**符昕宇 202364810311**

**To access my code more easily, you can find them at my github repository:**

**https://github.com/Fluorine-Brian/Code-for-Digital-Image-Processing**

**Homework1: Implement Figure 4.40 by programming with Python**

**Python Code:**

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib

**import** matplotlib.pyplot as plt

**import** os

**def** pad\_image\_for\_dft(image):

    """

    Pads the image to a size suitable for DFT

    """

    M, N **=** image.shape

    P, Q **=** 2 **\*** M, 2 **\*** N

    padded\_image **=** np.zeros((P, Q), dtype**=**image.dtype)

    padded\_image[0:M, 0:N] **=** image

**return** padded\_image, (M, N)

**def** calculate\_power\_percentage(centered\_dft\_spectrum, D0):

    """

    Calculates the percentage of total power contained within a circle of radius D0

    """

    P, Q **=** centered\_dft\_spectrum.shape

    power\_spectrum **=** np.abs(centered\_dft\_spectrum) **\*\*** 2

    total\_power **=** np.sum(power\_spectrum)

    power\_within\_D0 **=** 0.0

    center\_u, center\_v **=** P **/** 2, Q **/** 2

    u\_coords **=** np.arange(P) **-** P **/** 2

    v\_coords **=** np.arange(Q) **-** Q **/** 2

    U, V **=** np.meshgrid(u\_coords, v\_coords, indexing**=**'ij')

    D\_uv **=** np.sqrt(U **\*\*** 2 **+** V **\*\*** 2)

    power\_within\_D0 **=** np.sum(power\_spectrum[D\_uv <**=** D0])

**if** total\_power > 0:

**return** (power\_within\_D0 **/** total\_power) **\*** 100

**else**:

**return** 0.0

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input\_dir **=** "original\_image"

    output\_dir **=** "output\_image"

    os.makedirs(output\_dir, exist\_ok**=**True)

    image\_filename **=** "Fig0440.tif"

    image\_path **=** os.path.join(input\_dir, image\_filename)

    base\_name **=** os.path.splitext(image\_filename)[0]

    original\_image **=** imageio.imread(image\_path)

    original\_M, original\_N **=** original\_image.shape

    padded\_image, \_ **=** pad\_image\_for\_dft(original\_image)

    P, Q **=** padded\_image.shape

    dft\_original **=** np.fft.fft2(padded\_image.astype(float))

    centered\_dft **=** np.fft.fftshift(dft\_original)

    spectrum\_log\_magnitude **=** 20 **\*** np.log(np.abs(centered\_dft) **+** 1e**-**9)

    spectrum\_display **=** (spectrum\_log\_magnitude **-** np.min(spectrum\_log\_magnitude)) **/**

                       (np.max(spectrum\_log\_magnitude) **-** np.min(spectrum\_log\_magnitude)) **\*** 255

    spectrum\_display **=** spectrum\_display.astype(np.uint8)

    D0\_values **=** [10, 30, 60, 160, 460]

    power\_percentages **=** {}

**for** D0 **in** D0\_values:

        power\_percent **=** calculate\_power\_percentage(centered\_dft, D0)

        power\_percentages[D0] **=** power\_percent

    plt.figure(figsize**=**(14, 7))

    plt.suptitle(f'Figure 4.40: Test Pattern and its Frequency Spectrum for {image\_filename}', fontsize**=**16)

    plt.subplot(1, 2, 1)

    plt.imshow(original\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('a) Test Pattern Image')

    plt.axis('off')

    plt.subplot(1, 2, 2)

    plt.imshow(spectrum\_display, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('b) Frequency Spectrum with D0 Circles')

    plt.axis('off')

    center\_x, center\_y **=** Q **/** 2, P **/** 2

**for** D0 **in** D0\_values:

        circle **=** plt.Circle((center\_x, center\_y), D0, color**=**'red', fill**=**False, linewidth**=**1, alpha**=**0.7)

        plt.gca().add\_patch(circle)

        text\_x **=** center\_x **+** D0 **\*** np.cos(np.deg2rad(45))center

        text\_y **=** center\_y **+** D0 **\*** np.sin(np.deg2rad(45))

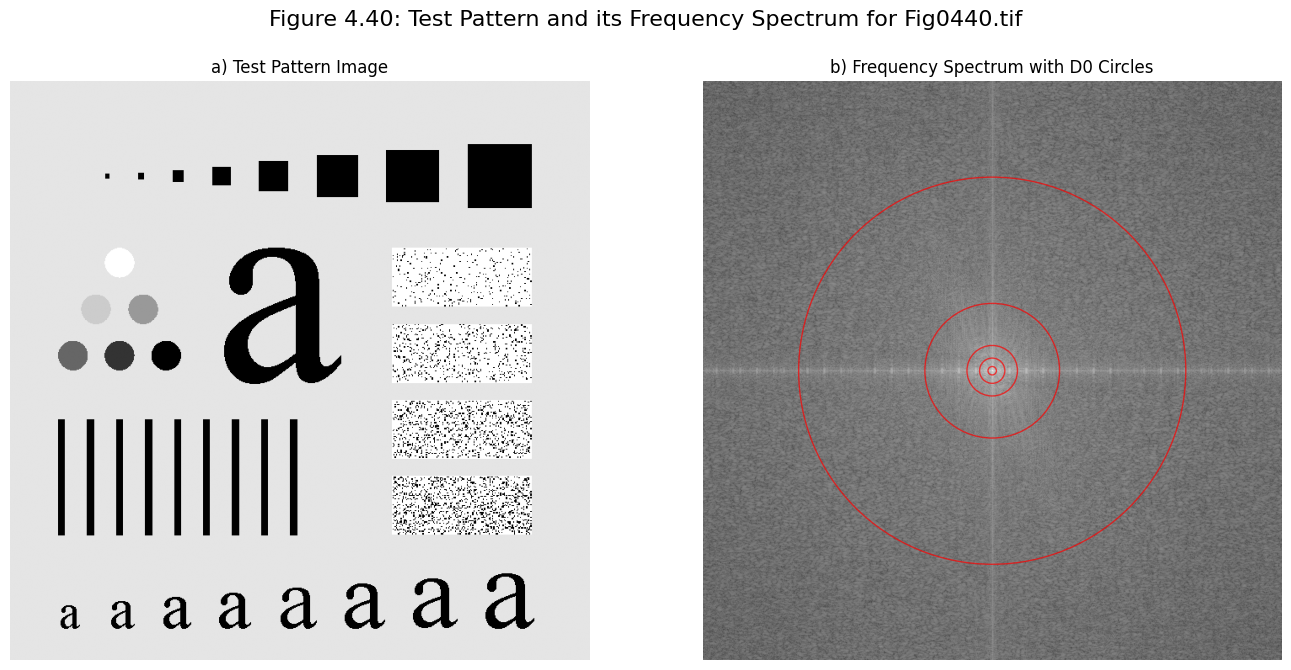
    plt.tight\_layout(rect**=**[0, 0.03, 1, 0.95])

    combined\_output\_path **=** os.path.join(output\_dir, f"{base\_name}\_fig4\_40\_reproduction.png")

    plt.savefig(combined\_output\_path, bbox\_inches**=**'tight')

    plt.close()

**Processed Images:**

****

**Homework 2: Implement Figure 4.41 by programming with Python**

**Python Code:**

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib.pyplot as plt

**import** os

**def** pad\_image\_for\_dft(image):

    """

    Pads the image to a size suitable for DFT (e.g., 2\*M x 2\*N)

    """

    M, N **=** image.shape

    P, Q **=** 2 **\*** M, 2 **\*** N  # Double the dimensions

    padded\_image **=** np.zeros((P, Q), dtype**=**image.dtype)

    padded\_image[0:M, 0:N] **=** image

**return** padded\_image, (M, N)

**def** calculate\_power\_percentage(centered\_dft\_spectrum, D0):

    """

    Calculates the percentage of total power contained within a circle of radius D0

    """

    P, Q **=** centered\_dft\_spectrum.shape

    power\_spectrum **=** np.abs(centered\_dft\_spectrum) **\*\*** 2

    total\_power **=** np.sum(power\_spectrum)

    power\_within\_D0 **=** 0.0

    center\_u, center\_v **=** P **/** 2, Q **/** 2

    u\_coords **=** np.arange(P) **-** P **/** 2

    v\_coords **=** np.arange(Q) **-** Q **/** 2

    U, V **=** np.meshgrid(u\_coords, v\_coords, indexing**=**'ij')

    D\_uv **=** np.sqrt(U **\*\*** 2 **+** V **\*\*** 2)

    power\_within\_D0 **=** np.sum(power\_spectrum[D\_uv <**=** D0])

**if** total\_power > 0:

**return** (power\_within\_D0 **/** total\_power) **\*** 100

**else**:

**return** 0.0

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input\_dir **=** "original\_image"

    output\_dir **=** "output\_image"

    os.makedirs(output\_dir, exist\_ok**=**True)

    image\_filename **=** "Fig0440.tif"

    image\_path **=** os.path.join(input\_dir, image\_filename)

    base\_name **=** os.path.splitext(image\_filename)[0]

    original\_image **=** imageio.imread(image\_path)

    original\_M, original\_N **=** original\_image.shape

    padded\_image, \_ **=** pad\_image\_for\_dft(original\_image)

    P, Q **=** padded\_image.shape

    dft\_original **=** np.fft.fft2(padded\_image.astype(float))

    centered\_dft **=** np.fft.fftshift(dft\_original)

    spectrum\_log\_magnitude **=** 20 **\*** np.log(np.abs(centered\_dft) **+** 1e**-**9)

    spectrum\_display **=** (spectrum\_log\_magnitude **-** np.min(spectrum\_log\_magnitude)) **/** \

                       (np.max(spectrum\_log\_magnitude) **-** np.min(spectrum\_log\_magnitude)) **\*** 255

    spectrum\_display **=** spectrum\_display.astype(np.uint8)

    D0\_values **=** [10, 30, 60, 160, 460]

    power\_percentages **=** {}

**for** D0 **in** D0\_values:

        power\_percent **=** calculate\_power\_percentage(centered\_dft, D0)

        power\_percentages[D0] **=** power\_percent

    plt.figure(figsize**=**(14, 7))

    plt.suptitle(f'Figure 4.40: Test Pattern and its Frequency Spectrum for {image\_filename}', fontsize**=**16)

    plt.subplot(1, 2, 1)

    plt.imshow(original\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('a) Test Pattern Image')

    plt.axis('off')

    plt.subplot(1, 2, 2)

    plt.imshow(spectrum\_display, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('b) Frequency Spectrum with D0 Circles')

    plt.axis('off')

    center\_x, center\_y **=** Q **/** 2, P **/** 2

**for** D0 **in** D0\_values:

        circle **=** plt.Circle((center\_x, center\_y), D0, color**=**'red', fill**=**False, linewidth**=**1, alpha**=**0.7)

        plt.gca().add\_patch(circle)

        text\_x **=** center\_x **+** D0 **\*** np.cos(np.deg2rad(45))

        text\_y **=** center\_y **+** D0 **\*** np.sin(np.deg2rad(45))

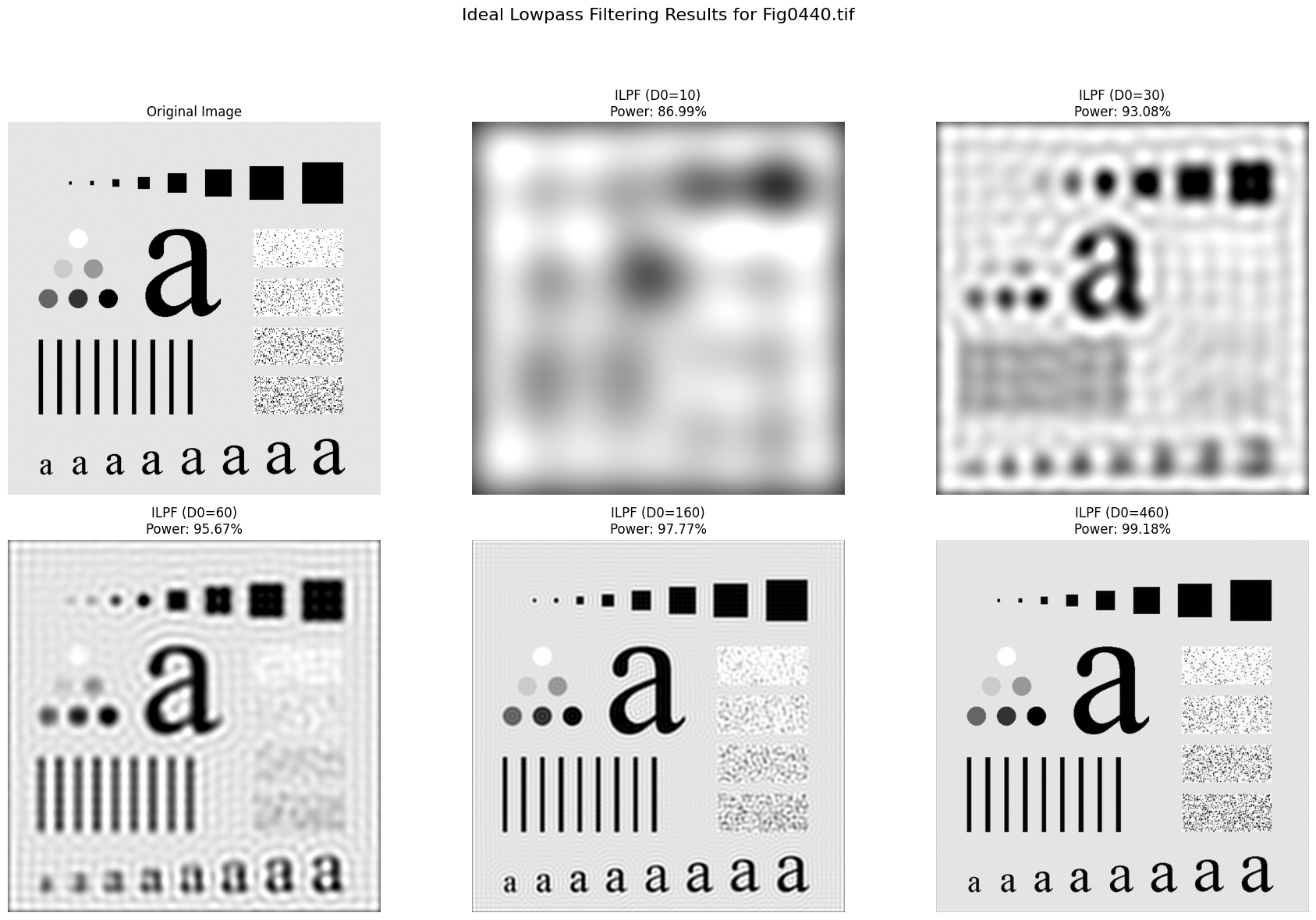
    plt.tight\_layout(rect**=**[0, 0.03, 1, 0.95])

    combined\_output\_path **=** os.path.join(output\_dir, f"{base\_name}\_fig4\_40\_reproduction.png")

    plt.savefig(combined\_output\_path, bbox\_inches**=**'tight')

    plt.close()

**Processed Images:**

****

**Homework3: Implement Figure 4.44 by programming with Python**

**Python Code:**

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib

**import** matplotlib.pyplot as plt

**import** os

**def** pad\_image\_for\_dft(image, padding\_mode**=**'reflect'):

    """

    Pads the image to a size suitable for DFT (e.g., 2\*M x 2\*N)

    """

    M, N **=** image.shape

    P, Q **=** 2 **\*** M, 2 **\*** N

    pad\_h **=** P **-** M

    pad\_w **=** Q **-** N

    padded\_image **=** np.pad(image, ((0, pad\_h), (0, pad\_w)), mode**=**padding\_mode)

**return** padded\_image, (M, N)

**def** create\_gaussian\_lowpass\_filter(shape, D0):

    """

    Creates a Gaussian Lowpass Filter (GLPF) in the frequency domain

    """

    P, Q **=** shape

    H **=** np.zeros((P, Q), dtype**=**float)

    center\_u, center\_v **=** P **/** 2, Q **/** 2

    u\_coords **=** np.arange(P) **-** center\_u

    v\_coords **=** np.arange(Q) **-** center\_v

    U, V **=** np.meshgrid(u\_coords, v\_coords, indexing**=**'ij')

    D\_uv\_squared **=** U **\*\*** 2 **+** V **\*\*** 2

**if** D0 **==** 0:

        H **=** np.ones(shape, dtype**=**float)

**else**:

        H **=** np.exp(**-**D\_uv\_squared **/** (2 **\*** D0 **\*\*** 2))

**return** H

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input\_dir **=** "original\_image"

    output\_dir **=** "output\_image"

    os.makedirs(output\_dir, exist\_ok**=**True)

    image\_filename **=** "Fig0440.tif"

    image\_path **=** os.path.join(input\_dir, image\_filename)

    base\_name **=** os.path.splitext(image\_filename)[0]

    original\_image **=** imageio.imread(image\_path)

    original\_M, original\_N **=** original\_image.shape

    padded\_image, \_ **=** pad\_image\_for\_dft(original\_image, padding\_mode**=**'reflect')

    P, Q **=** padded\_image.shape

    dft\_original **=** np.fft.fft2(padded\_image.astype(float))

    centered\_dft **=** np.fft.fftshift(dft\_original)

    D0\_values **=** [10, 30, 60, 160, 460]

    filtered\_images **=** {}

**for** D0 **in** D0\_values:

        H **=** create\_gaussian\_lowpass\_filter((P, Q), D0)

        filtered\_dft **=** centered\_dft **\*** H

        idft\_shifted **=** np.fft.ifftshift(filtered\_dft)

        filtered\_image\_complex **=** np.fft.ifft2(idft\_shifted)

        filtered\_image **=** np.real(filtered\_image\_complex)[0:original\_M, 0:original\_N]

        filtered\_image **=** np.clip(filtered\_image, 0, 255).astype(np.uint8)

        filtered\_images[D0] **=** filtered\_image

    plt.figure(figsize**=**(18, 12))

    plt.suptitle(f'Gaussian Lowpass Filtering Results for {image\_filename}', fontsize**=**16)

    plt.subplot(2, 3, 1)

    plt.imshow(original\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('a) Original Image')

    plt.axis('off')

    subplot\_labels **=** ['b', 'c', 'd', 'e', 'f']

**for** i, D0 **in** enumerate(D0\_values):

        plt.subplot(2, 3, i **+** 2)

        plt.imshow(filtered\_images[D0], cmap**=**'gray', vmin**=**0, vmax**=**255)

        plt.title(f'{subplot\_labels[i]}) GLPF (D0={D0})')

        plt.axis('off')

    plt.tight\_layout(rect**=**[0, 0.03, 1, 0.95])

    combined\_output\_path **=** os.path.join(output\_dir, f"{base\_name}\_glpf\_results\_combined.png")

    plt.savefig(combined\_output\_path, bbox\_inches**=**'tight')

    plt.close()

**Processed Images:**

****

**Homework4: Implement Figure 4.46 by programming with Python**

**Python Code:**

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib

**import** matplotlib.pyplot as plt

**import** os

**def** pad\_image\_for\_dft(image, padding\_mode**=**'reflect'):

    """

    Pads the image to a size suitable for DFT (e.g., 2\*M x 2\*N)

    """

    M, N **=** image.shape

    P, Q **=** 2 **\*** M, 2 **\*** N

    pad\_h **=** P **-** M

    pad\_w **=** Q **-** N

    padded\_image **=** np.pad(image, ((0, pad\_h), (0, pad\_w)), mode**=**padding\_mode)

**return** padded\_image, (M, N)

**def** create\_butterworth\_lowpass\_filter(shape, D0, n):

    """

    Creates a Butterworth Lowpass Filter (BLPF) in the frequency domain

    """

    P, Q **=** shape

    H **=** np.zeros((P, Q), dtype**=**float)

    center\_u, center\_v **=** P **/** 2, Q **/** 2

    u\_coords **=** np.arange(P) **-** center\_u

    v\_coords **=** np.arange(Q) **-** center\_v

    U, V **=** np.meshgrid(u\_coords, v\_coords, indexing**=**'ij')

    D\_uv **=** np.sqrt(U **\*\*** 2 **+** V **\*\*** 2)

**if** D0 **==** 0:

        H **=** np.ones(shape, dtype**=**float)  # Pass all frequencies if D0 is 0

**else**:

        H **=** 1 **/** (1 **+** (D\_uv **/** D0) **\*\*** (2 **\*** n))

        H[int(center\_u), int(center\_v)] **=** 1.0

**return** H

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input\_dir **=** "original\_image"

    output\_dir **=** "output\_image"

    os.makedirs(output\_dir, exist\_ok**=**True)

    image\_filename **=** "Fig0440.tif"

    image\_path **=** os.path.join(input\_dir, image\_filename)

    base\_name **=** os.path.splitext(image\_filename)[0]

    original\_image

    original\_M, original\_N **=** original\_image.shape

    padded\_image, \_ **=** pad\_image\_for\_dft(original\_image, padding\_mode**=**'reflect')

    P, Q **=** padded\_image.shape

    dft\_original **=** np.fft.fft2(padded\_image.astype(float))

    centered\_dft **=** np.fft.fftshift(dft\_original)

    D0\_values **=** [10, 30, 60, 160, 460]

    n\_order **=** 2.25

    filtered\_images **=** {}

**for** D0 **in** D0\_values:

        H **=** create\_butterworth\_lowpass\_filter((P, Q), D0, n\_order)

        filtered\_dft **=** centered\_dft **\*** H

        idft\_shifted **=** np.fft.ifftshift(filtered\_dft)

        filtered\_image\_complex **=** np.fft.ifft2(idft\_shifted)

        filtered\_image **=** np.real(filtered\_image\_complex)[0:original\_M, 0:original\_N]

        filtered\_image **=** np.clip(filtered\_image, 0, 255).astype(np.uint8)

        filtered\_images[D0] **=** filtered\_image

    plt.figure(figsize**=**(18, 12))

    plt.suptitle(f'Butterworth Lowpass Filtering Results for {image\_filename} (n={n\_order})', fontsize**=**16)

    plt.subplot(2, 3, 1)

    plt.imshow(original\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('a) Original Image')

    plt.axis('off')

    subplot\_labels **=** ['b', 'c', 'd', 'e', 'f']

**for** i, D0 **in** enumerate(D0\_values):

        plt.subplot(2, 3, i **+** 2)

        plt.imshow(filtered\_images[D0], cmap**=**'gray', vmin**=**0, vmax**=**255)

        plt.title(f'{subplot\_labels[i]}) BLPF (D0={D0}, n={n\_order})')

        plt.axis('off')

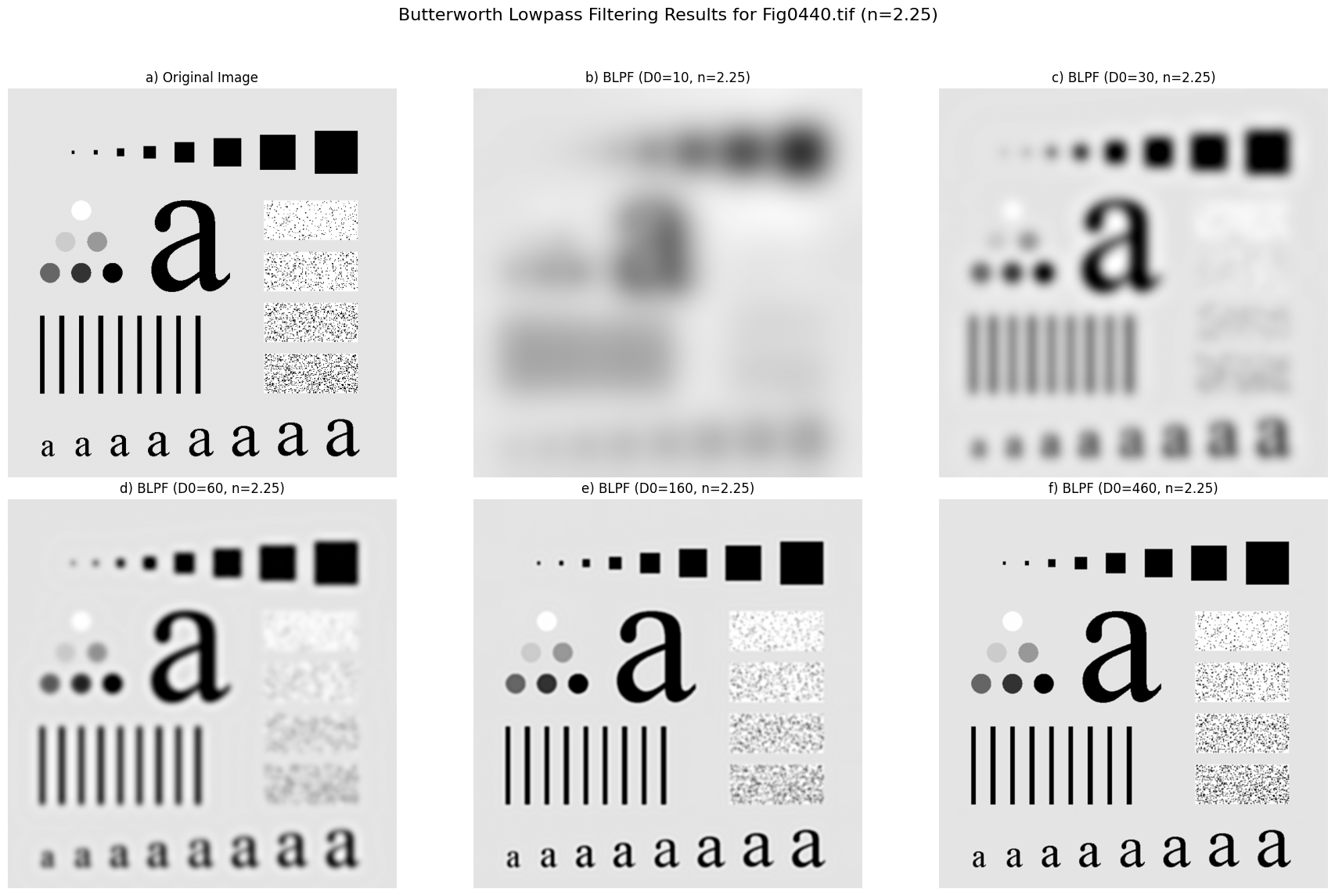
    plt.tight\_layout(rect**=**[0, 0.03, 1, 0.95])

    combined\_output\_path **=** os.path.join(output\_dir, f"{base\_name}\_blpf\_results\_combined\_n{n\_order}.png")

    plt.savefig(combined\_output\_path, bbox\_inches**=**'tight')

    plt.close()

**Processed Image**

****

**Homework5: Implement Figure 4.48 by programming with Python**

**Python Code:**

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib.pyplot as plt

**import** os

**def** pad\_image\_for\_dft(image, padding\_mode**=**'reflect'):

    """

    Pads the image to a size suitable for DFT (e.g., 2\*M x 2\*N)

    """

    M, N **=** image.shape

    P, Q **=** 2 **\*** M, 2 **\*** N

    pad\_h **=** P **-** M

    pad\_w **=** Q **-** N

    padded\_image **=** np.pad(image, ((0, pad\_h), (0, pad\_w)), mode**=**padding\_mode)

**return** padded\_image, (M, N)

**def** create\_gaussian\_lowpass\_filter(shape, D0):

    """

    Creates a Gaussian Lowpass Filter (GLPF) in the frequency domain

    """

    P, Q **=** shape

    H **=** np.zeros((P, Q), dtype**=**float)

    center\_u, center\_v **=** P **/** 2, Q **/** 2

    u\_coords **=** np.arange(P) **-** center\_u

    v\_coords **=** np.arange(Q) **-** center\_v

    U, V **=** np.meshgrid(u\_coords, v\_coords, indexing**=**'ij')

    D\_uv\_squared **=** U **\*\*** 2 **+** V **\*\*** 2

**if** D0 **==** 0:

        H **=** np.ones(shape, dtype**=**float)

**else**:

        H **=** np.exp(**-**D\_uv\_squared **/** (2 **\*** D0 **\*\*** 2))

**return** H

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input\_dir **=** "original\_image"

    output\_dir **=** "output\_image"

    os.makedirs(output\_dir, exist\_ok**=**True)

    image\_filename **=** "Fig0419(a)(text\_gaps\_of\_1\_and\_2\_pixels).tif"

    image\_path **=** os.path.join(input\_dir, image\_filename)

    base\_name **=** os.path.splitext(image\_filename)[0]

    original\_image **=** imageio.imread(image\_path)

    original\_M, original\_N **=** original\_image.shape

    padded\_image, \_ **=** pad\_image\_for\_dft(original\_image, padding\_mode**=**'reflect')

    P, Q **=** padded\_image.shape

    dft\_original **=** np.fft.fft2(padded\_image.astype(float))

    centered\_dft **=** np.fft.fftshift(dft\_original)

    D0\_value **=** 120

    H **=** create\_gaussian\_lowpass\_filter((P, Q), D0\_value)

    filtered\_dft **=** centered\_dft **\*** H

    idft\_shifted **=** np.fft.ifftshift(filtered\_dft)

    filtered\_image\_complex **=** np.fft.ifft2(idft\_shifted)

    filtered\_image **=** np.real(filtered\_image\_complex)[0:original\_M, 0:original\_N]

    filtered\_image **=** np.clip(filtered\_image, 0, 255).astype(np.uint8)

    print(f"Finished GLPF with D0={D0\_value}.")

    plt.figure(figsize**=**(12, 6))

    plt.suptitle(f'Text Gap Repair using GLPF for {image\_filename}', fontsize**=**16)

    plt.subplot(1, 2, 1)

    plt.imshow(original\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('a) Original Text with Gaps')

    plt.axis('off')

    plt.subplot(1, 2, 2)

    plt.imshow(filtered\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title(f'b) GLPF Filtered (D0={D0\_value})')

    plt.axis('off')

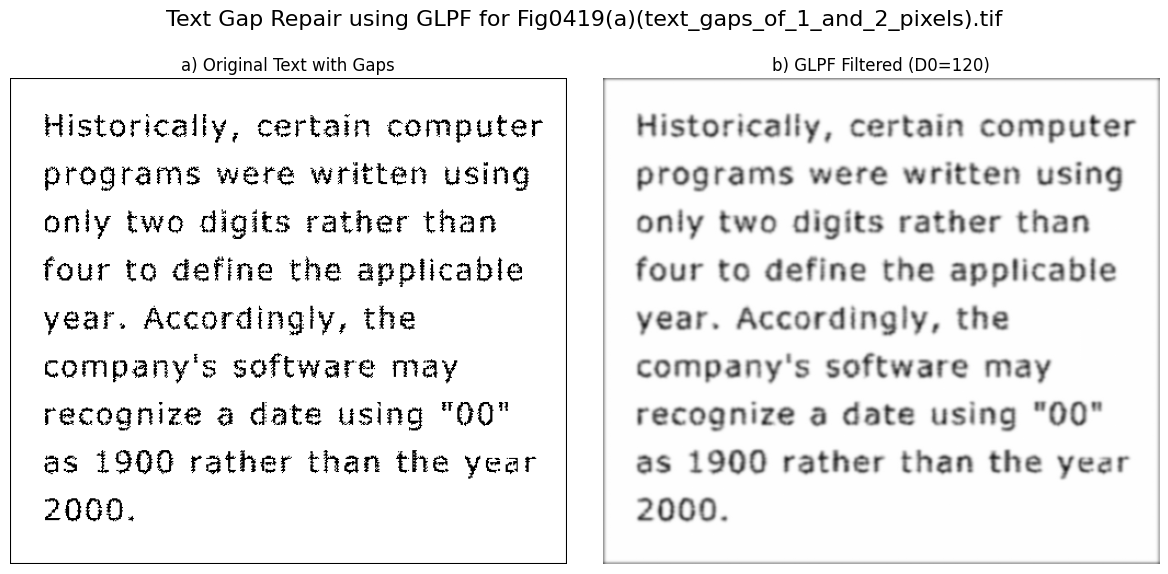
    plt.tight\_layout(rect**=**[0, 0.03, 1, 0.95])

    combined\_output\_path **=** os.path.join(output\_dir, f"{base\_name}\_glpf\_text\_repair\_results.png")

    plt.savefig(combined\_output\_path, bbox\_inches**=**'tight')

    plt.close()

**Processed Images:**

****

****

**Homework6: Implement Figure 4.49 by programming with Python**

**Python Code:**

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib.pyplot as plt

**import** os

**def** pad\_image\_for\_dft(image, padding\_mode**=**'reflect'):

    """

    Pads the image to a size suitable for DFT (e.g., 2\*M x 2\*N)

    """

    M, N **=** image.shape

    P, Q **=** 2 **\*** M, 2 **\*** N

    pad\_h **=** P **-** M

    pad\_w **=** Q **-** N

    padded\_image **=** np.pad(image, ((0, pad\_h), (0, pad\_w)), mode**=**padding\_mode)

**return** padded\_image, (M, N)

**def** create\_gaussian\_lowpass\_filter(shape, D0):

    """

    Creates a Gaussian Lowpass Filter (GLPF) in the frequency domain

    """

    P, Q **=** shape

    H **=** np.zeros((P, Q), dtype**=**float)

    center\_u, center\_v **=** P **/** 2, Q **/** 2

    u\_coords **=** np.arange(P) **-** center\_u

    v\_coords **=** np.arange(Q) **-** center\_v

    U, V **=** np.meshgrid(u\_coords, v\_coords, indexing**=**'ij')

    D\_uv\_squared **=** U **\*\*** 2 **+** V **\*\*** 2

**if** D0 **==** 0:

        H **=** np.ones(shape, dtype**=**float)

**else**:

        H **=** np.exp(**-**D\_uv\_squared **/** (2 **\*** D0 **\*\*** 2))

**return** H

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input\_dir **=** "original\_image"

    output\_dir **=** "output\_image"

    os.makedirs(output\_dir, exist\_ok**=**True)exist

    image\_filename **=** "Fig0450(a)(woman\_original).tif"

    image\_path **=** os.path.join(input\_dir, image\_filename)

    base\_name **=** os.path.splitext(image\_filename)[0]

    original\_image **=** imageio.imread(image\_path)

    original\_M, original\_N **=** original\_image.shape

    padded\_image, \_ **=** pad\_image\_for\_dft(original\_image, padding\_mode**=**'reflect')

    P, Q **=** padded\_image.shape

    dft\_original **=** np.fft.fft2(padded\_image.astype(float))

    centered\_dft **=** np.fft.fftshift(dft\_original)

    D0\_values **=** [150, 130]

    filtered\_images **=** {}

**for** D0 **in** D0\_values:

        H **=** create\_gaussian\_lowpass\_filter((P, Q), D0)

        filtered\_dft **=** centered\_dft **\*** H

        idft\_shifted **=** np.fft.ifftshift(filtered\_dft)

        filtered\_image\_complex **=** np.fft.ifft2(idft\_shifted)

        filtered\_image **=** np.real(filtered\_image\_complex)[0:original\_M, 0:original\_N]

        filtered\_image **=** np.clip(filtered\_image, 0, 255).astype(np.uint8)

        filtered\_images[D0] **=** filtered\_image

    plt.figure(figsize**=**(18, 6))

    plt.suptitle(f'Image Beautification using GLPF for {image\_filename}', fontsize**=**16)

    plt.subplot(1, 3, 1)

    plt.imshow(original\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('a) Original Image')

    plt.axis('off')

    plt.subplot(1, 3, 2)

    plt.imshow(filtered\_images[150], cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title(f'b) GLPF Filtered (D0=150)')

    plt.axis('off')

    plt.subplot(1, 3, 3)

    plt.imshow(filtered\_images[130], cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title(f'c) GLPF Filtered (D0=130)')

    plt.axis('off')

    plt.tight\_layout(rect**=**[0, 0.03, 1, 0.95])

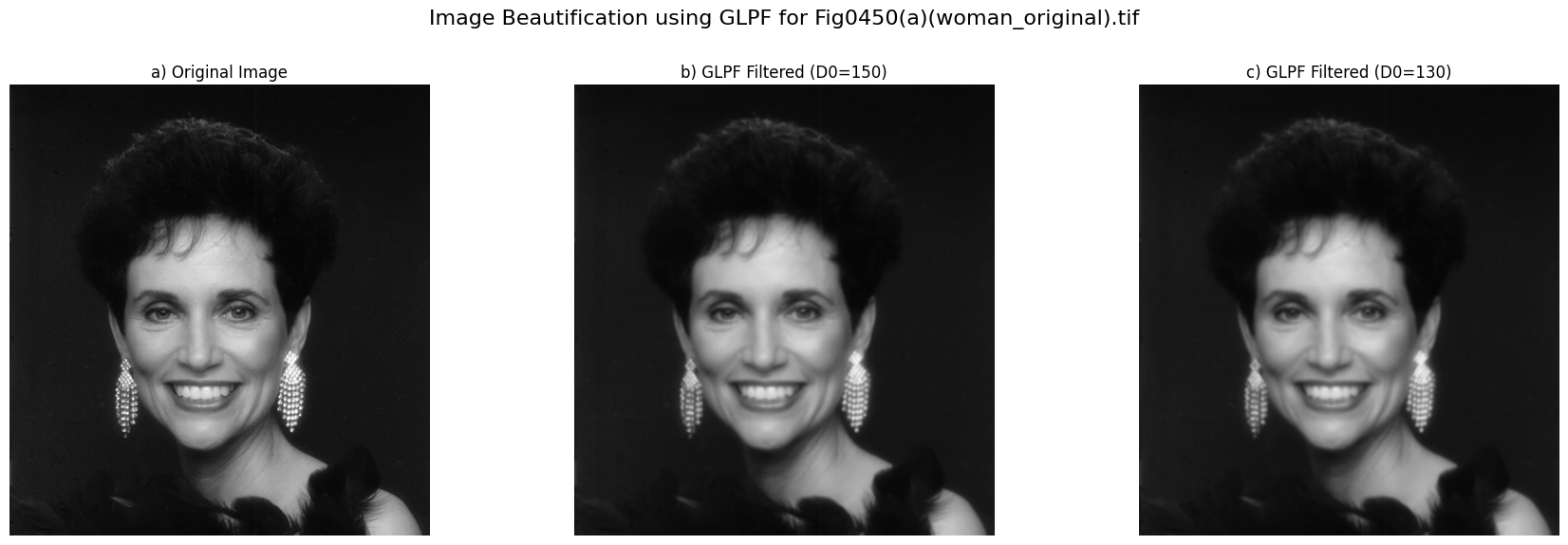
    combined\_output\_path **=** os.path.join(output\_dir, f"{base\_name}\_glpf\_beautification\_results.png")

    plt.savefig(combined\_output\_path, bbox\_inches**=**'tight')

    print(f"Combined visualization saved to: {combined\_output\_path}")

    plt.close()

**Processed Images:**

****

**Homework7: Implement Figure 4.50 by programming with Python**

**Python Code:**

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib.pyplot as plt

**import** os

**def** pad\_image\_for\_dft(image, padding\_mode**=**'reflect'):

    """Pads the image to a size suitable for DFT (e.g., 2\*M x 2\*N)"""

    M, N **=** image.shape

    P, Q **=** 2 **\*** M, 2 **\*** N

    pad\_h **=** P **-** M

    pad\_w **=** Q **-** N

    padded\_image **=** np.pad(image, ((0, pad\_h), (0, pad\_w)), mode**=**padding\_mode)

**return** padded\_image, (M, N)

**def** create\_gaussian\_lowpass\_filter(shape, D0):

    """Creates a Gaussian Lowpass Filter (GLPF) in the frequency domain"""

    P, Q **=** shape

    H **=** np.zeros((P, Q), dtype**=**float)

    center\_u, center\_v **=** P **/** 2, Q **/** 2

    u\_coords **=** np.arange(P) **-** center\_u

    v\_coords **=** np.arange(Q) **-** center\_v

    U, V **=** np.meshgrid(u\_coords, v\_coords, indexing**=**'ij')

    D\_uv\_squared **=** U **\*\*** 2 **+** V **\*\*** 2

**if** D0 **==** 0:

        H **=** np.ones(shape, dtype**=**float)

**else**:

        H **=** np.exp(**-**D\_uv\_squared **/** (2 **\*** D0 **\*\*** 2))

**return** H

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input\_dir **=** "original\_image"

    output\_dir **=** "output\_image"

    os.makedirs(output\_dir, exist\_ok**=**True)

    image\_filename **=** "Fig0451(a)(satellite\_original).tif"

    image\_path **=** os.path.join(input\_dir, image\_filename)

    base\_name **=** os.path.splitext(image\_filename)[0]

    original\_image **=** imageio.imread(image\_path)

    original\_M, original\_N **=** original\_image.shape

    padded\_image, \_ **=** pad\_image\_for\_dft(original\_image, padding\_mode**=**'reflect')

    P, Q **=** padded\_image.shape

    dft\_original **=** np.fft.fft2(padded\_image.astype(float))

    centered\_dft **=** np.fft.fftshift(dft\_original)

    D0\_values **=** [50, 20]

    filtered\_images **=** {}

**for** D0 **in** D0\_values:

        H **=** create\_gaussian\_lowpass\_filter((P, Q), D0)

        filtered\_dft **=** centered\_dft **\*** H

        idft\_shifted **=** np.fft.ifftshift(filtered\_dft)

        filtered\_image\_complex **=** np.fft.ifft2(idft\_shifted)

        filtered\_image **=** np.real(filtered\_image\_complex)[0:original\_M, 0:original\_N]

        filtered\_image **=** np.clip(filtered\_image, 0, 255).astype(np.uint8)

        filtered\_images[D0] **=** filtered\_image

    plt.figure(figsize**=**(18, 6))

    plt.suptitle(f'GLPF for Satellite Image (Fig 4.50) - {image\_filename}', fontsize**=**16)

    plt.subplot(1, 3, 1)

    plt.imshow(original\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('a) Original Satellite Image')

    plt.axis('off')

    plt.subplot(1, 3, 2)

    plt.imshow(filtered\_images[50], cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title(f'b) GLPF Filtered (D0=50)')

    plt.axis('off')

    plt.subplot(1, 3, 3)

    plt.imshow(filtered\_images[20], cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title(f'c) GLPF Filtered (D0=20)')

    plt.axis('off')

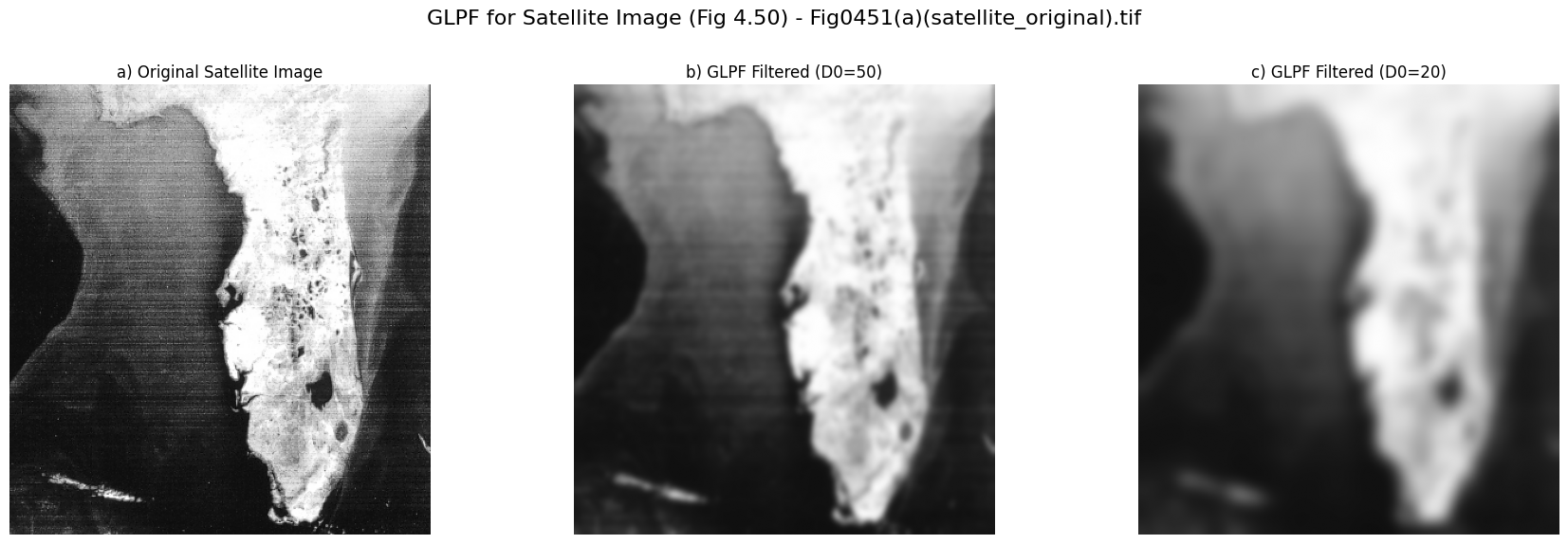
    plt.tight\_layout(rect**=**[0, 0.03, 1, 0.95])

    combined\_output\_path **=** os.path.join(output\_dir, f"{base\_name}\_glpf\_satellite\_results.png")

    plt.savefig(combined\_output\_path, bbox\_inches**=**'tight')

    plt.close()

**Processed Images:**

****