endoscopy.

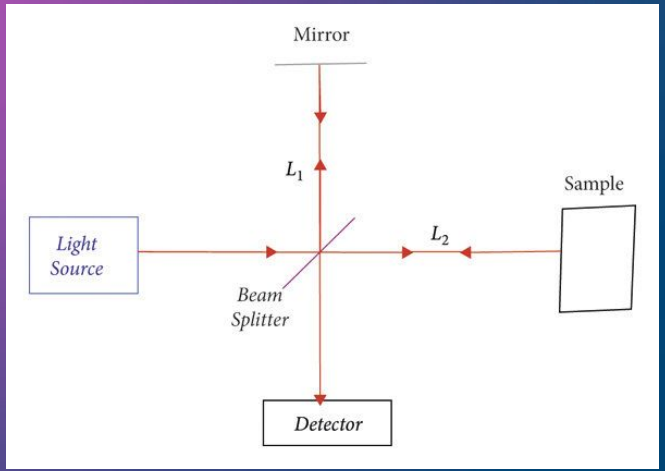
01

Frequency Domain - OCT MODEL



Interferometry basics

FD-OCT functions by measuring the frequency components of backscattered light from an object. In FD-OCT, light from a broadband source is split by a Michelson interferometer into two paths, a reference path and a sample path, and then recombined to reveal tissue depth information from interference.



The reference arm reflects off a mirror at a known distance and provides a baseline path length, while the sample arm reflects off tissue of different depths in the sample. The recombined paths create an interference pattern that is detected in the frequency-domain. If the sample arm pathlength matches that of the reference arm (in-phase), they interfere strongly. If they do not match (out of-of-phase), then there will be little or no signal. This enables determination of how deep a reflection came by using the Fourier Transform to recover tissue depth in the spatial-domain.

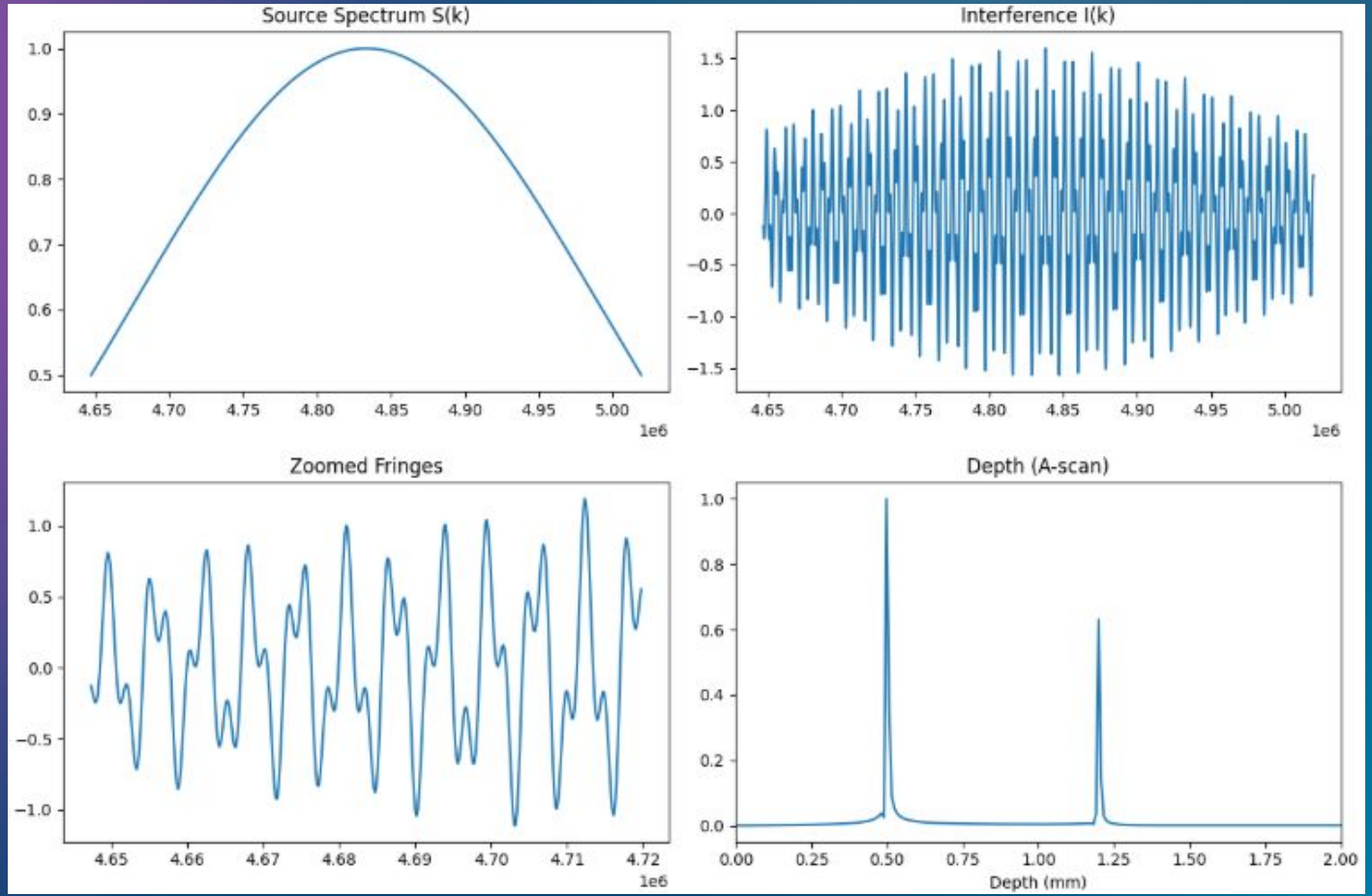
The Equation

How OCT converts interference in frequency (k-space) into depth information

$$I(k) = S(k)[R_r + R_s + 2\sqrt{R_r R_s}\cos 2kz + \phi]$$

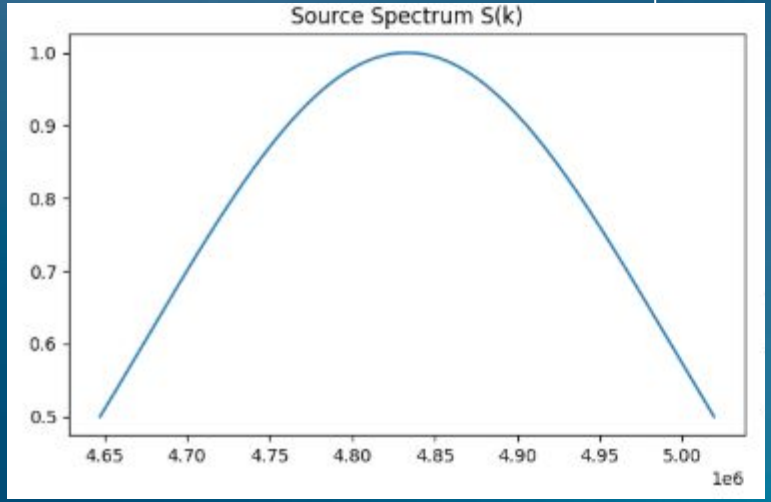
- The detected intensity, I, is modeled as a function of wavenumber, $k = 2\pi/\lambda$
- S(k) represents the spectrum of our source
- z is the depth of the reflecting tissue
- R_r and R_s represent the reflectivities of the reference arm and sample arm, respectively.

Key Idea: Each reflector in depth creates a cosine modulation in k-space so that shallow reflectors produce slow oscillations in k-space and deep reflectors produce fast oscillation in k-space. Thus allowing the Fourier Transform to retrieve depth information in the spatial-domain

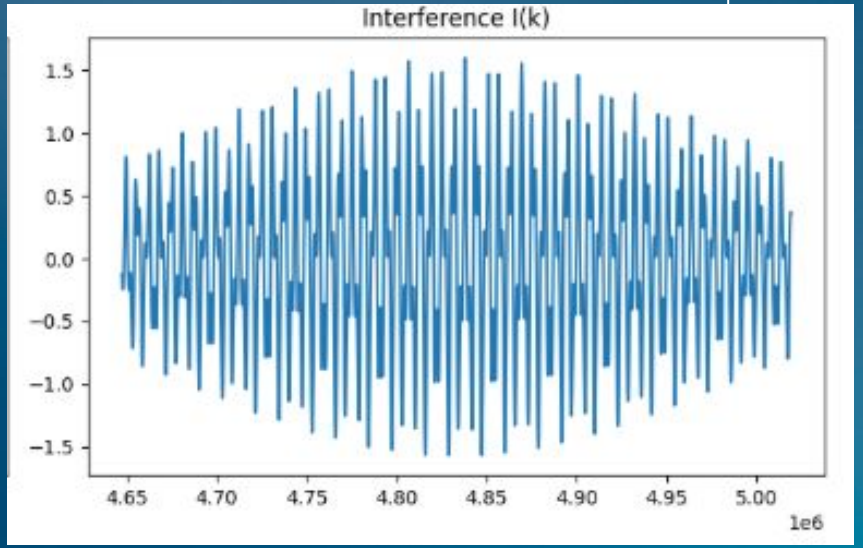

Source Spectrum S(k)
1.0
0.9
0.8
0.7
0.6
0.5
4.65
4.70
4.75
4.80
4.85
4.90
4.95
5.00
1e6
Interference I(k)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
4.65
4.70
4.75
4.80
4.85
4.90
4.95
5.00
1e6
Zoomed Fringes
1.0
0.5
0.0
−0.5
−1.0
4.65
4.66
4.67
4.68
4.69
4.70
4.71
4.72
1e6
Depth (A-scan)
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Depth (mm)

1. Source Spectrum Plot

- Gaussian broadband light source
- Width controlled by change in wavelength, Δλ
- Broader wavelength band → wider spectrum → better axial resolution



2. Interference Plot

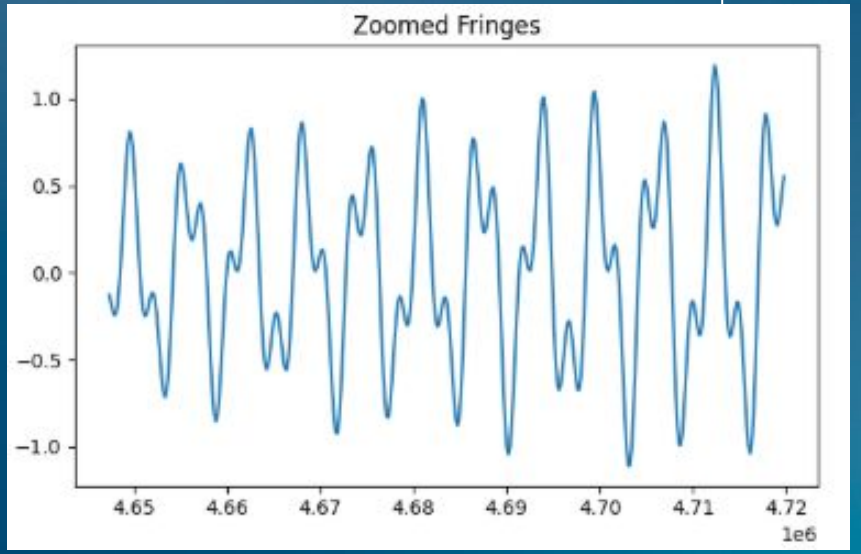


- Measured signal after light reflects from sample and interferes with the reference arm
- Oscillations = interference fringes
- Each reflector adds a cosine term
- Depth is encoded as frequency of oscillation in k-space
  - shallow reflector → slow oscillation
  - deep reflector → fast oscillation

3. Zoomed Fringes Plot

- Visualization of two superimposed cosine signal

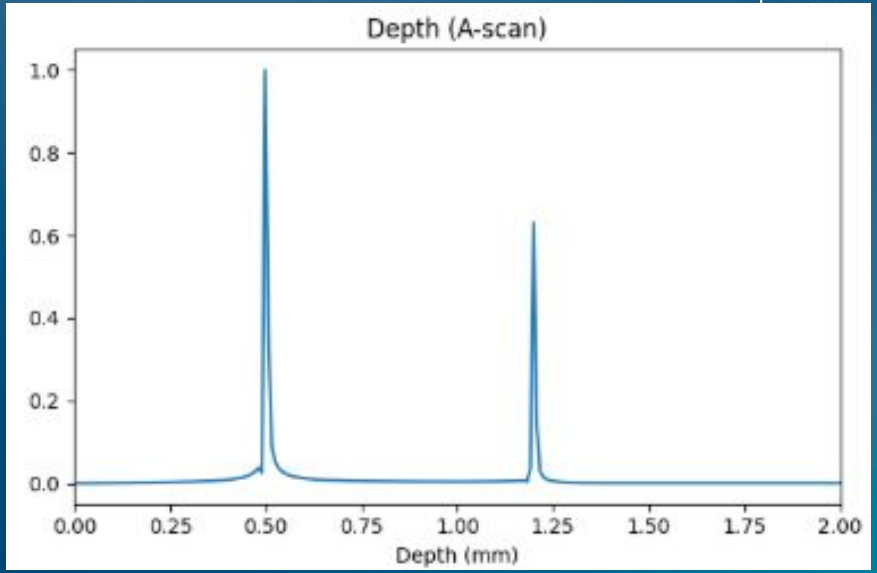
- Gets separated by the FT



4. Depth (A-scan) Plot

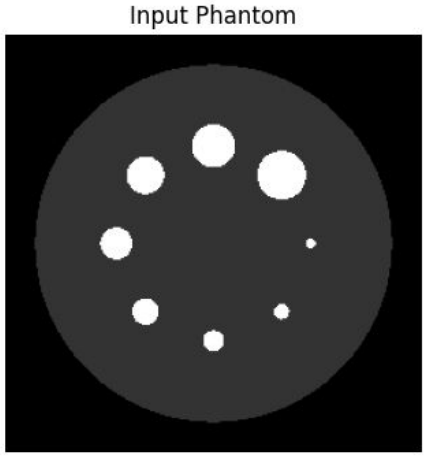
- Final reconstructed depth signal after FT

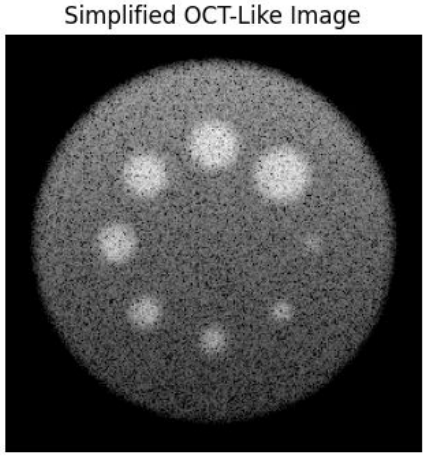
- Peaks = reflectors
  - Position = depth
  - Height = reflectivity

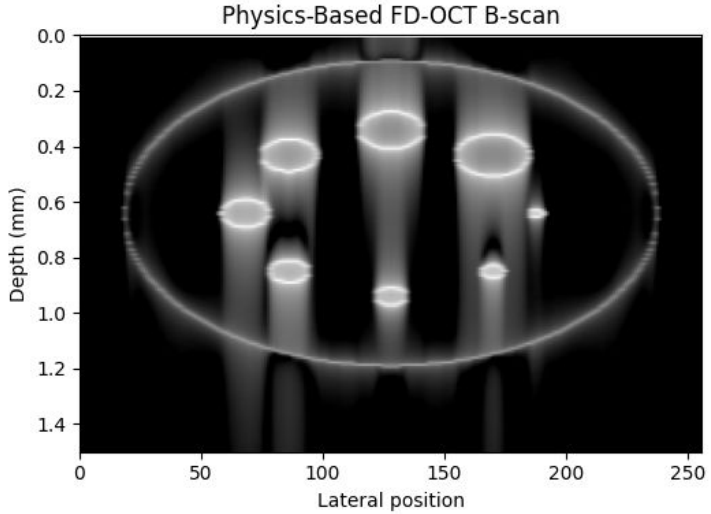


02
Frequency Domain - OCT SIMULATION

We first used a simplified image-quality model to show blur, speckle, rolloff, and dynamic range effects. We then implemented a more physics-based FD-OCT model by generating spectral interference signals for each phantom column and Fourier transforming them into A-scans, which were stacked into a B-scan

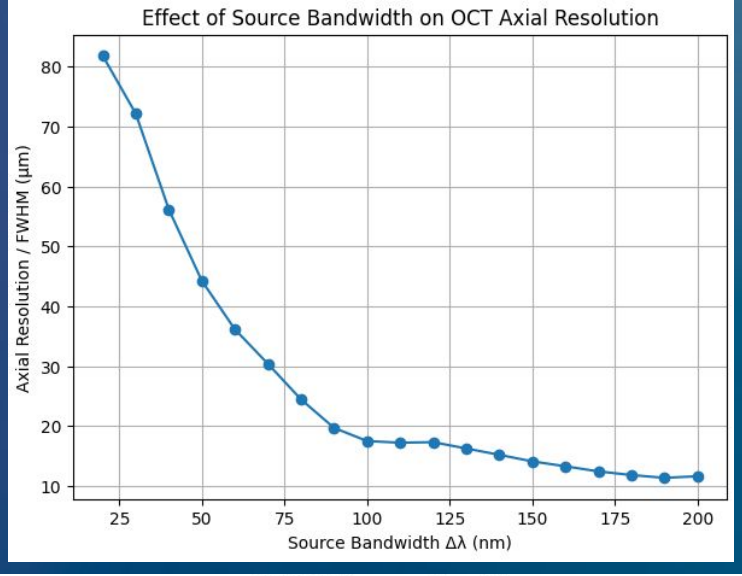


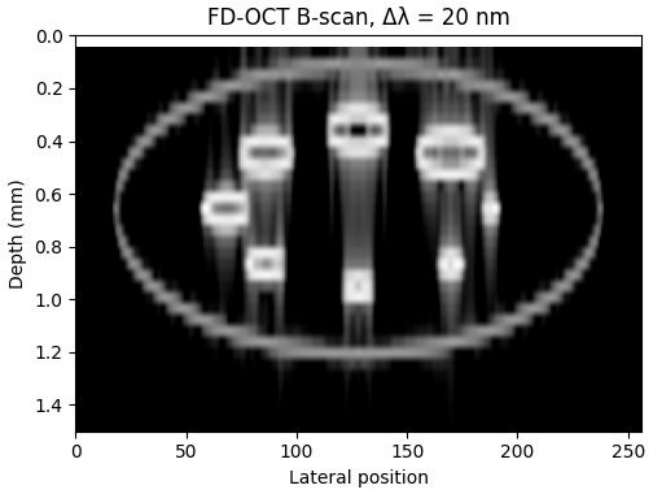


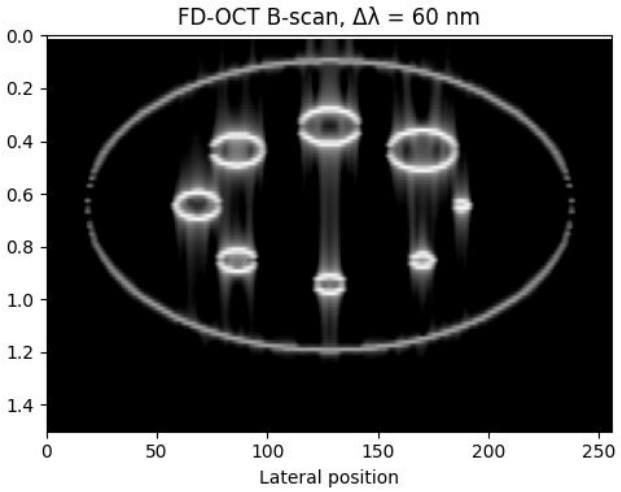


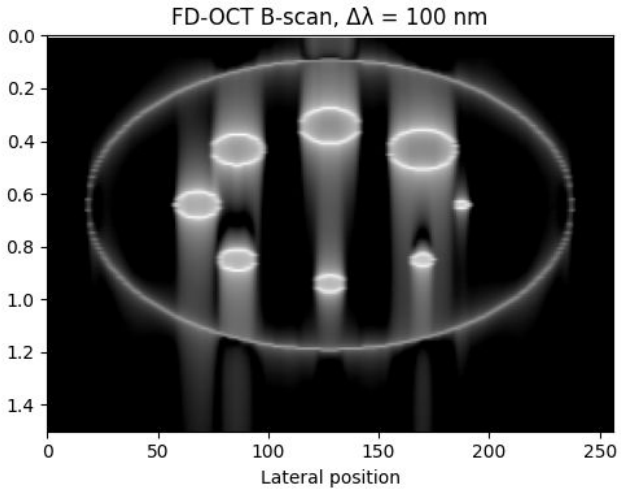
03

Frequency Domain - OCT RESOLUTION ANALYSIS


Effect of Source Bandwidth on OCT Axial Resolution
Axial Resolution / FWHM (μm)
Source Bandwidth Δλ (nm)
80
70
60
50
40
30
20
10
25
50
75
100
125
150
175
200

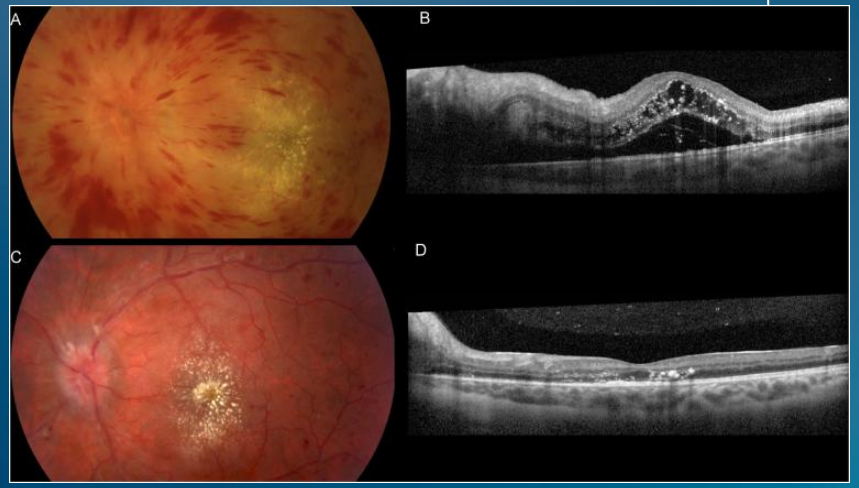

FD-OCT B-scan, Δλ = 20 nm
Depth (mm)
Lateral position
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
50
100
150
200
250


FD-OCT B-scan, Δλ = 60 nm
Lateral position
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
50
100
150
200
250


FD-OCT B-scan, Δλ = 100 nm
Lateral position
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
50
100
150
200
250

04 Clinical Applications

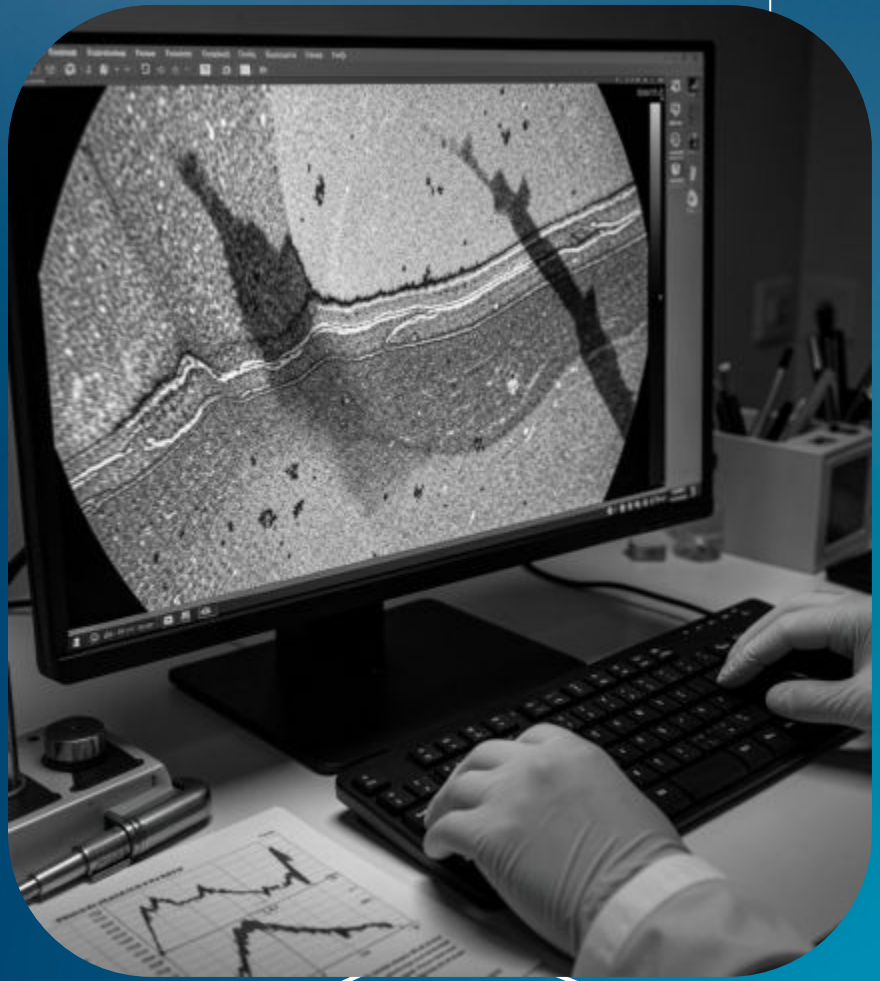
Retinal Imaging



- OCT provides high-resolution cross-sectional imaging of retinal layers
- Enables detection of early structural changes (e.g., drusen, layer disruption)
- Axial resolution is critical for distinguishing fine retinal features
- Motivates the importance of bandwidth and depth resolution in OCT systems

Artifacts & Limitations

- Speckle noise introduces grainy texture due to coherent imaging
- Signal rolloff reduces intensity at greater depths
- Motion and system effects can distort image structure
- These effects are approximated in the simplified OCT simulation
- Accurate interpretation requires understanding these artifacts



Conclusions

- Implemented both a simplified OCT model and a physics-based FD-OCT simulation
- FD-OCT reconstructs depth using Fourier transform of spectral interference
- Simplified model approximates image quality effects (blur, speckle, rolloff)
- Increasing source bandwidth improves axial resolution
- Demonstrated both quantitatively (FWHM) and visually (B-scan sharpening)