**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 9.5 by programming with Python**

**Python Code:**

**import** numpy as np

**import** cv2

**import** matplotlib.pyplot as plt

**import** os

**from** scipy **import** ndimage

MANUAL\_INVERT **=** False

**def** perform\_erosion(binary\_image, size):

    """

    执行形态学腐蚀

    """

    structure **=** np.ones((size, size), dtype**=**bool)

    eroded\_mask **=** ndimage.binary\_erosion(binary\_image, structure**=**structure)

**return** eroded\_mask.astype(np.uint8) **\*** 255

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

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

    output\_dir **=** "output\_image"

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

    image\_filename **=** "Fig0905(a)(wirebond-mask).tif"

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

    raw\_image **=** cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

    binary\_image **=** raw\_image > 127

**if** MANUAL\_INVERT:

        binary\_image **=** ~binary\_image

    foreground\_pixels **=** np.sum(binary\_image)

    total\_pixels **=** binary\_image.size

    ratio **=** foreground\_pixels **/** total\_pixels

    # 11x11

    image\_b **=** perform\_erosion(binary\_image, 11)

    # 15x15

    image\_c **=** perform\_erosion(binary\_image, 15)

    # 45x45

    image\_d **=** perform\_erosion(binary\_image, 45)

    fig, axes **=** plt.subplots(2, 2, figsize**=**(10, 10))

    fig.suptitle(f'Morphological Erosion (Fig 9.5) - {image\_filename}', fontsize**=**16)

    display\_original **=** binary\_image.astype(np.uint8) **\*** 255

    axes[0, 0].imshow(display\_original, cmap**=**'gray', vmin**=**0, vmax**=**255)

    axes[0, 0].set\_title('a) Original Image (White=Object)')

    axes[0, 1].imshow(image\_b, cmap**=**'gray', vmin**=**0, vmax**=**255)

    axes[0, 1].set\_title('b) Erosion with 11x11 SE')

    axes[1, 0].imshow(image\_c, cmap**=**'gray', vmin**=**0, vmax**=**255)

    axes[1, 0].set\_title('c) Erosion with 15x15 SE')

    axes[1, 1].imshow(image\_d, cmap**=**'gray', vmin**=**0, vmax**=**255)

    axes[1, 1].set\_title('d) Erosion with 45x45 SE')

**for** ax **in** axes.flat:

        ax.axis('off')

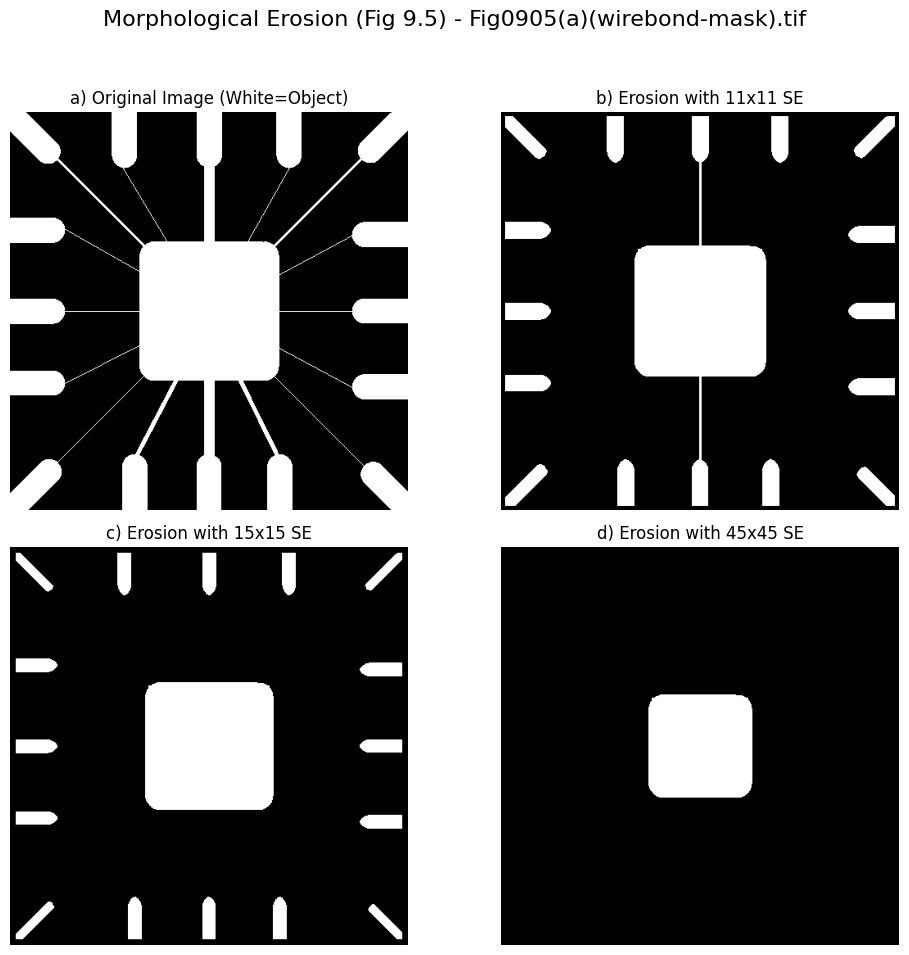
    plt.tight\_layout(rect**=**[0, 0.03, 1, 0.95])

    save\_path **=** os.path.join(output\_dir, f"{os.path.splitext(image\_filename)[0]}\_final\_fix.png")

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

    plt.close()

**Processed Images:**



**Homework2: Implement Figure 9.7 by programming with Python**

**Python Code:**

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib.pyplot as plt

**import** os

**from** scipy **import** ndimage

**def** perform\_dilation(binary\_image, size):

    """

    执行形态学膨胀操作

    """

    structure **=** np.ones((size, size), dtype**=**bool)

    dilated\_mask **=** ndimage.binary\_dilation(binary\_image, structure**=**structure)

**return** dilated\_mask.astype(np.uint8) **\*** 255

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

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

    output\_dir **=** "output\_image"

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

    image\_filename **=** "Fig0907(a)(text\_gaps\_1\_and\_2\_pixels).tif"

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

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

    binary\_image **=** raw\_image.astype(bool)

    size\_se **=** 3

    image\_c **=** perform\_dilation(binary\_image, size\_se)

    fig, axes **=** plt.subplots(1, 2, figsize**=**(12, 6))

    fig.suptitle(f'Morphological Dilation (Fig 9.7) - {image\_filename}', fontsize**=**16)

    display\_original **=** binary\_image.astype(np.uint8) **\*** 255

    axes[0].imshow(display\_original, cmap**=**'gray', vmin**=**0, vmax**=**255)

    axes[0].set\_title('a) Original Image (Broken Text)')

    axes[1].imshow(image\_c, cmap**=**'gray', vmin**=**0, vmax**=**255)

    axes[1].set\_title(f'c) Dilation with {size\_se}x{size\_se} SE (Repaired Text)')

**for** ax **in** axes.flat:

        ax.axis('off')

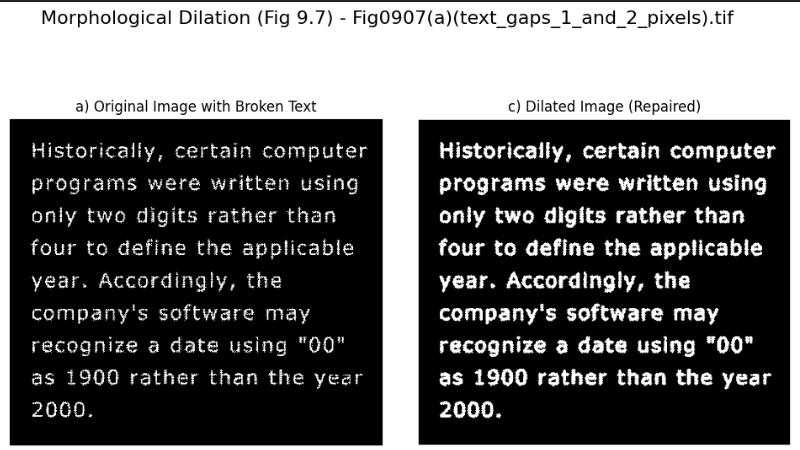
    plt.tight\_layout(rect**=**[0, 0.03, 1, 0.95])

    combined\_output\_path **=** os.path.join(output\_dir, f"{os.path.splitext(image\_filename)[0]}\_dilation\_corrected.png")

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

plt.close()

**Processed Images:**



**Homework3: Implement Figure 9.11 by programming with Python**

**Python Code:**

**import** numpy as np

**import** matplotlib.pyplot as plt

**import** os

**from** scipy **import** ndimage

**from** PIL **import** Image

**def** perform\_erosion(binary\_image, structure):

    """ Performs Morphological Erosion """

**return** ndimage.binary\_erosion(binary\_image, structure**=**structure)

**def** perform\_dilation(binary\_image, structure):

    """ Performs Morphological Dilation """

**return** ndimage.binary\_dilation(binary\_image, structure**=**structure)

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

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

    output\_dir **=** "output\_image"

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

    image\_filename **=** "Fig0911(a)(noisy\_fingerprint).tif"

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

    raw\_image\_pil **=** Image.open(image\_path).convert('L')

    raw\_image **=** np.array(raw\_image\_pil)

    binary\_image **=** raw\_image > 128

    structure\_size **=** 3

    structure **=** np.ones((structure\_size, structure\_size), dtype**=**bool)

    display\_original **=** binary\_image.astype(np.uint8) **\*** 255

    image\_c\_eroded\_bool **=** perform\_erosion(binary\_image, structure)

    image\_c **=** image\_c\_eroded\_bool.astype(np.uint8) **\*** 255

    image\_d\_opened\_bool **=** perform\_dilation(image\_c\_eroded\_bool, structure)

    image\_d **=** image\_d\_opened\_bool.astype(np.uint8) **\*** 255

    image\_e\_dilated\_again\_bool **=** perform\_dilation(image\_d\_opened\_bool, structure)

    image\_e **=** image\_e\_dilated\_again\_bool.astype(np.uint8) **\*** 255

    image\_f\_closed\_bool **=** perform\_erosion(image\_e\_dilated\_again\_bool, structure)

    image\_f **=** image\_f\_closed\_bool.astype(np.uint8) **\*** 255

    fig, axes **=** plt.subplots(2, 3, figsize**=**(15, 10))

    fig.suptitle(f'Morphological Filtering (Fig 9.11) - {image\_filename}', fontsize**=**16)

    axes[0, 0].imshow(display\_original, cmap**=**'gray', vmin**=**0, vmax**=**255)

    axes[0, 0].set\_title(r'a) Noisy Image ($A$)')

    axes[0, 1].imshow(image\_c, cmap**=**'gray', vmin**=**0, vmax**=**255)

    axes[0, 1].set\_title(r'c) Eroded Image ($A \ominus B$)')

    axes[0, 2].imshow(image\_d, cmap**=**'gray', vmin**=**0, vmax**=**255)

    axes[0, 2].set\_title(r'd) Opening ($A \circ B$)')

    axes[1, 0].imshow(image\_e, cmap**=**'gray', vmin**=**0, vmax**=**255)

    axes[1, 0].set\_title(r'e) Dilation of the Opening ($(A \circ B) \oplus B$)')

    axes[1, 1].imshow(image\_f, cmap**=**'gray', vmin**=**0, vmax**=**255)

    axes[1, 1].set\_title(r'f) Closing of the Opening ($(A \circ B) \bullet B$)')

    axes[1, 2].axis('off')

**for** ax **in** axes.flat:

        ax.axis('off')

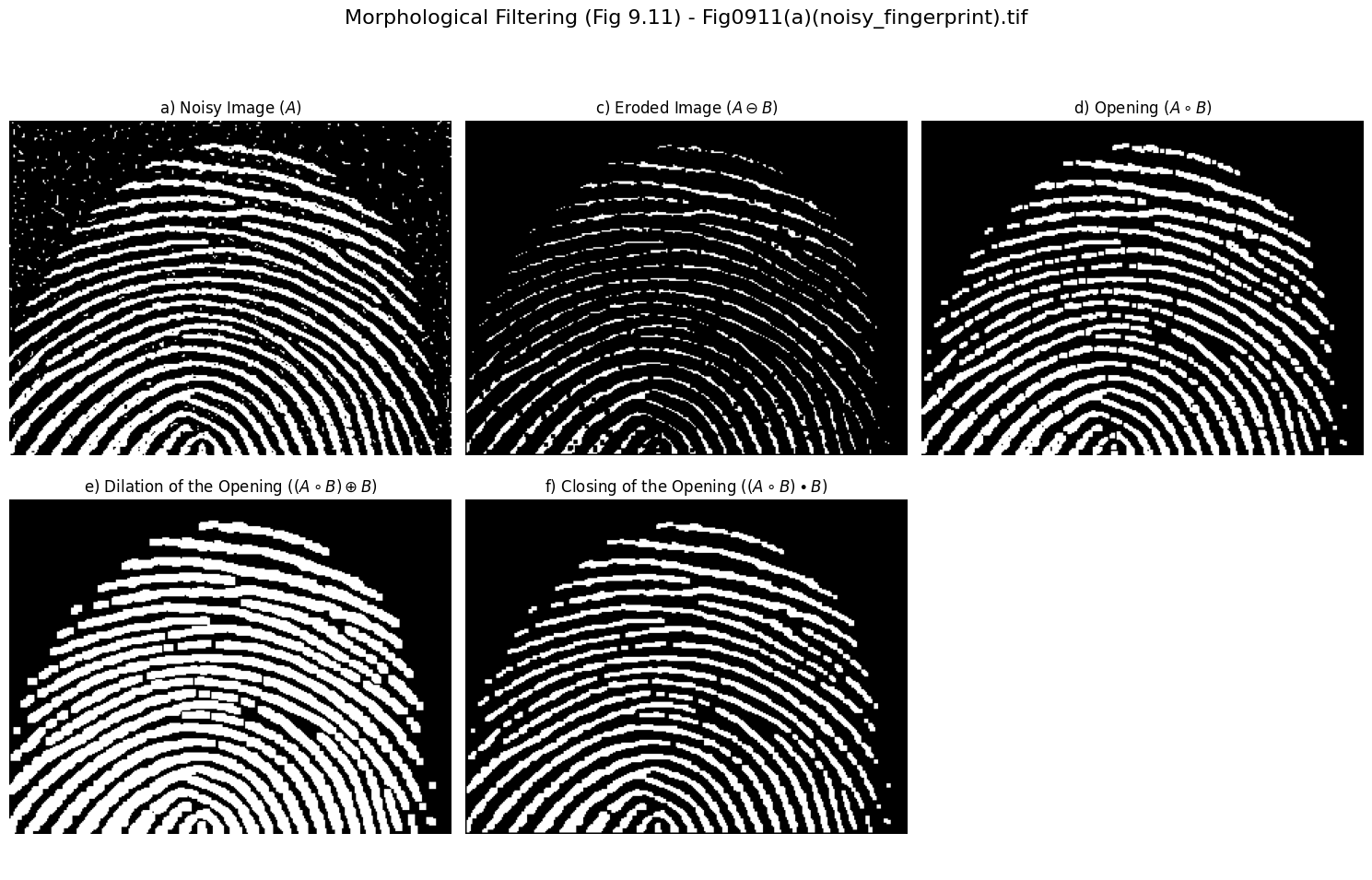
    plt.tight\_layout(rect**=**[0, 0.03, 1, 0.95])

    combined\_output\_path **=** os.path.join(output\_dir, f"{os.path.splitext(image\_filename)[0]}\_morph\_filter.png")

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

plt.close()

**Processed Images:**



**Homework4: Implement Figure 9.16 by programming with Python**

**Python Code:**

**import** numpy as np

**import** matplotlib.pyplot as plt

**import** os

**from** scipy **import** ndimage

**from** PIL **import** Image

MANUAL\_INVERT **=** False

**def** extract\_boundary(binary\_image, structure):

    # Formula: Beta(A) = A - (A eroded by B)

    eroded **=** ndimage.binary\_erosion(binary\_image, structure**=**structure)

**return** binary\_image ^ eroded

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

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

    output\_dir **=** "output\_image"

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

    image\_filename **=** "Fig0914(a)(licoln from penny).tif"

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

    raw\_image\_pil **=** Image.open(image\_path).convert('L')

    raw\_image **=** np.array(raw\_image\_pil)

    binary\_image **=** raw\_image > 128

**if** MANUAL\_INVERT:

        binary\_image **=** ~binary\_image

    structure **=** np.ones((3, 3), dtype**=**bool)

    boundary\_image **=** extract\_boundary(binary\_image, structure)

    fig, axes **=** plt.subplots(1, 2, figsize**=**(10, 5))

    fig.suptitle(f'Boundary Extraction (Fig 9.16) - {image\_filename}', fontsize**=**14)

    axes[0].imshow(binary\_image.astype(np.uint8) **\*** 255, cmap**=**'gray')

    axes[0].set\_title('a) Original Image')

    axes[0].axis('off')

    axes[1].imshow(boundary\_image.astype(np.uint8) **\*** 255, cmap**=**'gray')

    axes[1].set\_title('b) Boundary Extraction')

    axes[1].axis('off')

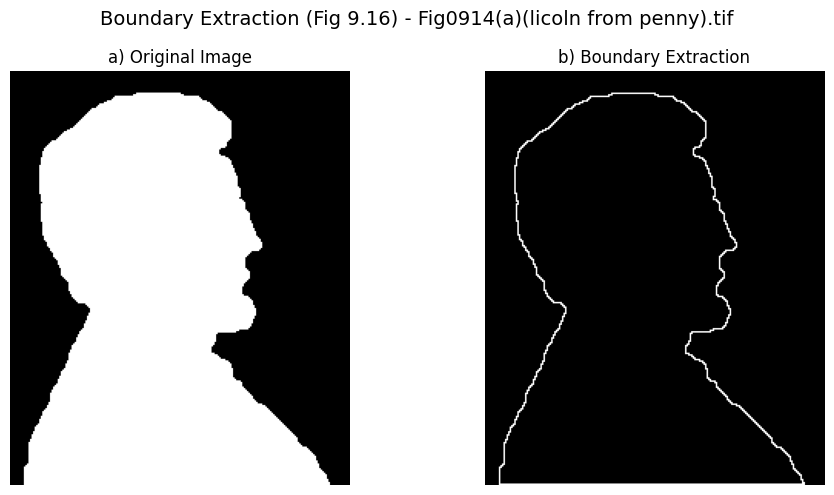
    plt.tight\_layout()

    save\_path **=** os.path.join(output\_dir, f"{os.path.splitext(image\_filename)[0]}\_boundary.png")

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

plt.close()

**Processed Images:**



**Homework5: Implement Figure 9.18 by programming with Python**

**Python Code:**

**import** numpy as np

**import** matplotlib.pyplot as plt

**import** os

**from** scipy **import** ndimage

**from** PIL **import** Image

MANUAL\_INVERT **=** False

**def** fill\_all\_holes\_robust(binary\_image):

    background\_mask **=** ~binary\_image

    labels, num\_features **=** ndimage.label(background\_mask)

**if** num\_features **==** 0:

**return** binary\_image

    sizes **=** np.bincount(labels.ravel())

    main\_background\_label **=** np.argmax(sizes[1:]) **+** 1

    holes\_mask **=** (labels !**=** 0) & (labels !**=** main\_background\_label)

**return** binary\_image | holes\_mask

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

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

    output\_dir **=** "output\_image"

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

    image\_filename **=** "Fig0916(a)(region-filling-reflections).tif"

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

    raw\_image\_pil **=** Image.open(image\_path).convert('L')

    raw\_image **=** np.array(raw\_image\_pil)

    binary\_image **=** raw\_image > 128

**if** MANUAL\_INVERT:

        binary\_image **=** ~binary\_image

    filled\_image **=** fill\_all\_holes\_robust(binary\_image)

    fig, axes **=** plt.subplots(1, 2, figsize**=**(10, 5))

    fig.suptitle(f'Morphological Hole Filling (Fig 9.18) - {image\_filename}', fontsize**=**14)

    axes[0].imshow(binary\_image.astype(np.uint8) **\*** 255, cmap**=**'gray', vmin**=**0, vmax**=**255)

    axes[0].set\_title('a) Original Image')

    axes[0].axis('off')

    axes[1].imshow(filled\_image.astype(np.uint8) **\*** 255, cmap**=**'gray', vmin**=**0, vmax**=**255)

    axes[1].set\_title('b) Hole Filling Result')

    axes[1].axis('off')

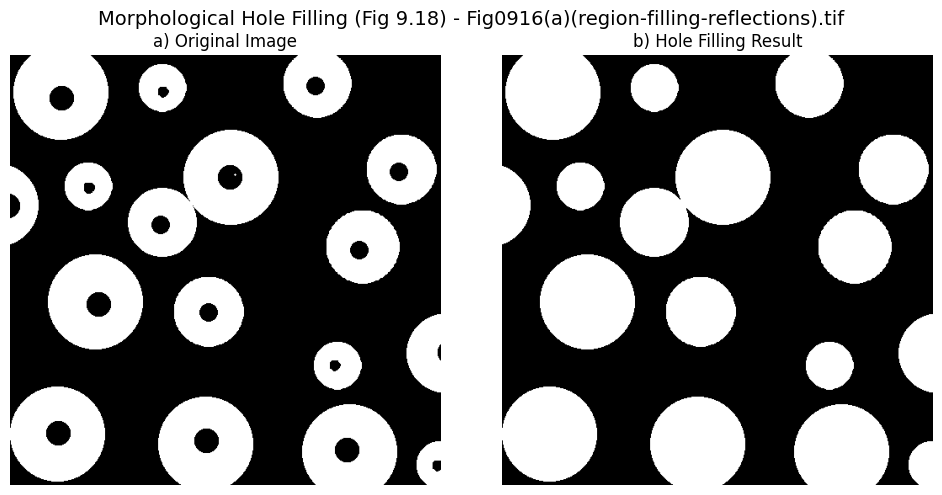
    plt.tight\_layout()

    save\_path **=** os.path.join(output\_dir, f"{os.path.splitext(image\_filename)[0]}\_hole\_filling\_final.png")

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

    plt.close()

**Processed Images:**



**Homework6: Implement Figure 9.20 by programming with Python**

**Python Code:**

**import** numpy as np

**import** matplotlib.pyplot as plt

**import** os

**from** PIL **import** Image

**from** scipy **import** ndimage

**def** manual\_threshold(img\_arr, threshold**=**205):

    binary **=** np.zeros\_like(img\_arr)

    binary[img\_arr > threshold] **=** 255

**return** binary

**def** manual\_erosion(binary\_img, kernel\_size**=**5):

    img\_bool **=** (binary\_img **/** 255).astype(float)

    kernel **=** np.ones((kernel\_size, kernel\_size), dtype**=**float)

    kernel\_area **=** kernel\_size **\*** kernel\_size

    neighbor\_sum **=** ndimage.convolve(img\_bool, kernel, mode**=**'constant', cval**=**0.0)

    eroded\_bool **=** neighbor\_sum > (kernel\_area **-** 0.5)

**return** eroded\_bool.astype(np.uint8) **\*** 255

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

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

    output\_dir **=** "output\_image"

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

    image\_filename **=** "Fig0918(a)(Chickenfilet with bones).tif"

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

    pil\_img **=** Image.open(image\_path).convert('L')

    img **=** np.array(pil\_img)

    binary **=** manual\_threshold(img, threshold**=**205)

    eroded **=** manual\_erosion(binary, kernel\_size**=**5)

    display\_b **=** 255 **-** binary

    display\_c **=** eroded

    fig, axes **=** plt.subplots(3, 1, figsize**=**(6, 12))

    fig.suptitle(f'Connected Components Extraction (Fig 9.20)', fontsize**=**14)

    axes[0].imshow(img, cmap**=**'gray')

    axes[0].set\_title('a) Original Image')

    axes[0].axis('off')

    axes[1].imshow(display\_b, cmap**=**'gray')

    axes[1].set\_title('b) Thresholded (Negative)')

    axes[1].axis('off')

    axes[2].imshow(display\_c, cmap**=**'gray')

    axes[2].set\_title('c) Eroded (Positive)')

    axes[2].axis('off')

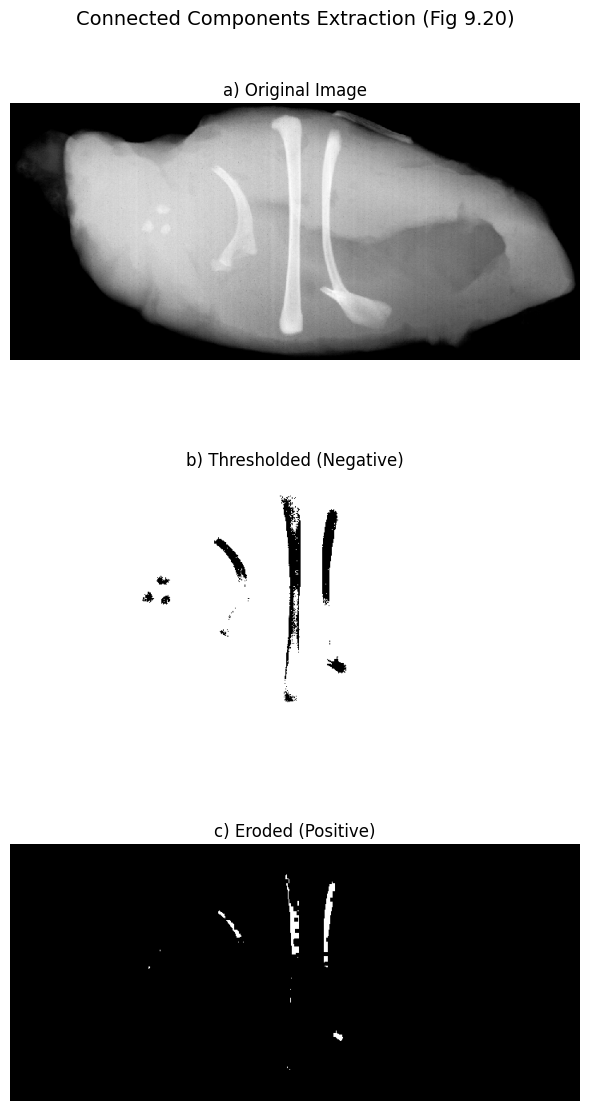
    plt.tight\_layout()

    save\_path **=** os.path.join(output\_dir, "Fig0920\_bone\_analysis\_compact.png")

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

    plt.close()

**Processed Images：**



**Homework7: Implement Figure 10.5 by programming with Python**

**Python Code:**

**import** numpy as np

**import** matplotlib.pyplot as plt

**import** os

**from** scipy **import** ndimage

**from** PIL **import** Image

**def** process\_laplacian\_line\_detection(image\_path, output\_dir):

    pil\_img **=** Image.open(image\_path).convert('L')

    img **=** np.array(pil\_img)

**if** np.mean(img) > 128:

        img **=** 255 **-** img

    img\_float **=** img.astype(float)

    kernel **=** np.array([[1, 1, 1],

                       [1, **-**8, 1],

                       [1, 1, 1]], dtype**=**float)

    laplacian **=** ndimage.convolve(img\_float, kernel)

    img\_b **=** np.clip(laplacian **+** 128, 0, 255)

    img\_c **=** np.abs(laplacian)

    img\_d **=** np.clip(laplacian, 0, None)

    zoom\_slice **=** (slice(125, 175), slice(220, 270))

    img\_b\_zoom **=** img\_b[zoom\_slice]

    fig **=** plt.figure(figsize**=**(15, 10))

    ax1 **=** plt.subplot(2, 3, 1)

    ax1.imshow(img, cmap**=**'gray')

    ax1.set\_title('a) Original Image')

    ax1.axis('off')

    ax2 **=** plt.subplot(2, 3, 2)

    ax2.imshow(img\_b, cmap**=**'gray')

    ax2.set\_title('b) Laplacian Image')

    ax2.axis('off')

    ax3 **=** plt.subplot(2, 3, 3)

    ax3.imshow(img\_b\_zoom, cmap**=**'gray', interpolation**=**'nearest')

    ax3.set\_title('b) Zoomed Detail')

    ax3.axis('off')

**for** spine **in** ax3.spines.values():

        spine.set\_edgecolor('red')

        spine.set\_linewidth(2)

    ax4 **=** plt.subplot(2, 3, 4)

    ax4.imshow(img\_c, cmap**=**'gray')

    ax4.set\_title('c) Absolute Value')

    ax4.axis('off')

    ax5 **=** plt.subplot(2, 3, 5)

    ax5.imshow(img\_d, cmap**=**'gray')

    ax5.set\_title('d) Positive Values')

    ax5.axis('off')

    plt.tight\_layout()

    save\_path **=** os.path.join(output\_dir, "Fig1005\_line\_detection.png")

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

    plt.close()

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

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

    output\_dir **=** "output\_image"

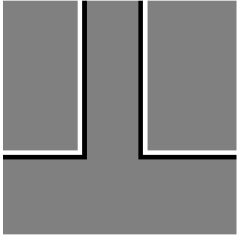
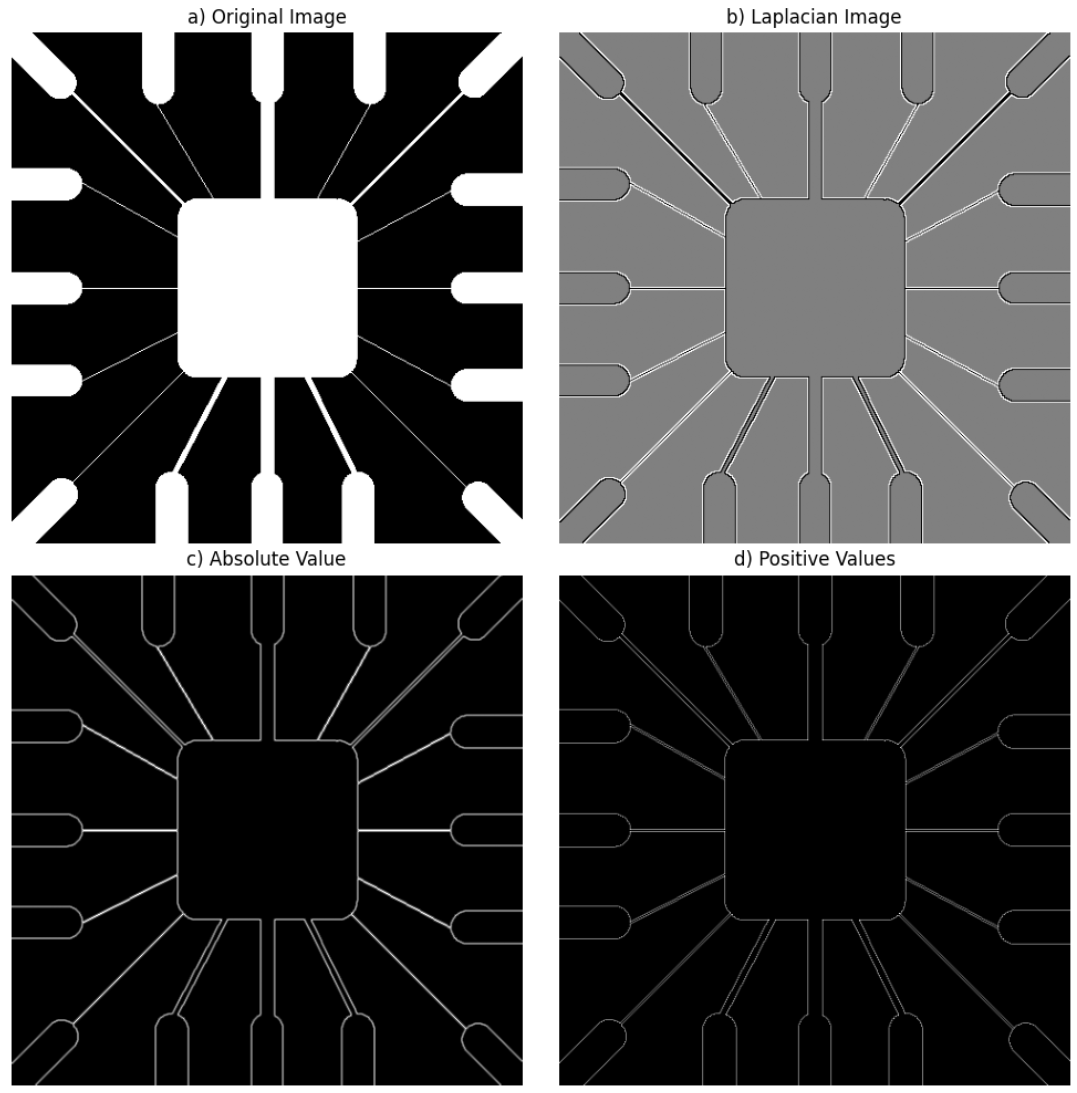
    os.makedirs(output\_dir, exist\_ok**=**True)

    filename **=** "Fig0905(a)(wirebond-mask).tif"

    path **=** os.path.join(input\_dir, filename)

    process\_laplacian\_line\_detection(path, output\_dir)

**Processed Images：**

****

**Homework8: Implement Figure 10.7 by programming with Python**

**Python Code:**

**import** numpy as np

**import** matplotlib.pyplot as plt

**import** os

**from** scipy **import** ndimage

**from** PIL **import** Image

**def** apply\_45\_degree\_kernel(img\_arr):

    img\_float **=** img\_arr.astype(float)

    kernel **=** np.array([[2, **-**1, **-**1],

                       [**-**1, 2, **-**1],

                       [**-**1, **-**1, 2]], dtype**=**float)

    response **=** ndimage.convolve(img\_float, kernel)

**return** response

**def** process\_and\_visualize\_six\_figures(image\_path, output\_dir):

    pil\_img **=** Image.open(image\_path).convert('L')

    img **=** np.array(pil\_img)

    raw\_response **=** apply\_45\_degree\_kernel(img)

    max\_abs **=** np.max(np.abs(raw\_response))

**if** max\_abs **==** 0: max\_abs **=** 1

    display\_b **=** ((raw\_response **/** max\_abs) **\*** 127.5 **+** 127.5).astype(np.uint8)

    slice\_c **=** (slice(10, 110), slice(10, 110))

    img\_c **=** display\_b[slice\_c]

    slice\_d **=** (slice(370, 480), slice(370, 480))

    img\_d **=** display\_b[slice\_d]

    response\_e **=** raw\_response.copy()

    response\_e[response\_e < 0] **=** 0

    display\_e **=** np.clip(response\_e, 0, 255).astype(np.uint8)

    max\_val **=** np.max(response\_e)

**if** max\_val > 0:

        T **=** max\_val

        mask\_f **=** response\_e >**=** T

**else**:

        mask\_f **=** np.zeros\_like(response\_e, dtype**=**bool)

    img\_f\_binary **=** np.zeros\_like(img)

    img\_f\_binary[mask\_f] **=** 255

    structure **=** np.ones((5, 5), dtype**=**bool)

    img\_f\_dilated **=** ndimage.binary\_dilation(img\_f\_binary > 0, structure**=**structure)

    img\_f **=** img\_f\_dilated.astype(np.uint8) **\*** 255

    fig, axes **=** plt.subplots(2, 3, figsize**=**(15, 10))

    axes[0, 0].imshow(img, cmap**=**'gray')

    axes[0, 0].set\_title('(a) Original Image')

    axes[0, 0].axis('off')

    axes[0, 1].imshow(display\_b, cmap**=**'gray')

    axes[0, 1].set\_title('(b) +45 Line Response')

    axes[0, 1].axis('off')

    axes[0, 2].imshow(img\_c, cmap**=**'gray')

    axes[0, 2].set\_title('(c) Zoomed Top-Left')

    axes[0, 2].axis('off')

**for** spine **in** axes[0, 2].spines.values():

        spine.set\_edgecolor('white')

    axes[1, 0].imshow(img\_d, cmap**=**'gray')

    axes[1, 0].set\_title('(d) Zoomed Bottom-Right')

    axes[1, 0].axis('off')

    axes[1, 1].imshow(display\_e, cmap**=**'gray')

    axes[1, 1].set\_title('(e) Positive Response')

    axes[1, 1].axis('off')

    axes[1, 2].imshow(img\_f, cmap**=**'gray')

    axes[1, 2].set\_title('(f) Strongest Points (>T)')

    axes[1, 2].axis('off')

    plt.tight\_layout()

    save\_path **=** os.path.join(output\_dir, "Fig1007\_full\_process.png")

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

    plt.close()

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

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

    output\_dir **=** "output\_image"

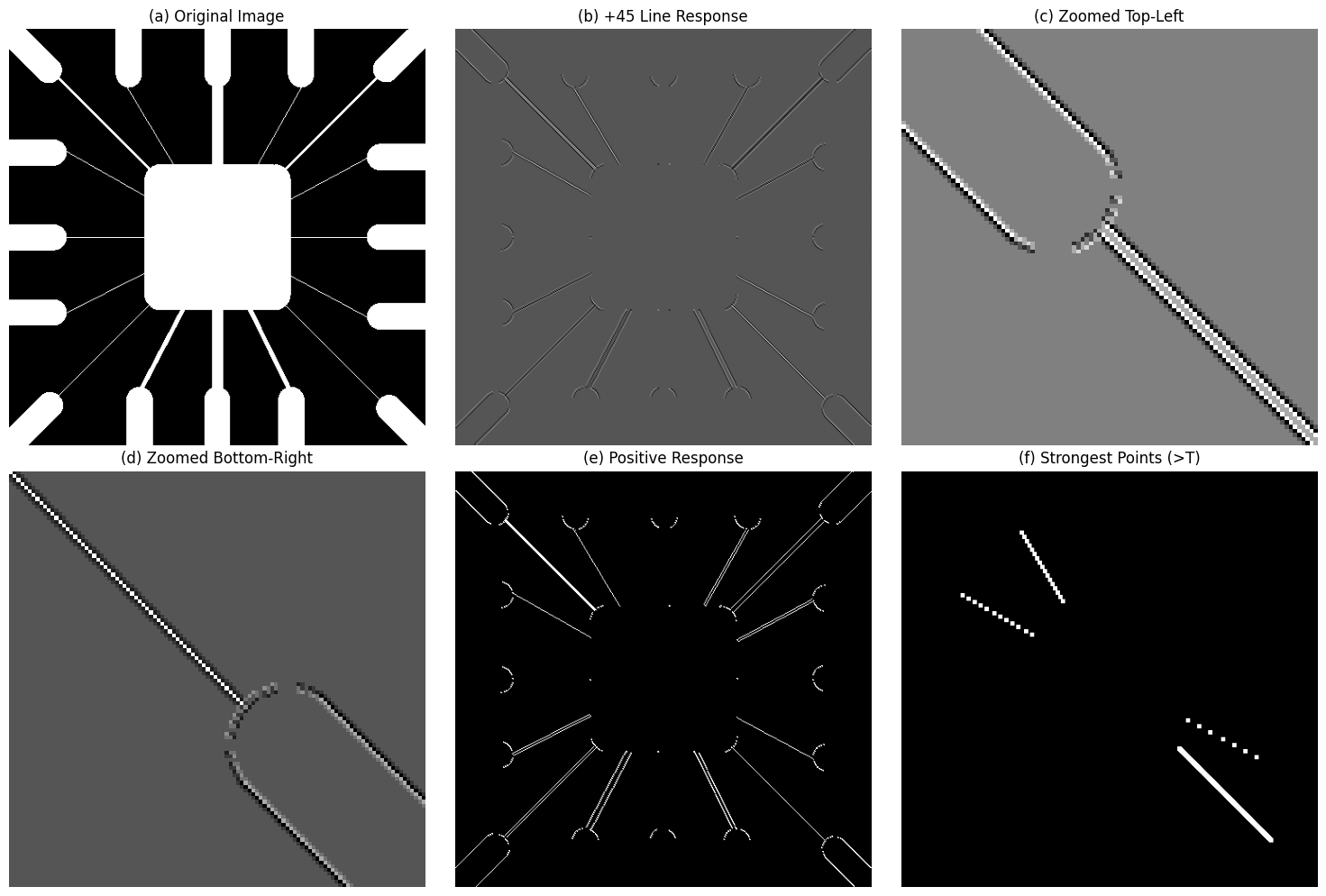
    os.makedirs(output\_dir, exist\_ok**=**True)

    filename **=** "Fig1007(a)(wirebond\_mask).tif"

    path **=** os.path.join(input\_dir, filename)

process\_and\_visualize\_six\_figures(path, output\_dir)

**Processed Images：**

****

**Homework9: Implement Figure 10.16 by programming with Python**

**Python Code:**

**import** numpy as np

**import** matplotlib.pyplot as plt

**import** os

**from** scipy **import** ndimage

**from** PIL **import** Image

**def** compute\_sobel\_gradients(img\_arr):

    img\_float **=** img\_arr.astype(float)

    kx **=** np.array([[**-**1, **-**2, **-**1],

                   [0, 0, 0],

                   [1, 2, 1]], dtype**=**float)

    ky **=** np.array([[**-**1, 0, 1],

                   [**-**2, 0, 2],

                   [**-**1, 0, 1]], dtype**=**float)

    gx **=** ndimage.convolve(img\_float, kx)

    gy **=** ndimage.convolve(img\_float, ky)

**return** np.abs(gx), np.abs(gy)

**def** process\_sobel\_edge\_detection(image\_path, output\_dir):

    pil\_img **=** Image.open(image\_path).convert('L')

    img **=** np.array(pil\_img)

    abs\_gx, abs\_gy **=** compute\_sobel\_gradients(img)

    magnitude **=** abs\_gx **+** abs\_gy

    display\_gx **=** np.clip(abs\_gx, 0, 255).astype(np.uint8)

    display\_gy **=** np.clip(abs\_gy, 0, 255).astype(np.uint8)

    display\_mag **=** np.clip(magnitude, 0, 255).astype(np.uint8)

    fig, axes **=** plt.subplots(2, 2, figsize**=**(10, 10))

    axes[0, 0].imshow(img, cmap**=**'gray')

    axes[0, 0].set\_title('(a) Original Image')

    axes[0, 0].axis('off')

    axes[0, 1].imshow(display\_gx, cmap**=**'gray')

    axes[0, 1].set\_title('(b) |gx| component (Vertical Edges)')

    axes[0, 1].axis('off')

    axes[1, 0].imshow(display\_gy, cmap**=**'gray')

    axes[1, 0].set\_title('(c) |gy| component (Horizontal Edges)')

    axes[1, 0].axis('off')

    axes[1, 1].imshow(display\_mag, cmap**=**'gray')

    axes[1, 1].set\_title('(d) Gradient image |gx| + |gy|')

    axes[1, 1].axis('off')

    plt.tight\_layout()

    save\_path **=** os.path.join(output\_dir, "Fig1016\_sobel\_gradients.png")

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

    plt.close()

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

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

    output\_dir **=** "output\_image"

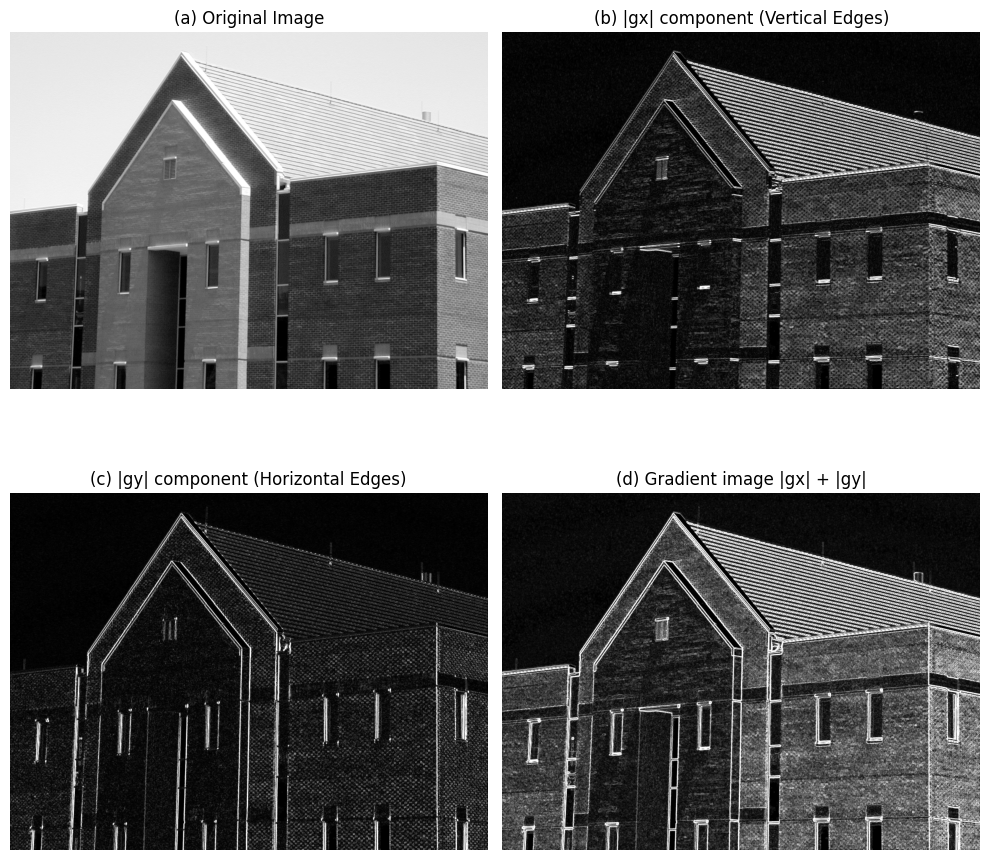
    os.makedirs(output\_dir, exist\_ok**=**True)

    filename **=** "Fig1016(a)(building\_original).tif"

    path **=** os.path.join(input\_dir, filename)

    process\_sobel\_edge\_detection(path, output\_dir)

**Processed Images：**

****

**Homework10: Implement Figure 10.18 by programming with Python**

**Python Code:**

**import** numpy as np

**import** matplotlib.pyplot as plt

**import** os

**from** scipy **import** ndimage

**from** PIL **import** Image

**def** apply\_average\_filter(img\_arr, kernel\_size**=**5):

    img\_float **=** img\_arr.astype(float)

    smoothed **=** ndimage.uniform\_filter(img\_float, size**=**kernel\_size)

**return** smoothed

**def** compute\_sobel\_gradients(img\_arr):

    img\_float **=** img\_arr.astype(float) **/** 255.0

    kx **=** np.array([[**-**1, **-**2, **-**1],

                   [0, 0, 0],

                   [1, 2, 1]], dtype**=**float)

    ky **=** np.array([[**-**1, 0, 1],

                   [**-**2, 0, 2],

                   [**-**1, 0, 1]], dtype**=**float)

    gx **=** ndimage.convolve(img\_float, kx)

    gy **=** ndimage.convolve(img\_float, ky)

**return** np.abs(gx), np.abs(gy)

**def** normalize\_for\_display(img\_data):

    min\_val **=** np.min(img\_data)

    max\_val **=** np.max(img\_data)

**if** max\_val **-** min\_val **==** 0:

**return** np.zeros\_like(img\_data, dtype**=**np.uint8)

    norm\_img **=** (img\_data **-** min\_val) **/** (max\_val **-** min\_val) **\*** 255

**return** norm\_img.astype(np.uint8)

**def** process\_smoothed\_sobel(image\_path, output\_dir):

    pil\_img **=** Image.open(image\_path).convert('L')

    img **=** np.array(pil\_img)

    img\_smoothed **=** apply\_average\_filter(img, kernel\_size**=**5)

    abs\_gx, abs\_gy **=** compute\_sobel\_gradients(img\_smoothed)

    magnitude **=** abs\_gx **+** abs\_gy

    display\_gx **=** normalize\_for\_display(abs\_gx)

    display\_gy **=** normalize\_for\_display(abs\_gy)

    display\_mag **=** normalize\_for\_display(magnitude)

    fig, axes **=** plt.subplots(2, 2, figsize**=**(10, 12))

    axes[0, 0].imshow(img\_smoothed, cmap**=**'gray')

    axes[0, 0].set\_title('(a) Smoothed Image (5x5 Avg)')

    axes[0, 0].axis('off')

    axes[0, 1].imshow(display\_gx, cmap**=**'gray')

    axes[0, 1].set\_title('(b) |gx| (Vertical)')

    axes[0, 1].axis('off')

    axes[1, 0].imshow(display\_gy, cmap**=**'gray')

    axes[1, 0].set\_title('(c) |gy| (Horizontal)')

    axes[1, 0].axis('off')

    axes[1, 1].imshow(display\_mag, cmap**=**'gray')

    axes[1, 1].set\_title('(d) Gradient |gx| + |gy|')

    axes[1, 1].axis('off')

    plt.tight\_layout()

    save\_path **=** os.path.join(output\_dir, "Fig1018\_smoothed\_sobel.png")

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

    plt.close()

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

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

    output\_dir **=** "output\_image"

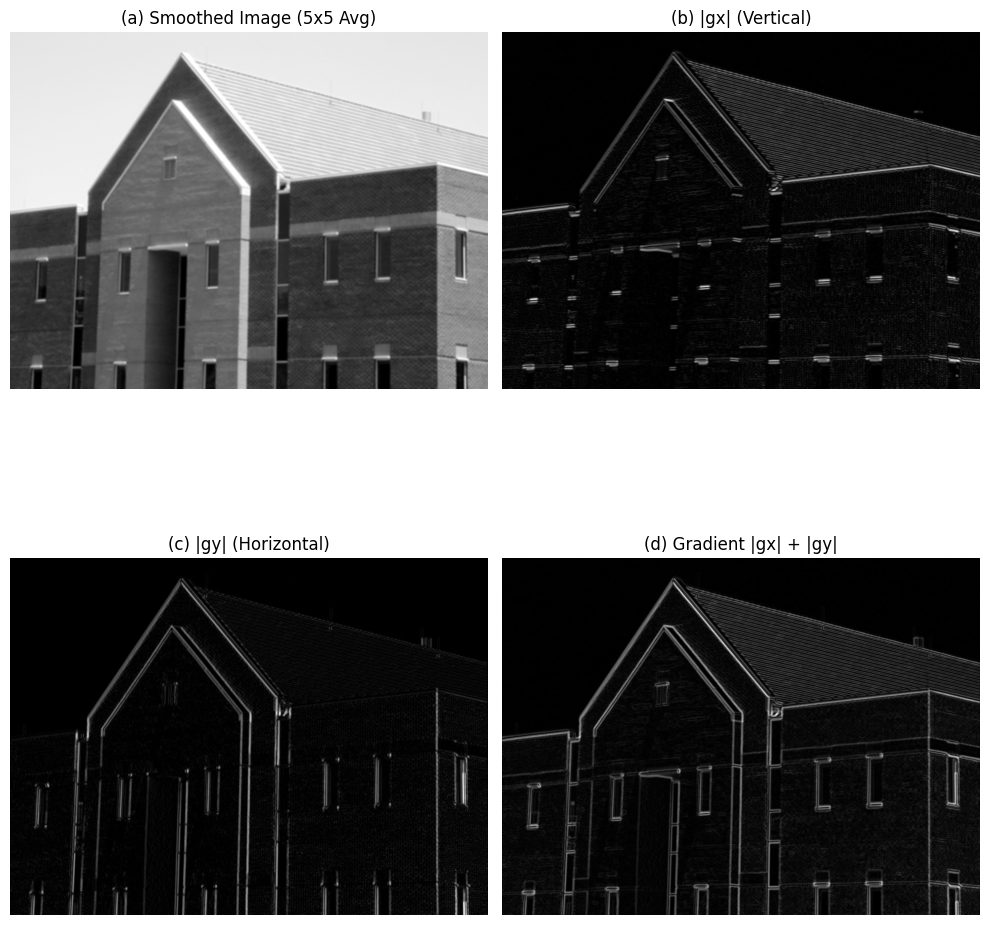
    os.makedirs(output\_dir, exist\_ok**=**True)

    filename **=** "Fig1016(a)(building\_original).tif"

    path **=** os.path.join(input\_dir, filename)

    process\_smoothed\_sobel(path, output\_dir)

**Processed Images：**

****

**Homework11: Implement Figure 10.19 by programming with Python**

**Python Code:**

**import** numpy as np

**import** matplotlib.pyplot as plt

**import** os

**from** scipy **import** ndimage

**from** PIL **import** Image

**def** compute\_kirsch\_responses(img\_arr):

    img\_float **=** img\_arr.astype(float) **/** 255.0

    k\_nw **=** np.array([[**-**3, 5, 5],

                     [**-**3, 0, 5],

                     [**-**3, **-**3, **-**3]], dtype**=**float)

    k\_sw **=** np.array([[5, 5, **-**3],

                     [5, 0, **-**3],

                     [**-**3, **-**3, **-**3]], dtype**=**float)

    resp\_nw **=** ndimage.convolve(img\_float, k\_nw)

    resp\_sw **=** ndimage.convolve(img\_float, k\_sw)

**return** np.abs(resp\_nw), np.abs(resp\_sw)

**def** normalize\_for\_display(img\_data):

    min\_val **=** np.min(img\_data)

    max\_val **=** np.max(img\_data)

**if** max\_val **-** min\_val **==** 0:

**return** np.zeros\_like(img\_data, dtype**=**np.uint8)

    norm\_img **=** (img\_data **-** min\_val) **/** (max\_val **-** min\_val) **\*** 255

**return** norm\_img.astype(np.uint8)

**def** process\_diagonal\_edge\_detection(image\_path, output\_dir):

    pil\_img **=** Image.open(image\_path).convert('L')

    img **=** np.array(pil\_img)

    img\_smoothed **=** img

    abs\_nw, abs\_sw **=** compute\_kirsch\_responses(img\_smoothed)

    display\_nw **=** normalize\_for\_display(abs\_nw)

    display\_sw **=** normalize\_for\_display(abs\_sw)

    fig, axes **=** plt.subplots(1, 2, figsize**=**(12, 6))

    fig.suptitle(f'Diagonal Edge Detection (Fig 10.19)', fontsize**=**14)

    axes[0].imshow(display\_nw, cmap**=**'gray')

    axes[0].set\_title('(a) NW Kernel (45 degree)')

    axes[0].axis('off')

    axes[1].imshow(display\_sw, cmap**=**'gray')

    axes[1].set\_title('(b) SW Kernel (-45 degree)')

    axes[1].axis('off')

    plt.tight\_layout()

    save\_path **=** os.path.join(output\_dir, "Fig1019\_diagonal\_edges.png")

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

    plt.close()

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

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

    output\_dir **=** "output\_image"

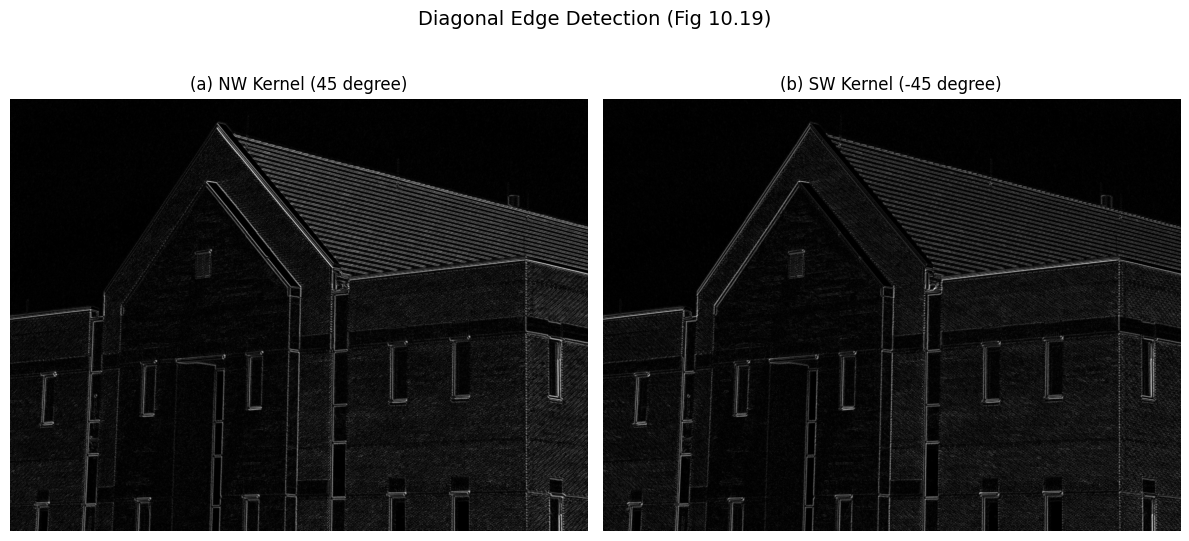
    os.makedirs(output\_dir, exist\_ok**=**True)

    filename **=** "Fig1016(a)(building\_original).tif"

    path **=** os.path.join(input\_dir, filename)

    process\_diagonal\_edge\_detection(path, output\_dir)

**Processed Images：**

****

**Homework12: Implement Figure 10.20 by programming with Python**

**Python Code:**

**import** numpy as np

**import** matplotlib.pyplot as plt

**import** os

**from** scipy **import** ndimage

**from** PIL **import** Image

**def** apply\_average\_filter(img\_arr, kernel\_size**=**5):

    img\_float **=** img\_arr.astype(float)

**return** ndimage.uniform\_filter(img\_float, size**=**kernel\_size)

**def** compute\_sobel\_magnitude(img\_arr):

    img\_float **=** img\_arr.astype(float) **/** 255.0

    kx **=** np.array([[**-**1, 0, 1],

                   [**-**2, 0, 2],

                   [**-**1, 0, 1]], dtype**=**float)

    ky **=** np.array([[**-**1, **-**2, **-**1],

                   [0, 0, 0],

                   [1, 2, 1]], dtype**=**float)

    gx **=** ndimage.convolve(img\_float, kx)

    gy **=** ndimage.convolve(img\_float, ky)

**return** np.abs(gx) **+** np.abs(gy)

**def** threshold\_image(magnitude, percentage**=**0.33):

    max\_val **=** np.max(magnitude)

    T **=** max\_val **\*** percentage

    binary\_edges **=** (magnitude >**=** T).astype(np.uint8) **\*** 255

**return** binary\_edges

**def** process\_thresholded\_edges(image\_path, output\_dir):

    pil\_img **=** Image.open(image\_path).convert('L')

    img **=** np.array(pil\_img)

    mag\_original **=** compute\_sobel\_magnitude(img)

    edges\_original **=** threshold\_image(mag\_original, percentage**=**0.33)

    img\_smoothed **=** apply\_average\_filter(img, kernel\_size**=**5)

    mag\_smoothed **=** compute\_sobel\_magnitude(img\_smoothed)

    edges\_smoothed **=** threshold\_image(mag\_smoothed, percentage**=**0.33)

    fig, axes **=** plt.subplots(1, 2, figsize**=**(12, 6))

    fig.suptitle(f'Thresholded Gradient (Fig 10.20)', fontsize**=**14)

    axes[0].imshow(edges\_original, cmap**=**'gray')

    axes[0].set\_title('(a) Thresholded Gradient of Original Image')

    axes[0].axis('off')

    axes[1].imshow(edges\_smoothed, cmap**=**'gray')

    axes[1].set\_title('(b) Thresholded Gradient of Smoothed Image')

    axes[1].axis('off')

    plt.tight\_layout()

    save\_path **=** os.path.join(output\_dir, "Fig1020\_thresholded\_edges.png")

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

    plt.close()

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

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

    output\_dir **=** "output\_image"

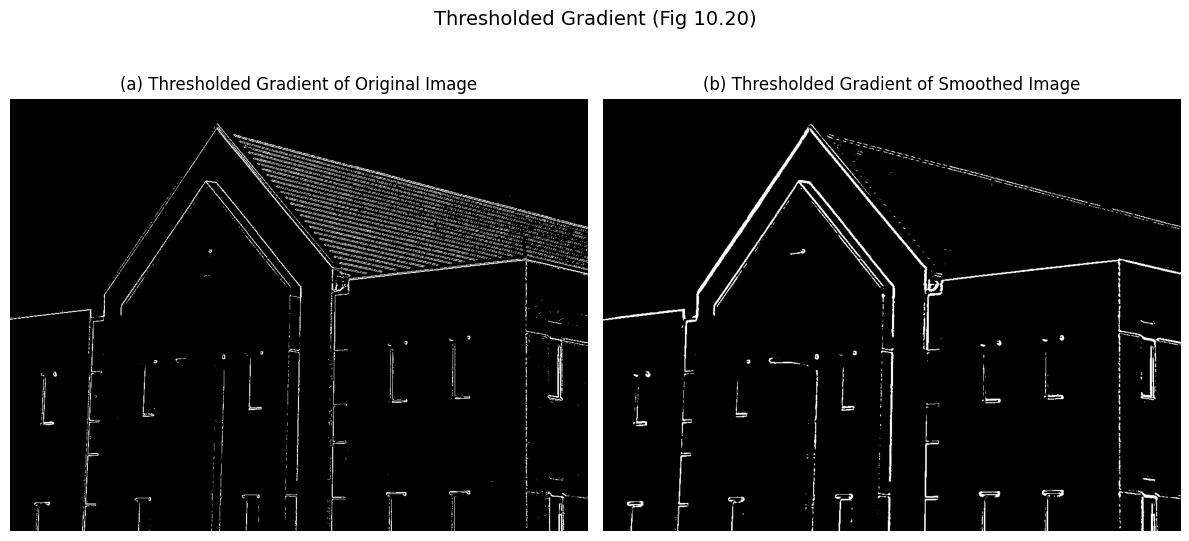
    os.makedirs(output\_dir, exist\_ok**=**True)

    filename **=** "Fig1016(a)(building\_original).tif"

    path **=** os.path.join(input\_dir, filename)

    process\_thresholded\_edges(path, output\_dir)

**Processed Images：**

****

**Homework13: Implement Figure 10.22 by programming with Python**

**Python Code:**

**import** numpy as np

**import** matplotlib.pyplot as plt

**import** os

**from** scipy **import** ndimage

**from** PIL **import** Image

**def** compute\_log\_response(img\_arr, sigma**=**4):

    """

    Compute Laplacian of Gaussian (LoG)

    """

    img\_float **=** img\_arr.astype(float)

    log\_response **=** ndimage.gaussian\_laplace(img\_float, sigma**=**sigma, truncate**=**3.0)

**return** log\_response

**def** find\_zero\_crossings(log\_img, threshold**=**0):

    """

    Find zero crossings

    """

    rows, cols **=** log\_img.shape

    edges **=** np.zeros\_like(log\_img, dtype**=**np.uint8)

    curr\_h **=** log\_img[:, :**-**1]

    right\_h **=** log\_img[:, 1:]

    sign\_diff\_h **=** (curr\_h **\*** right\_h) < 0

    curr\_v **=** log\_img[:**-**1, :]

    down\_v **=** log\_img[1:, :]

    sign\_diff\_v **=** (curr\_v **\*** down\_v) < 0

**if** threshold > 0:

        mag\_check\_h **=** np.abs(curr\_h **-** right\_h) > threshold

        mag\_check\_v **=** np.abs(curr\_v **-** down\_v) > threshold

        edges[:, :**-**1][sign\_diff\_h & mag\_check\_h] **=** 255

        edges[:**-**1, :][sign\_diff\_v & mag\_check\_v] **=** 255

**else**:

        edges[:, :**-**1][sign\_diff\_h] **=** 255

        edges[:**-**1, :][sign\_diff\_v] **=** 255

**return** edges

**def** process\_marr\_hildreth\_final(image\_path, output\_dir):

    pil\_img **=** Image.open(image\_path).convert('L')

    img **=** np.array(pil\_img)

    noise **=** np.random.normal(0, 1.0, img.shape)

    img\_noisy **=** img.astype(float) **+** noise

    log\_response **=** compute\_log\_response(img\_noisy, sigma**=**4)

    max\_log **=** np.max(np.abs(log\_response))

**if** max\_log **==** 0: max\_log **=** 1

    display\_log **=** ((log\_response **/** max\_log) **\*** 127.5 **+** 127.5).astype(np.uint8)

    edges\_zero **=** find\_zero\_crossings(log\_response, threshold**=**0)

    thresh\_val **=** 0.04 **\*** max\_log

    edges\_thresh **=** find\_zero\_crossings(log\_response, threshold**=**thresh\_val)

    plt.rcParams.update({'font.size': 10})

    fig, axes **=** plt.subplots(2, 2, figsize**=**(10, 12))

    axes[0, 0].imshow(img, cmap**=**'gray')

    axes[0, 0].set\_title('(a) Original Image')

    axes[0, 0].axis('off')

    axes[0, 1].imshow(display\_log, cmap**=**'gray')

    axes[0, 1].set\_title('(b) LoG (sigma=4, n=25)')

    axes[0, 1].axis('off')

    axes[1, 0].imshow(edges\_zero, cmap**=**'gray')

    axes[1, 0].set\_title('(c) Zero Crossing (Thresh=0)')

    axes[1, 0].axis('off')

    axes[1, 1].imshow(edges\_thresh, cmap**=**'gray')

    axes[1, 1].set\_title('(d) Zero Crossing (Thresh=4% Max)')

    axes[1, 1].axis('off')

    plt.tight\_layout()

    save\_path **=** os.path.join(output\_dir, "Fig1022\_Marr\_Hildreth\_Final.png")

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

    plt.close()

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

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

    output\_dir **=** "output\_image"

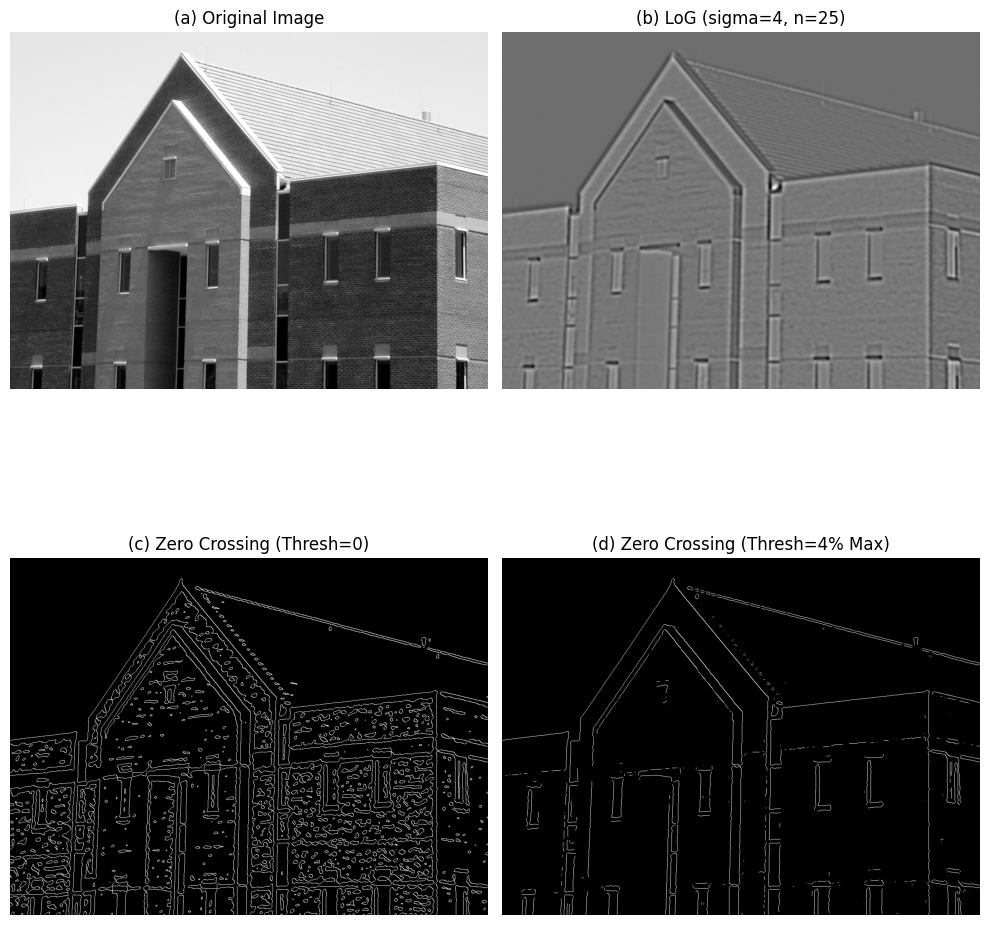
    os.makedirs(output\_dir, exist\_ok**=**True)

    filename **=** "Fig1016(a)(building\_original).tif"

    path **=** os.path.join(input\_dir, filename)

process\_marr\_hildreth\_final(path, output\_dir)

**Processed Images：**

****

**Homework14: Implement Figure 10.25 by programming with Python**

**Python Code:**

**import** numpy as np

**import** matplotlib.pyplot as plt

**import** os

**from** scipy **import** ndimage

**from** PIL **import** Image

**def** gaussian\_kernel(size, sigma**=**1):

    size **=** int(size) **//** 2

    x, y **=** np.mgrid[**-**size:size **+** 1, **-**size:size **+** 1]

    normal **=** 1 **/** (2.0 **\*** np.pi **\*** sigma **\*\*** 2)

    g **=** np.exp(**-**((x **\*\*** 2 **+** y **\*\*** 2) **/** (2.0 **\*** sigma **\*\*** 2))) **\*** normal

**return** g

**def** apply\_average\_filter(img\_arr, kernel\_size**=**5):

    img\_float **=** img\_arr.astype(float)

**return** ndimage.uniform\_filter(img\_float, size**=**kernel\_size)

**def** get\_image\_b(img):

    img\_smoothed **=** apply\_average\_filter(img, kernel\_size**=**5)

    img\_float **=** img\_smoothed.astype(float) **/** 255.0

    kx **=** np.array([[**-**1, 0, 1], [**-**2, 0, 2], [**-**1, 0, 1]], dtype**=**float)

    ky **=** np.array([[**-**1, **-**2, **-**1], [0, 0, 0], [1, 2, 1]], dtype**=**float)

    gx **=** ndimage.convolve(img\_float, kx)

    gy **=** ndimage.convolve(img\_float, ky)

    magnitude **=** np.abs(gx) **+** np.abs(gy)

    max\_val **=** np.max(magnitude)

    threshold **=** 0.33 **\*** max\_val

    edges **=** (magnitude >**=** threshold).astype(np.uint8) **\*** 255

**return** edges

**def** get\_image\_c(img):

    img\_float **=** img.astype(float)

    log\_resp **=** ndimage.gaussian\_laplace(img\_float, sigma**=**4, truncate**=**3.0)

    max\_log **=** np.max(np.abs(log\_resp))

    threshold **=** 0.04 **\*** max\_log

    edges **=** np.zeros\_like(img, dtype**=**np.uint8)

    curr\_h **=** log\_resp[:, :**-**1]

    right\_h **=** log\_resp[:, 1:]

    sign\_diff\_h **=** (curr\_h **\*** right\_h) < 0

    mag\_check\_h **=** np.abs(curr\_h **-** right\_h) > threshold

    edges[:, :**-**1][sign\_diff\_h & mag\_check\_h] **=** 255

    curr\_v **=** log\_resp[:**-**1, :]

    down\_v **=** log\_resp[1:, :]

    sign\_diff\_v **=** (curr\_v **\*** down\_v) < 0

    mag\_check\_v **=** np.abs(curr\_v **-** down\_v) > threshold

    edges[:**-**1, :][sign\_diff\_v & mag\_check\_v] **=** 255

**return** edges

**def** sobel\_filters\_manual(img):

    Kx **=** np.array([[**-**1, 0, 1], [**-**2, 0, 2], [**-**1, 0, 1]], dtype**=**float)

    Ky **=** np.array([[**-**1, **-**2, **-**1], [0, 0, 0], [1, 2, 1]], dtype**=**float)

    Ix **=** ndimage.convolve(img, Kx)

    Iy **=** ndimage.convolve(img, Ky)

    G **=** np.hypot(Ix, Iy)

    G **=** G **/** G.max() **\*** 255

    theta **=** np.arctan2(Iy, Ix)

**return** G, theta

**def** non\_max\_suppression\_manual(img, D):

    M, N **=** img.shape

    Z **=** np.zeros((M, N), dtype**=**np.float32)

    angle **=** D **\*** 180. **/** np.pi

    angle[angle < 0] **+=** 180

**for** i **in** range(1, M **-** 1):

**for** j **in** range(1, N **-** 1):

**try**:

                q **=** 255

                r **=** 255

**if** (0 <**=** angle[i, j] < 22.5) **or** (157.5 <**=** angle[i, j] <**=** 180):

                    q **=** img[i, j **+** 1]

                    r **=** img[i, j **-** 1]

**elif** (22.5 <**=** angle[i, j] < 67.5):

                    q **=** img[i **+** 1, j **-** 1]

                    r **=** img[i **-** 1, j **+** 1]

**elif** (67.5 <**=** angle[i, j] < 112.5):

                    q **=** img[i **+** 1, j]

                    r **=** img[i **-** 1, j]

**elif** (112.5 <**=** angle[i, j] < 157.5):

                    q **=** img[i **-** 1, j **-** 1]

                    r **=** img[i **+** 1, j **+** 1]

**if** (img[i, j] >**=** q) **and** (img[i, j] >**=** r):

                    Z[i, j] **=** img[i, j]

**else**:

                    Z[i, j] **=** 0

**except** IndexError:

**pass**

**return** Z

**def** hysteresis\_manual(img, weak, strong**=**255):

    M, N **=** img.shape

    strong\_rows, strong\_cols **=** np.where(img **==** strong)

    stack **=** list(zip(strong\_rows, strong\_cols))

**while** stack:

        i, j **=** stack.pop()

**for** di **in** [**-**1, 0, 1]:

**for** dj **in** [**-**1, 0, 1]:

**if** di **==** 0 **and** dj **==** 0: **continue**

                ni, nj **=** i **+** di, j **+** dj

**if** 0 <**=** ni < M **and** 0 <**=** nj < N:

**if** img[ni, nj] **==** weak:

                        img[ni, nj] **=** strong

                        stack.append((ni, nj))

**for** i **in** range(M):

**for** j **in** range(N):

**if** img[i, j] **==** weak:

                img[i, j] **=** 0

**return** img

**def** get\_image\_d\_manual\_canny(img):

    img\_float **=** img.astype(float)

    kernel **=** gaussian\_kernel(size**=**25, sigma**=**4)

    img\_smoothed **=** ndimage.convolve(img\_float, kernel)

    grad\_mag, grad\_dir **=** sobel\_filters\_manual(img\_smoothed)

    img\_nms **=** non\_max\_suppression\_manual(grad\_mag, grad\_dir)

    max\_val **=** img\_nms.max()

    lowThreshold **=** max\_val **\*** 0.04

    highThreshold **=** max\_val **\*** 0.10

    res **=** np.zeros\_like(img\_nms, dtype**=**np.int32)

    weak\_val **=** np.int32(50)

    strong\_val **=** np.int32(255)

    strong\_i, strong\_j **=** np.where(img\_nms >**=** highThreshold)

    weak\_i, weak\_j **=** np.where((img\_nms <**=** highThreshold) & (img\_nms >**=** lowThreshold))

    res[strong\_i, strong\_j] **=** strong\_val

    res[weak\_i, weak\_j] **=** weak\_val

    img\_final **=** hysteresis\_manual(res, weak\_val, strong\_val)

**return** img\_final

**def** process\_comparison(image\_path, output\_dir):

    pil\_img **=** Image.open(image\_path).convert('L')

    img **=** np.array(pil\_img)

    img\_b **=** get\_image\_b(img)

    img\_c **=** get\_image\_c(img)

    img\_d **=** get\_image\_d\_manual\_canny(img)

    fig, axes **=** plt.subplots(2, 2, figsize**=**(12, 10))

    axes[0, 0].imshow(img, cmap**=**'gray')

    axes[0, 0].set\_title('(a) Original Image')

    axes[0, 0].axis('off')

    axes[0, 1].imshow(img\_b, cmap**=**'gray')

    axes[0, 1].set\_title('(b) Thresholded Gradient (Smoothed)')

    axes[0, 1].axis('off')

    axes[1, 0].imshow(img\_c, cmap**=**'gray')

    axes[1, 0].set\_title('(c) Marr-Hildreth (LoG)')

    axes[1, 0].axis('off')

    axes[1, 1].imshow(img\_d, cmap**=**'gray')

    axes[1, 1].set\_title('(d) Canny Algorithm (Manual)')

    axes[1, 1].axis('off')

    plt.tight\_layout()

    save\_path **=** os.path.join(output\_dir, "Fig1025\_Full\_Comparison\_Manual.png")

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

    plt.close()

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

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

    output\_dir **=** "output\_image"

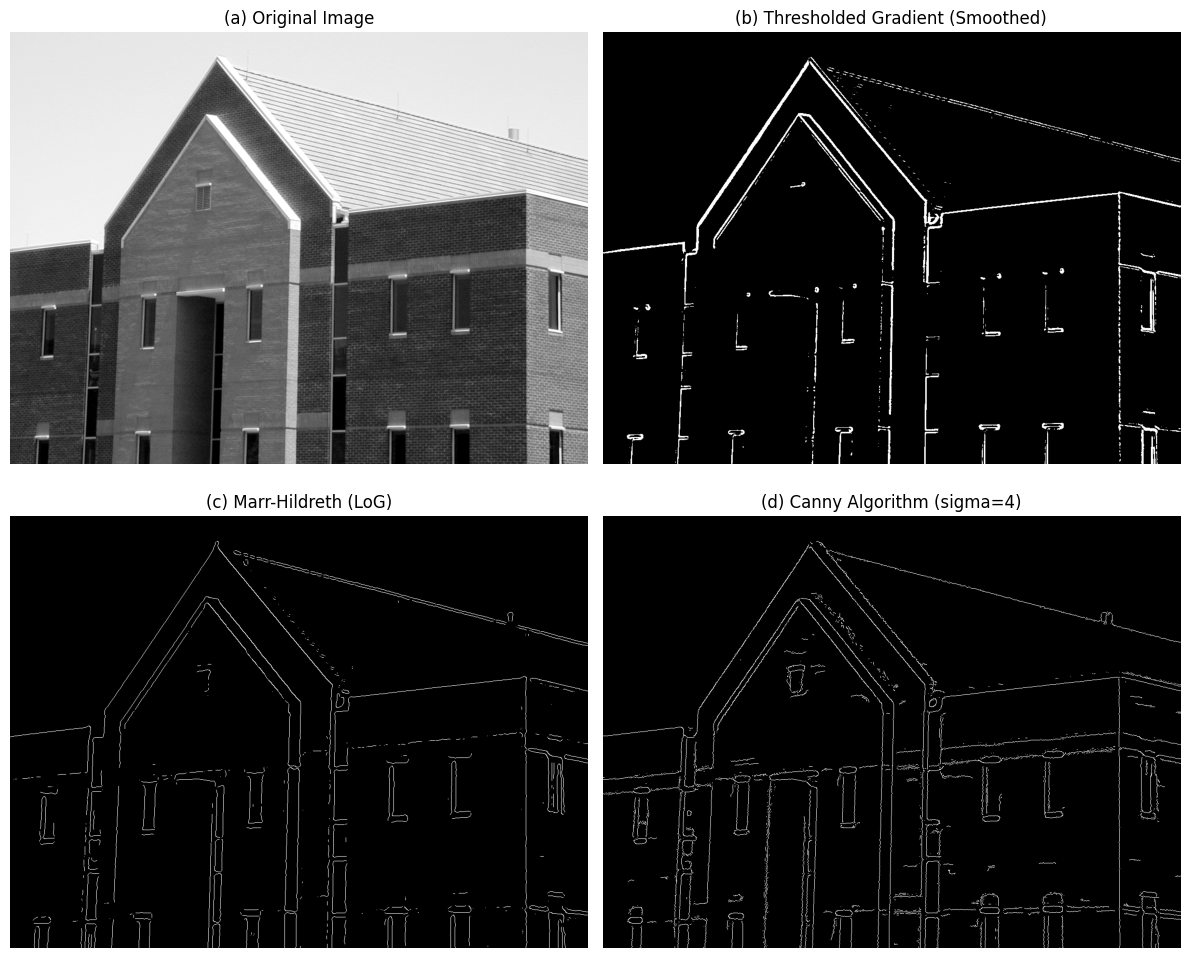
    os.makedirs(output\_dir, exist\_ok**=**True)

    filename **=** "Fig1016(a)(building\_original).tif"

    path **=** os.path.join(input\_dir, filename)

    process\_comparison(path, output\_dir)

**Processed Images：**

****

**Homework15: Implement Figure 10.26 by programming with Python**

**Python Code:**

**import** numpy as np

**import** matplotlib.pyplot as plt

**import** os

**from** scipy **import** ndimage

**from** PIL **import** Image

**def** gaussian\_kernel(size, sigma**=**1):

    size **=** int(size) **//** 2

    x, y **=** np.mgrid[**-**size:size **+** 1, **-**size:size **+** 1]

    normal **=** 1 **/** (2.0 **\*** np.pi **\*** sigma **\*\*** 2)

    g **=** np.exp(**-**((x **\*\*** 2 **+** y **\*\*** 2) **/** (2.0 **\*** sigma **\*\*** 2))) **\*** normal

**return** g

**def** sobel\_filters\_manual(img):

    Kx **=** np.array([[**-**1, 0, 1], [**-**2, 0, 2], [**-**1, 0, 1]], dtype**=**float)

    Ky **=** np.array([[**-**1, **-**2, **-**1], [0, 0, 0], [1, 2, 1]], dtype**=**float)

    Ix **=** ndimage.convolve(img, Kx)

    Iy **=** ndimage.convolve(img, Ky)

    G **=** np.hypot(Ix, Iy)

    theta **=** np.arctan2(Iy, Ix)

**return** G, theta

**def** non\_max\_suppression\_manual(img, D):

    M, N **=** img.shape

    Z **=** np.zeros((M, N), dtype**=**np.float32)

    angle **=** D **\*** 180. **/** np.pi

    angle[angle < 0] **+=** 180

**for** i **in** range(1, M **-** 1):

**for** j **in** range(1, N **-** 1):

**try**:

                q **=** 255

                r **=** 255

**if** (0 <**=** angle[i, j] < 22.5) **or** (157.5 <**=** angle[i, j] <**=** 180):

                    q **=** img[i, j **+** 1]

                    r **=** img[i, j **-** 1]

**elif** (22.5 <**=** angle[i, j] < 67.5):

                    q **=** img[i **+** 1, j **-** 1]

                    r **=** img[i **-** 1, j **+** 1]

**elif** (67.5 <**=** angle[i, j] < 112.5):

                    q **=** img[i **+** 1, j]

                    r **=** img[i **-** 1, j]

**elif** (112.5 <**=** angle[i, j] < 157.5):

                    q **=** img[i **-** 1, j **-** 1]

                    r **=** img[i **+** 1, j **+** 1]

**if** (img[i, j] >**=** q) **and** (img[i, j] >**=** r):

                    Z[i, j] **=** img[i, j]

**else**:

                    Z[i, j] **=** 0

**except** IndexError:

**pass**

**return** Z

**def** hysteresis\_manual(img, weak, strong**=**255):

    M, N **=** img.shape

    strong\_rows, strong\_cols **=** np.where(img **==** strong)

    stack **=** list(zip(strong\_rows, strong\_cols))

**while** stack:

        i, j **=** stack.pop()

**for** di **in** [**-**1, 0, 1]:

**for** dj **in** [**-**1, 0, 1]:

**if** di **==** 0 **and** dj **==** 0: **continue**

                ni, nj **=** i **+** di, j **+** dj

**if** 0 <**=** ni < M **and** 0 <**=** nj < N:

**if** img[ni, nj] **==** weak:

                        img[ni, nj] **=** strong

                        stack.append((ni, nj))

**for** i **in** range(M):

**for** j **in** range(N):

**if** img[i, j] **==** weak:

                img[i, j] **=** 0

**return** img

**def** get\_image\_b\_thresholded\_gradient(img\_norm):

    img\_smoothed **=** ndimage.uniform\_filter(img\_norm, size**=**5)

    Kx **=** np.array([[**-**1, 0, 1], [**-**2, 0, 2], [**-**1, 0, 1]], dtype**=**float)

    Ky **=** np.array([[**-**1, **-**2, **-**1], [0, 0, 0], [1, 2, 1]], dtype**=**float)

    gx **=** ndimage.convolve(img\_smoothed, Kx)

    gy **=** ndimage.convolve(img\_smoothed, Ky)

    magnitude **=** np.hypot(gx, gy)

    thresh\_val **=** 0.15 **\*** np.max(magnitude)

    edges **=** (magnitude >**=** thresh\_val).astype(np.uint8) **\*** 255

**return** edges

**def** get\_image\_c\_marr\_hildreth(img\_norm):

    log\_resp **=** ndimage.gaussian\_laplace(img\_norm, sigma**=**3, truncate**=**3.0)

    threshold **=** 0.002

    edges **=** np.zeros\_like(img\_norm, dtype**=**np.uint8)

    curr\_h **=** log\_resp[:, :**-**1]

    right\_h **=** log\_resp[:, 1:]

    sign\_diff\_h **=** (curr\_h **\*** right\_h) < 0

    mag\_check\_h **=** np.abs(curr\_h **-** right\_h) > threshold

    edges[:, :**-**1][sign\_diff\_h & mag\_check\_h] **=** 255

    curr\_v **=** log\_resp[:**-**1, :]

    down\_v **=** log\_resp[1:, :]

    sign\_diff\_v **=** (curr\_v **\*** down\_v) < 0

    mag\_check\_v **=** np.abs(curr\_v **-** down\_v) > threshold

    edges[:**-**1, :][sign\_diff\_v & mag\_check\_v] **=** 255

**return** edges

**def** get\_image\_d\_manual\_canny(img\_norm):

    kernel **=** gaussian\_kernel(size**=**13, sigma**=**2)

    img\_smoothed **=** ndimage.convolve(img\_norm, kernel)

    grad\_mag, grad\_dir **=** sobel\_filters\_manual(img\_smoothed)

    grad\_mag\_norm **=** grad\_mag **/** np.max(grad\_mag) **if** np.max(grad\_mag) > 0 **else** grad\_mag

    img\_nms **=** non\_max\_suppression\_manual(grad\_mag\_norm, grad\_dir)

    t\_low **=** 0.05

    t\_high **=** 0.15

    weak\_val **=** np.int32(50)

    strong\_val **=** np.int32(255)

    res **=** np.zeros\_like(img\_nms, dtype**=**np.int32)

    strong\_i, strong\_j **=** np.where(img\_nms >**=** t\_high)

    weak\_i, weak\_j **=** np.where((img\_nms <**=** t\_high) & (img\_nms >**=** t\_low))

    res[strong\_i, strong\_j] **=** strong\_val

    res[weak\_i, weak\_j] **=** weak\_val

    img\_final **=** hysteresis\_manual(res, weak\_val, strong\_val)

**return** img\_final

**def** process\_head\_ct(image\_path, output\_dir):

    pil\_img **=** Image.open(image\_path).convert('L')

    img\_norm **=** np.array(pil\_img).astype(float) **/** 255.0

    img\_b **=** get\_image\_b\_thresholded\_gradient(img\_norm)

    img\_c **=** get\_image\_c\_marr\_hildreth(img\_norm)

    img\_d **=** get\_image\_d\_manual\_canny(img\_norm)

    fig, axes **=** plt.subplots(2, 2, figsize**=**(10, 10))

    # fig.suptitle('Edge Detection on Head CT (Fig 10.26)', fontsize=14)

    axes[0, 0].imshow(img\_norm, cmap**=**'gray')

    axes[0, 0].set\_title('(a) Original Image (Head CT)')

    axes[0, 0].axis('off')

    axes[0, 1].imshow(img\_b, cmap**=**'gray')

    axes[0, 1].set\_title('(b) Thresholded Gradient (15% Max)')

    axes[0, 1].axis('off')

    axes[1, 0].imshow(img\_c, cmap**=**'gray')

    axes[1, 0].set\_title('(c) Marr-Hildreth (LoG)')

    axes[1, 0].axis('off')

    axes[1, 1].imshow(img\_d, cmap**=**'gray')

    axes[1, 1].set\_title('(d) Canny Algorithm (Manual)')

    axes[1, 1].axis('off')

    plt.tight\_layout()

    save\_path **=** os.path.join(output\_dir, "Fig1026\_HeadCT\_EdgeDetection.png")

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

    plt.close()

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

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

    output\_dir **=** "output\_image"

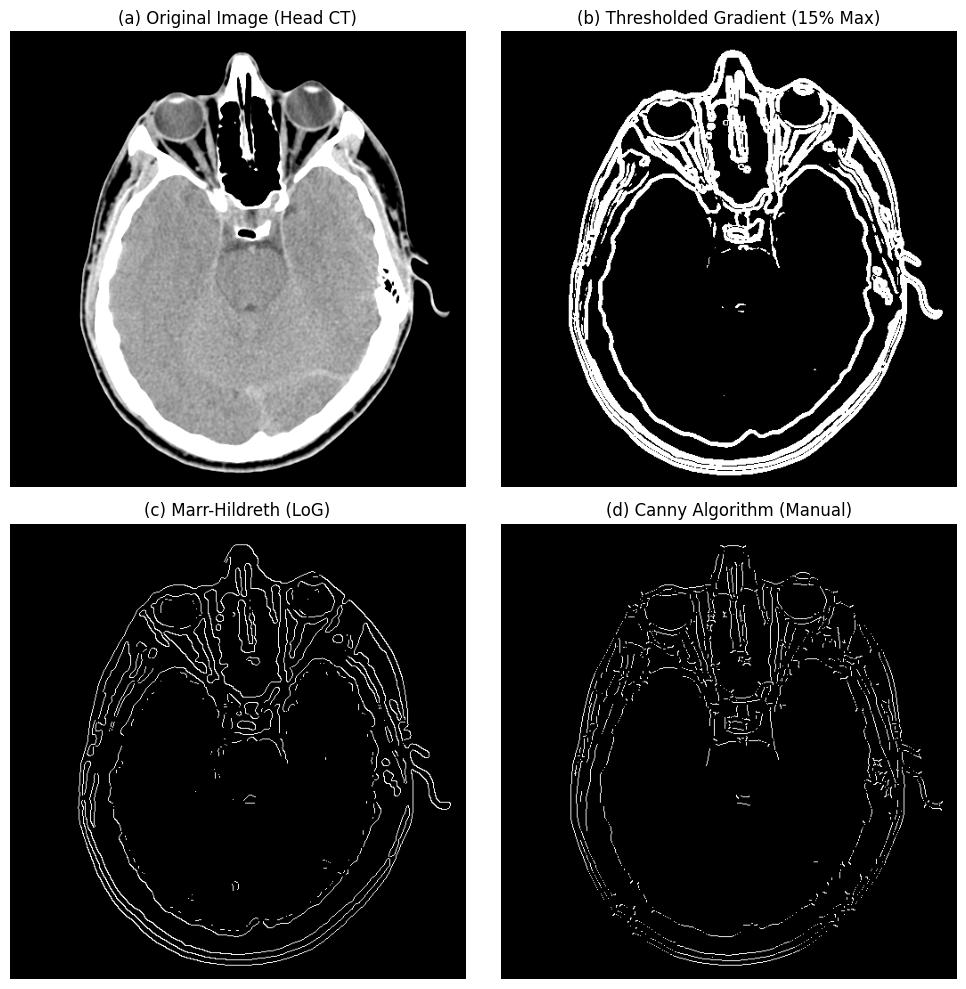
    os.makedirs(output\_dir, exist\_ok**=**True)

    filename **=** "Fig1026(a)(headCT-Vandy).tif"

    path **=** os.path.join(input\_dir, filename)

process\_head\_ct(path, output\_dir)

**Processed Images：**

****