**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 6.29 by programming with Python**

**Python Code:**

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib.pyplot as plt

**import** os

**import** matplotlib.gridspec as gridspec

**def** adjust\_intensity\_rgb(image, k):

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

    processed **=** img\_float **\*** k

    processed **=** np.clip(processed, 0, 1)

**return** (processed **\*** 255).astype(np.uint8)

**def** plot\_mapping\_function(ax, slope**=**None, intercept**=**None, label\_text**=**"", k\_val=0.7, subplot\_label=""):

    x **=** np.linspace(0, 1, 100)

**if** slope **is** **not** None **and** intercept **is** **not** None:

        y **=** slope **\*** x **+** intercept

**elif** slope **is** **not** None:

        y **=** slope **\*** x

**else**:

        y **=** x

    y **=** np.clip(y, 0, 1)

    ax.plot(x, y, 'b-', linewidth**=**2)

    ax.set\_xlim(0, 1)

    ax.set\_ylim(0, 1)

    ax.set\_xticks(np.linspace(0, 1, 5))

    ax.set\_yticks(np.linspace(0, 1, 5))

    ax.tick\_params(axis**=**'both', which**=**'major', labelsize**=**10)

    ax.grid(True, linestyle**=**'-', alpha**=**0.8, color**=**'gray', which**=**'major')

    ax.set\_aspect('equal')

**if** k\_val **is** **not** None:

**if** "R,G,B" **in** label\_text **or** label\_text **==** "I":

            ax.plot([0, 1], [k\_val, k\_val], 'k--', alpha**=**0.3)

            ax.text(**-**0.1, k\_val, r'$k$', va**=**'center', ha**=**'right', fontsize**=**12)

**elif** "C,M,Y" **in** label\_text **and** intercept **is** **not** None **and** intercept > 0:

            val **=** 1 **-** k\_val

            ax.plot([0, 1], [val, val], 'k--', alpha**=**0.3)

            ax.text(**-**0.1, val, r'$1-k$', va**=**'center', ha**=**'right', fontsize**=**12)

    ax.set\_xticklabels([0, '', '', '', 1])

    ax.set\_yticklabels([0, '', '', '', 1])

    ax.tick\_params(axis**=**'y', pad**=**5)

    ax.text(0.95, 0.05, label\_text, transform**=**ax.transAxes,

            ha**=**'right', va**=**'bottom', color**=**'white', backgroundcolor**=**'black', fontsize**=**9, fontweight**=**'bold')

    ax.text(**-**0.25, 1.1, subplot\_label, transform**=**ax.transAxes,

            ha**=**'left', va**=**'top', fontsize**=**12, fontweight**=**'bold')

**def** plot\_image\_subplot(ax, image\_data, subplot\_label**=**""):

    ax.imshow(image\_data)

    ax.axis('off')

    ax.text(**-**0.05, 1.05, subplot\_label, transform**=**ax.transAxes,

            ha**=**'left', va**=**'top', fontsize**=**12, fontweight**=**'bold')

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

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

    output\_dir **=** "output\_image"

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

    image\_filename **=** "Fig0630(01)(strawberries\_fullcolor).tif"

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

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

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

**if** len(original\_image.shape) **==** 2:

        original\_image **=** np.stack((original\_image,) **\*** 3, axis**=-**1)

**elif** original\_image.shape[2] **==** 4:

        original\_image **=** original\_image[:, :, :3]

    k **=** 0.7

    result\_image **=** adjust\_intensity\_rgb(original\_image, k)

    fig **=** plt.figure(figsize**=**(12, 8))

    gs **=** gridspec.GridSpec(2, 6, figure**=**fig, hspace**=**0.4, wspace**=**0.3, height\_ratios**=**[2.5, 1])

    ax\_img1 **=** fig.add\_subplot(gs[0, 0:3])

    plot\_image\_subplot(ax\_img1, original\_image, subplot\_label**=**"a")

    ax\_img2 **=** fig.add\_subplot(gs[0, 3:6])

    plot\_image\_subplot(ax\_img2, result\_image, subplot\_label**=**"b")

    ax\_f1 **=** fig.add\_subplot(gs[1, 0])

    plot\_mapping\_function(ax\_f1, slope**=**k, intercept**=**0, label\_text**=**"R,G,B", k\_val**=**k, subplot\_label**=**"c")

    ax\_f2 **=** fig.add\_subplot(gs[1, 1])

    plot\_mapping\_function(ax\_f2, slope**=**k, intercept**=**1 **-** k, label\_text**=**"C,M,Y", k\_val**=**k, subplot\_label**=**"d")

    ax\_f3 **=** fig.add\_subplot(gs[1, 2])

    plot\_mapping\_function(ax\_f3, slope**=**1, intercept**=**0, label\_text**=**"K", k\_val**=**k, subplot\_label**=**"e")

    ax\_f4 **=** fig.add\_subplot(gs[1, 3])

    plot\_mapping\_function(ax\_f4, slope**=**k, intercept**=**1 **-** k, label\_text**=**"C,M,Y", k\_val**=**k, subplot\_label**=**"f")

    ax\_f5 **=** fig.add\_subplot(gs[1, 4])

    plot\_mapping\_function(ax\_f5, slope**=**k, intercept**=**0, label\_text**=**"I", k\_val**=**k, subplot\_label**=**"g")

    ax\_f6 **=** fig.add\_subplot(gs[1, 5])

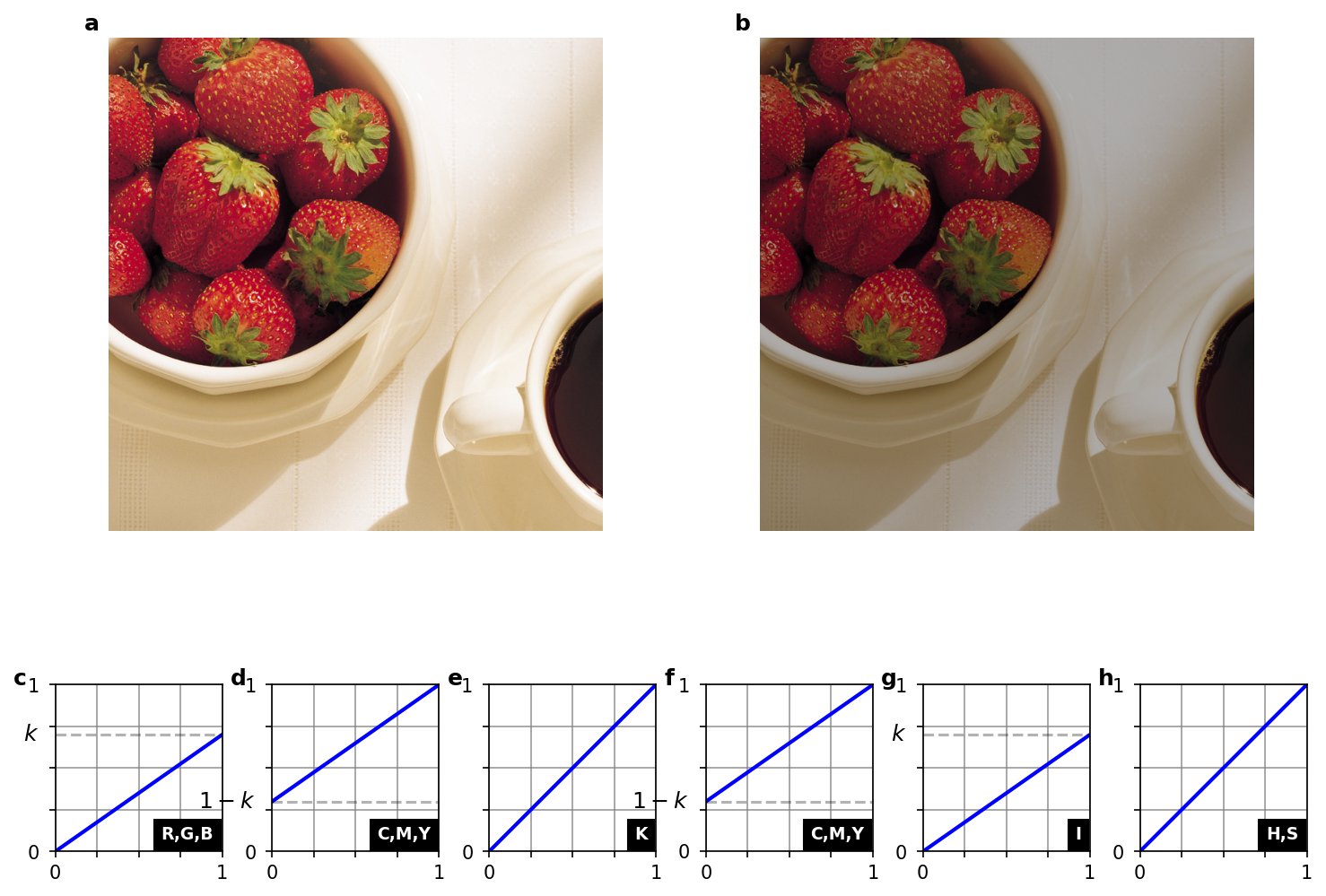
    plot\_mapping\_function(ax\_f6, slope**=**1, intercept**=**0, label\_text**=**"H,S", k\_val**=**k, subplot\_label**=**"h")

    save\_path **=** os.path.join(output\_dir, f"{base\_name}\_fig629\_final.png")

    plt.savefig(save\_path, bbox\_inches**=**'tight', dpi**=**150)

    plt.close(fig)

**Processed Images:**



**Homework2: Implement Figure 6.31 by programming with Python**

**Python Code:**

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib.pyplot as plt

**import** os

**import** matplotlib.gridspec as gridspec

**def** rgb\_complement(image):

**return** 255 **-** image

**def** rgb\_to\_hsi(rgb\_image):

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

    R, G, B **=** img\_float[:, :, 0], img\_float[:, :, 1], img\_float[:, :, 2]

    H **=** np.zeros\_like(R)

    S **=** np.zeros\_like(R)

    I **=** (R **+** G **+** B) **/** 3.0

    min\_rgb **=** np.minimum(np.minimum(R, G), B)

    with warnings.catch\_warnings():

        warnings.simplefilter("ignore", RuntimeWarning)

        S **=** 1 **-** (3.0 **/** (R **+** G **+** B **+** 1e**-**6)) **\*** min\_rgb

    num **=** 0.5 **\*** ((R **-** G) **+** (R **-** B))

    den **=** np.sqrt((R **-** G) **\*\*** 2 **+** (R **-** B) **\*** (G **-** B))

    theta **=** np.arccos(np.clip(num **/** (den **+** 1e**-**6), **-**1, 1))

    theta **=** np.degrees(theta)

    H[B <**=** G] **=** theta[B <**=** G]

    H[B > G] **=** 360 **-** theta[B > G]

    H[S **==** 0] **=** 0

    hsi\_image **=** np.stack([H, S, I], axis**=**2)

**return** hsi\_image

**def** hsi\_to\_rgb(hsi\_image):

    H, S, I **=** hsi\_image[:, :, 0], hsi\_image[:, :, 1], hsi\_image[:, :, 2]

    R, G, B **=** np.zeros\_like(H), np.zeros\_like(H), np.zeros\_like(H)

    idx **=** (H >**=** 0) & (H < 120)

    B[idx] **=** I[idx] **\*** (1 **-** S[idx])

    R[idx] **=** I[idx] **\*** (1 **+** (S[idx] **\*** np.cos(np.radians(H[idx]))) **/** np.cos(np.radians(60 **-** H[idx])))

    G[idx] **=** 3 **\*** I[idx] **-** (R[idx] **+** B[idx])

    idx **=** (H >**=** 120) & (H < 240)

    H\_prime **=** H[idx] **-** 120

    R[idx] **=** I[idx] **\*** (1 **-** S[idx])

    G[idx] **=** I[idx] **\*** (1 **+** (S[idx] **\*** np.cos(np.radians(H\_prime))) **/** np.cos(np.radians(60 **-** H\_prime)))

    B[idx] **=** 3 **\*** I[idx] **-** (R[idx] **+** G[idx])

    idx **=** (H >**=** 240) & (H < 360)

    H\_prime **=** H[idx] **-** 240

    G[idx] **=** I[idx] **\*** (1 **-** S[idx])

    B[idx] **=** I[idx] **\*** (1 **+** (S[idx] **\*** np.cos(np.radians(H\_prime))) **/** np.cos(np.radians(60 **-** H\_prime)))

    R[idx] **=** 3 **\*** I[idx] **-** (G[idx] **+** B[idx])

    rgb\_image **=** np.stack([R, G, B], axis**=**2)

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

**return** rgb\_image

**def** hsi\_transform\_from\_book(image):

    hsi\_img **=** rgb\_to\_hsi(image)

    H, S, I **=** hsi\_img[:, :, 0], hsi\_img[:, :, 1], hsi\_img[:, :, 2]

    H\_transformed **=** (H **+** 180) **%** 360

    S\_transformed **=** S

    I\_transformed **=** 1.0 **-** I

    hsi\_transformed **=** np.stack([H\_transformed, S\_transformed, I\_transformed], axis**=**2)

**return** hsi\_to\_rgb(hsi\_transformed)

**def** plot\_transform\_functions(fig, spec):

    """

    Plots the four transformation functions

    """

    gs\_nested **=** gridspec.GridSpecFromSubplotSpec(2, 2, subplot\_spec**=**spec, wspace**=**0.1, hspace**=**0.1)

    x **=** np.linspace(0, 1, 100)

    plots\_def **=** {

        '00': {'type': 'rgb', 'label': 'R,G,B'},

        '01': {'type': 'h', 'label': 'H'},

        '10': {'type': 's', 'label': 'S'},

        '11': {'type': 'i', 'label': 'I'}

    }

**for** i **in** range(2):

**for** j **in** range(2):

            ax **=** fig.add\_subplot(gs\_nested[i, j])

            key **=** f'{i}{j}'

            p\_def **=** plots\_def[key]

**if** p\_def['type'] **==** 'rgb' **or** p\_def['type'] **==** 'i':

                ax.plot([0, 1], [1, 0], 'k')

**elif** p\_def['type'] **==** 'h':

                ax.plot([0, 0.5], [0.5, 1], 'k')

                ax.plot([0.5, 1], [0, 0.5], 'k')

**elif** p\_def['type'] **==** 's':

                ax.plot([0, 1], [0, 1], 'k')

            ax.set\_aspect('equal', adjustable**=**'box')

            ax.set\_xlim(0, 1)

            ax.set\_ylim(0, 1)

            ticks **=** np.linspace(0, 1, 6)

            ax.set\_xticks(ticks)

            ax.set\_yticks(ticks)

            ax.set\_xticklabels(['0', '', '', '', '', '1'])

            ax.set\_yticklabels(['0', '', '', '', '', '1'])

            ax.grid(True, color**=**'black', linewidth**=**0.75)

            ax.tick\_params(length**=**0)

            ax.text(0.25, 0.15, p\_def['label'], ha**=**'center', va**=**'center',

                    bbox**=**{'facecolor': 'darkgray', 'edgecolor': 'black', 'pad': 4})

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

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

    output\_dir **=** "output\_image"

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

    image\_filename **=** "Fig0630(01)(strawberries\_fullcolor).tif"

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

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

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

    rgb\_comp\_image **=** rgb\_complement(original\_image)

    hsi\_comp\_image **=** hsi\_transform\_from\_book(original\_image)

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

    fig.suptitle(f'Color Complement Transformations (Fig 6.31) - {image\_filename}', fontsize**=**16)

    gs\_main **=** gridspec.GridSpec(2, 2, figure**=**fig, wspace**=**0.05, hspace**=**0.15)

    ax\_a **=** fig.add\_subplot(gs\_main[0, 0])

    ax\_a.imshow(original\_image)

    ax\_a.set\_title('a) Original Image')

    ax\_a.axis('off')

    plot\_transform\_functions(fig, gs\_main[0, 1])

    ax\_b\_title **=** fig.add\_subplot(gs\_main[0, 1])

    ax\_b\_title.set\_title('b) Transformation Functions')

    ax\_b\_title.axis('off')

    ax\_c **=** fig.add\_subplot(gs\_main[1, 0])

    ax\_c.imshow(rgb\_comp\_image)

    ax\_c.set\_title('c) RGB Complement')

    ax\_c.axis('off')

    ax\_d **=** fig.add\_subplot(gs\_main[1, 1])

    ax\_d.imshow(hsi\_comp\_image)

    ax\_d.set\_title('d) HSI-based Complement')

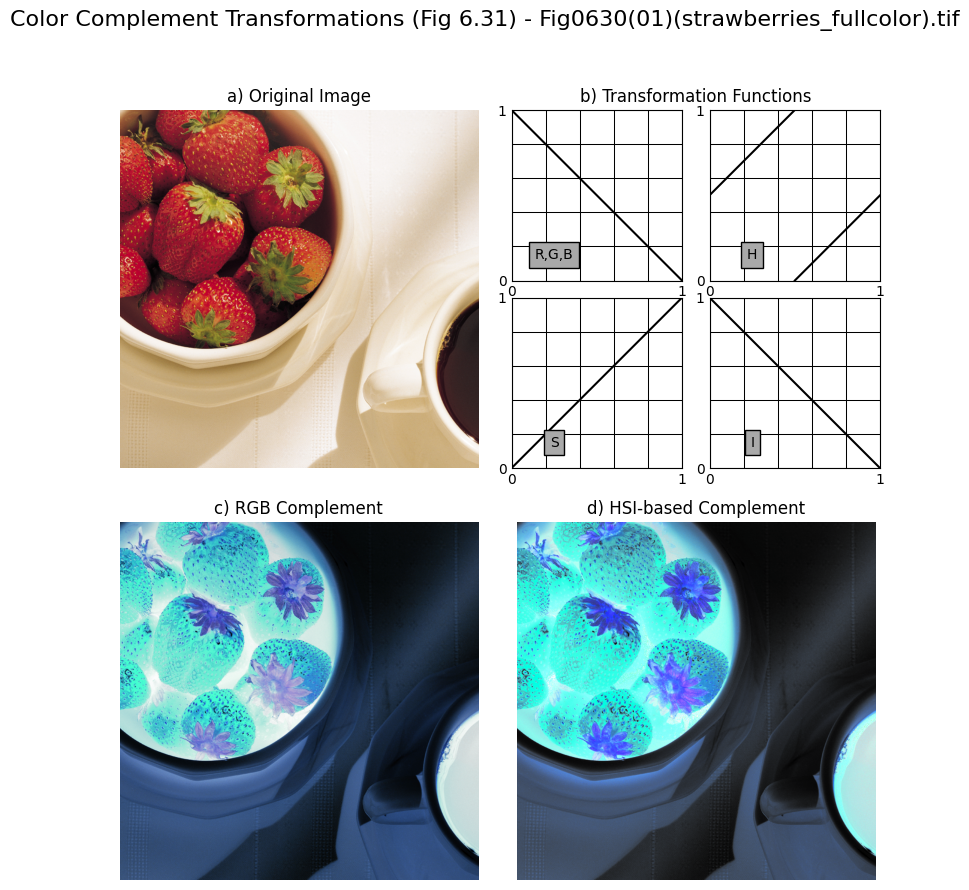
    ax\_d.axis('off')

    combined\_output\_path **=** os.path.join(output\_dir, f"{base\_name}\_complement\_full\_figure.png")

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

    plt.close()

**Processed Images:**



**Homework3: Implement Figure 6.31 by programming with Python**

**Python Code:**

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib.pyplot as plt

**import** os

**def** color\_slicing\_cube(image, center\_color, width, neutral\_color**=**(0.5, 0.5, 0.5)):

    img\_float **=** image.astype(np.float64) **/** 255.0

    center **=** np.array(center\_color)

    diff **=** np.abs(img\_float **-** center)

    mask **=** np.any(diff > width **/** 2.0, axis**=**2)

    result\_image **=** np.copy(image)

    neutral\_pixel **=** (np.array(neutral\_color) **\*** 255).astype(np.uint8)

    result\_image[mask] **=** neutral\_pixel

**return** result\_image

**def** color\_slicing\_sphere(image, center\_color, radius, neutral\_color**=**(0.5, 0.5, 0.5)):

    img\_float **=** image.astype(np.float64) **/** 255.0

    center **=** np.array(center\_color)

    distances\_sq **=** np.sum((img\_float **-** center) **\*\*** 2, axis**=**2)

    mask **=** distances\_sq > radius **\*\*** 2

    result\_image **=** np.copy(image)

    neutral\_pixel **=** (np.array(neutral\_color) **\*** 255).astype(np.uint8)

    result\_image[mask] **=** neutral\_pixel

**return** result\_image

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

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

    output\_dir **=** "output\_image"

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

    image\_filename **=** "Fig0630(01)(strawberries\_fullcolor).tif"

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

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

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

    center\_color\_a **=** (0.6863, 0.1608, 0.1922)

    width\_W **=** 0.2549

    radius\_R0 **=** 0.1765

    cube\_sliced\_image **=** color\_slicing\_cube(original\_image, center\_color\_a, width\_W)

    sphere\_sliced\_image **=** color\_slicing\_sphere(original\_image, center\_color\_a, radius\_R0)

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

    fig.suptitle(f'Color Slicing (Fig 6.32) - {image\_filename}', fontsize**=**16)

    axes[0].imshow(cube\_sliced\_image)

    axes[0].set\_title(f'a) Cube transform, $W={width\_W}$')

    axes[0].axis('off')

    axes[1].imshow(sphere\_sliced\_image)

    axes[1].set\_title(f'b) Sphere transform, $R\_0={radius\_R0}$')

    axes[1].axis('off')

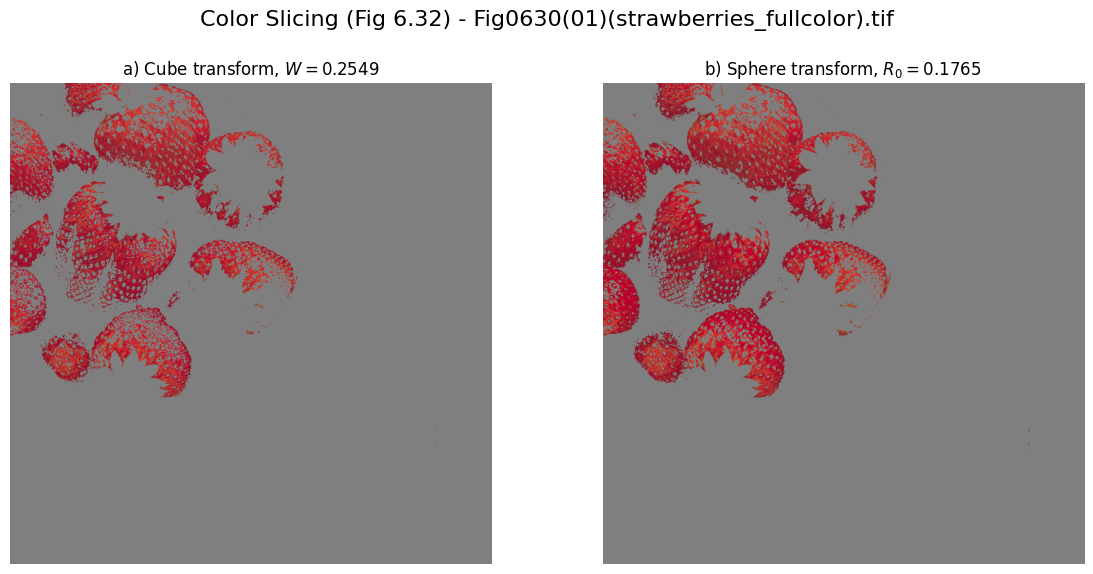
    plt.tight\_layout(rect**=**[0, 0.03, 1, 0.95])

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

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

    plt.close()

**Processed Images:**



**Homework4: Implement Figure 6.33 by programming with Python**

**Python Code:**

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib.pyplot as plt

**import** os

**def** tone\_correction(image, contrast, midpoint):

    img\_float **=** image.astype(np.float64) **/** 255.0

    epsilon **=** 1e**-**6

    corrected\_float **=** 1 **/** (1 **+** (midpoint **/** (img\_float **+** epsilon)) **\*\*** contrast)

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

**return** corrected\_image

**def** plot\_tone\_curve(ax, contrast, midpoint):

    r **=** np.linspace(0, 1, 256)

    epsilon **=** 1e**-**6

    s **=** 1 **/** (1 **+** (midpoint **/** (r **+** epsilon)) **\*\*** contrast)

    ax.plot(r, s, 'k')

    ax.set\_aspect('equal', adjustable**=**'box')

    ax.set\_xlim(0, 1)

    ax.set\_ylim(0, 1)

    ticks **=** np.linspace(0, 1, 5)  # For 4 squares

    ax.set\_xticks(ticks)

    ax.set\_yticks(ticks)

    ax.set\_xticklabels(['0', '', '', '', '1'])

    ax.set\_yticklabels(['0', '', '', '', '1'])

    ax.grid(True, color**=**'black', linewidth**=**0.75)

    ax.tick\_params(length**=**0)  # Remove tick marks

    ax.text(0.5, 0.2, 'R,G,B', ha**=**'center', va**=**'center',

            bbox**=**{'facecolor': 'white', 'edgecolor': 'black', 'pad': 4})

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

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

    output\_dir **=** "output\_image"

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

    image\_filename **=** "Fig0635(top\_ left\_flower).tif"

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

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

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

    contrast\_E **=** 10

    midpoint\_m **=** 0.5

    corrected\_image **=** tone\_correction(original\_image, contrast\_E, midpoint\_m)

    fig **=** plt.figure(figsize**=**(18, 7))

    gs **=** fig.add\_gridspec(1, 3, width\_ratios**=**[4, 4, 1.5], wspace**=**0.05)

    fig.suptitle(f'Tone Correction (Fig 6.33) - {image\_filename}', fontsize**=**16)

    ax0 **=** fig.add\_subplot(gs[0])

    ax0.imshow(original\_image)

    ax0.set\_title('a) Flat Image')

    ax0.axis('off')

    ax1 **=** fig.add\_subplot(gs[1])

    ax1.imshow(corrected\_image)

    ax1.set\_title('b) Corrected Image')

    ax1.axis('off')

    ax2 **=** fig.add\_subplot(gs[2])

    plot\_tone\_curve(ax2, contrast\_E, midpoint\_m)

    ax2.set\_title('c) Transformation')

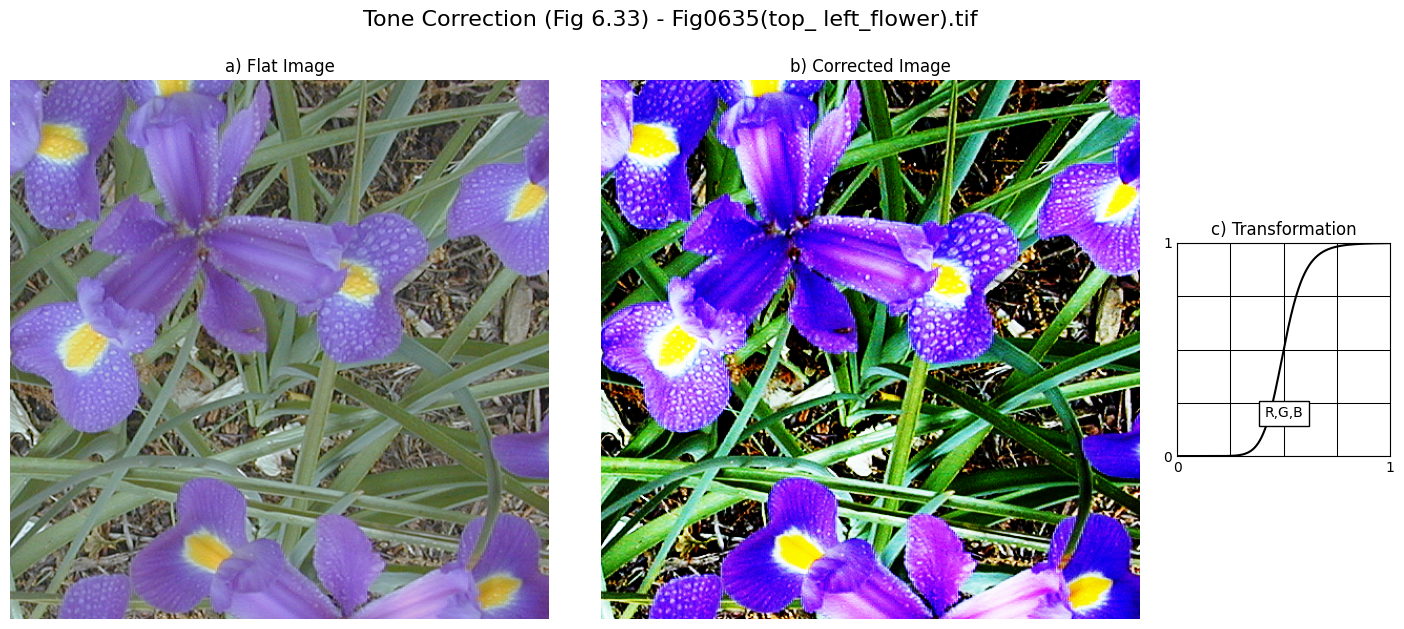
    plt.tight\_layout(rect**=**[0, 0.03, 1, 0.95])

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

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

    plt.close()

**Processed Images:**



**Homework5: Implement Figure 6.35 by programming with Python**

**Python Code:**

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib.pyplot as plt

**import** os

**import** warnings

**import** matplotlib.gridspec as gridspec

**def** rgb\_to\_hsi(rgb\_image):

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

    R, G, B **=** img\_float[:, :, 0], img\_float[:, :, 1], img\_float[:, :, 2]

    H **=** np.zeros\_like(R)

    S **=** np.zeros\_like(R)

    I **=** (R **+** G **+** B) **/** 3.0

    min\_rgb **=** np.minimum(np.minimum(R, G), B)

    with warnings.catch\_warnings():

        warnings.simplefilter("ignore", RuntimeWarning)

        S **=** 1 **-** (3.0 **/** (R **+** G **+** B **+** 1e**-**6)) **\*** min\_rgb

    num **=** 0.5 **\*** ((R **-** G) **+** (R **-** B))

    den **=** np.sqrt((R **-** G) **\*\*** 2 **+** (R **-** B) **\*** (G **-** B))

    with warnings.catch\_warnings():

        warnings.simplefilter("ignore", RuntimeWarning)

        theta **=** np.arccos(np.clip(num **/** (den **+** 1e**-**6), **-**1, 1))

    theta **=** np.degrees(theta)

    H[B <**=** G] **=** theta[B <**=** G]

    H[B > G] **=** 360 **-** theta[B > G]

    H[S **==** 0] **=** 0

    hsi\_image **=** np.stack([H, S, I], axis**=**2)

**return** hsi\_image

**def** hsi\_to\_rgb(hsi\_image):

    H, S, I **=** hsi\_image[:, :, 0], hsi\_image[:, :, 1], hsi\_image[:, :, 2]

    R, G, B **=** np.zeros\_like(H), np.zeros\_like(H), np.zeros\_like(H)

    idx **=** (H >**=** 0) & (H < 120)

    B[idx] **=** I[idx] **\*** (1 **-** S[idx])

    R[idx] **=** I[idx] **\*** (1 **+** (S[idx] **\*** np.cos(np.radians(H[idx]))) **/** np.cos(np.radians(60 **-** H[idx])))

    G[idx] **=** 3 **\*** I[idx] **-** (R[idx] **+** B[idx])

    idx **=** (H >**=** 120) & (H < 240)

    H\_prime **=** H[idx] **-** 120

    R[idx] **=** I[idx] **\*** (1 **-** S[idx])

    G[idx] **=** I[idx] **\*** (1 **+** (S[idx] **\*** np.cos(np.radians(H\_prime))) **/** np.cos(np.radians(60 **-** H\_prime)))

    B[idx] **=** 3 **\*** I[idx] **-** (R[idx] **+** G[idx])

    idx **=** (H >**=** 240) & (H < 360)

    H\_prime **=** H[idx] **-** 240

    G[idx] **=** I[idx] **\*** (1 **-** S[idx])

    B[idx] **=** I[idx] **\*** (1 **+** (S[idx] **\*** np.cos(np.radians(H\_prime))) **/** np.cos(np.radians(60 **-** H\_prime)))

    R[idx] **=** 3 **\*** I[idx] **-** (G[idx] **+** B[idx])

    rgb\_image **=** np.stack([R, G, B], axis**=**2)

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

**return** rgb\_image

**def** histogram\_equalization(intensity\_channel):

    I\_uint8 **=** np.clip(intensity\_channel **\*** 255, 0, 255).astype(np.uint8)

    hist, \_ **=** np.histogram(I\_uint8.flatten(), 256, [0, 256])

    cdf **=** hist.cumsum()

    cdf\_m **=** np.ma.masked\_equal(cdf, 0)

    cdf\_m **=** (cdf\_m **-** cdf\_m.min()) **\*** 255 **/** (cdf\_m.max() **-** cdf\_m.min())

    cdf **=** np.ma.filled(cdf\_m, 0).astype('uint8')

    equalized\_I\_uint8 **=** cdf[I\_uint8]

    equalized\_I\_float **=** equalized\_I\_uint8.astype(float) **/** 255.0

    equalized\_hist, \_ **=** np.histogram(equalized\_I\_uint8.flatten(), 256, [0, 256])

    transform\_func **=** cdf **/** 255.0

**return** equalized\_I\_float, hist, equalized\_hist, transform\_func

**def** plot\_panel\_b(fig, spec, hist\_orig, hist\_eq, transform\_func, median\_orig, median\_eq):

    gs\_nested **=** gridspec.GridSpecFromSubplotSpec(2, 2, subplot\_spec**=**spec, wspace**=**0.2, hspace**=**0.2)

**def** style\_ax(ax):

        ax.set\_aspect('equal', adjustable**=**'box')

        ax.set\_xlim(0, 1)

        ax.set\_ylim(0, 1)

        ticks **=** np.linspace(0, 1, 5)

        ax.set\_xticks(ticks)

        ax.set\_yticks(ticks)

        ax.set\_xticklabels(['0', '', '', '', '1'])

        ax.set\_yticklabels(['0', '', '', '', '1'])

        ax.grid(True, color**=**'black', linewidth**=**0.75)

        ax.tick\_params(length**=**0)

    ax\_h **=** fig.add\_subplot(gs\_nested[0, 0])

    ax\_h.plot([0, 1], [0, 1], 'k')

    ax\_h.text(0.5, 0.5, 'H', ha**=**'center', va**=**'center', bbox**=**{'facecolor': 'white', 'edgecolor': 'black'})

    style\_ax(ax\_h)

    ax\_s **=** fig.add\_subplot(gs\_nested[0, 1])

    ax\_s.plot([0, 1], [0, 1], 'k')

    ax\_s.text(0.5, 0.5, 'S', ha**=**'center', va**=**'center', bbox**=**{'facecolor': 'white', 'edgecolor': 'black'})

    style\_ax(ax\_s)

    ax\_i **=** fig.add\_subplot(gs\_nested[1, 0])

    r **=** np.linspace(0, 1, 256)

    ax\_i.plot(r, transform\_func, 'k')

    ax\_i.plot([median\_orig, median\_orig], [0, median\_eq], 'k--')

    ax\_i.plot([0, median\_orig], [median\_eq, median\_eq], 'k--')

    ax\_i.text(0.5, 0.5, 'I', ha**=**'center', va**=**'center', bbox**=**{'facecolor': 'white', 'edgecolor': 'black'})

    style\_ax(ax\_i)

    # Specific labels for I plot

    ax\_i.set\_xticklabels(['0', '', '', '', '1'])

    ax\_i.set\_yticklabels(['0', '', '', '', '1'])

    ax\_i.set\_xlabel(f'{median\_orig:.2f}', labelpad**=-**10)

    ax\_i.set\_ylabel(f'{median\_eq:.2f}', labelpad**=-**10, rotation**=**0, ha**=**'right')

    gs\_hist **=** gridspec.GridSpecFromSubplotSpec(2, 1, subplot\_spec**=**gs\_nested[1, 1], hspace**=**0.1)

    ax\_hist1 **=** fig.add\_subplot(gs\_hist[0])

    ax\_hist2 **=** fig.add\_subplot(gs\_hist[1])

    ax\_hist1.bar(range(256), hist\_orig, color**=**'black', width**=**1.0)

    ax\_hist1.set\_title(f'Before (median={median\_orig:.2f})', fontsize**=**8)

    ax\_hist2.bar(range(256), hist\_eq, color**=**'black', width**=**1.0)

    ax\_hist2.set\_title(f'After (median={median\_eq:.2f})', fontsize**=**8)

**for** ax **in** [ax\_hist1, ax\_hist2]:

        ax.set\_yticks([])

        ax.set\_xticks([])

        ax.set\_xlim(0, 255)

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

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

    output\_dir **=** "output\_image"

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

    image\_filename **=** "Fig0637(a)(caster\_stand\_original).tif"

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

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

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

    hsi\_orig **=** rgb\_to\_hsi(original\_image)

    H\_orig, S\_orig, I\_orig **=** hsi\_orig[:, :, 0], hsi\_orig[:, :, 1], hsi\_orig[:, :, 2]

    I\_eq, hist\_orig, hist\_eq, transform\_func **=** histogram\_equalization(I\_orig)

    hsi\_c **=** np.stack([H\_orig, S\_orig, I\_eq], axis**=**2)

    image\_c **=** hsi\_to\_rgb(hsi\_c)

    S\_d\_pre **=** S\_orig **\*\*** 0.75

    I\_d\_pre **=** I\_orig **\*\*** 0.75

    I\_d\_eq, \_, \_, \_ **=** histogram\_equalization(I\_d\_pre)

    hsi\_d **=** np.stack([H\_orig, S\_d\_pre, I\_d\_eq], axis**=**2)

    image\_d **=** hsi\_to\_rgb(hsi\_d)

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

    gs\_main **=** gridspec.GridSpec(2, 2, figure**=**fig, wspace**=**0.05, hspace**=**0.3)

    fig.suptitle(f'HSI Histogram Equalization (Fig 6.35) - {image\_filename}', fontsize**=**16)

    ax\_a **=** fig.add\_subplot(gs\_main[0, 0])

    ax\_a.imshow(original\_image)

    ax\_a.set\_title('a) Original Image')

    ax\_a.axis('off')

    median\_orig **=** np.median(I\_orig)

    median\_eq **=** np.median(I\_eq)

    plot\_panel\_b(fig, gs\_main[0, 1], hist\_orig, hist\_eq, transform\_func, median\_orig, median\_eq)

    ax\_b\_title **=** fig.add\_subplot(gs\_main[0, 1])

    ax\_b\_title.set\_title('b) Transformations and Histograms')

    ax\_b\_title.axis('off')

    ax\_c **=** fig.add\_subplot(gs\_main[1, 0])

    ax\_c.imshow(image\_c)

    ax\_c.set\_title('c) Intensity Equalized')

    ax\_c.axis('off')

    ax\_d **=** fig.add\_subplot(gs\_main[1, 1])

    ax\_d.imshow(image\_d)

    ax\_d.set\_title('d) Sat. & Int. Adjusted, then Int. Equalized')

    ax\_d.axis('off')

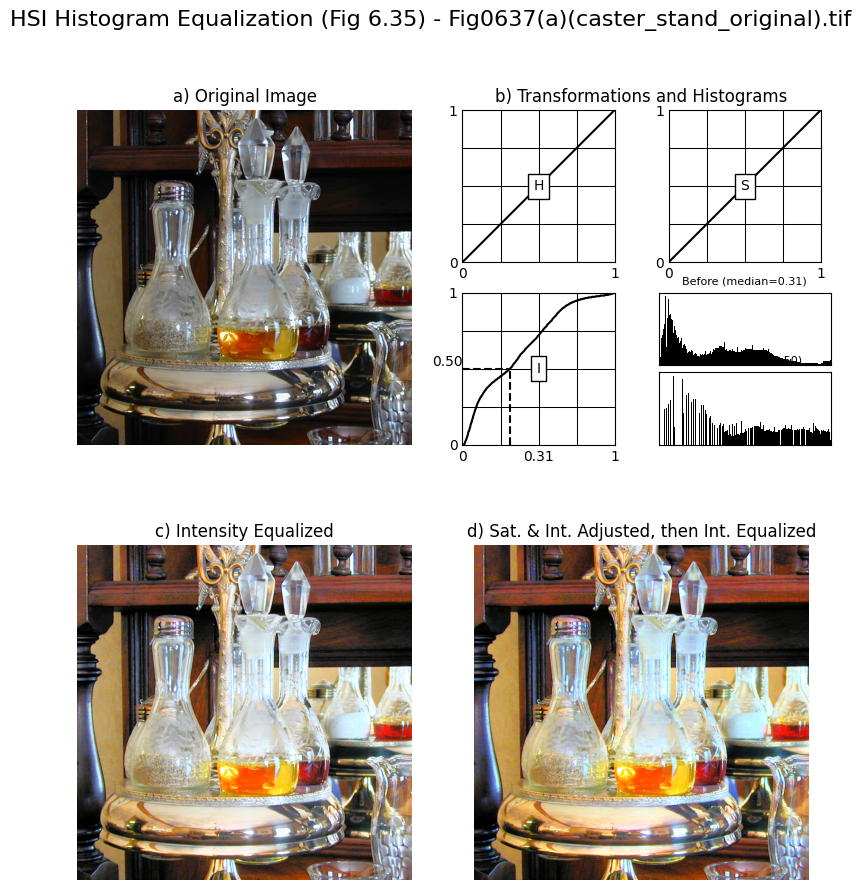
    plt.tight\_layout(rect**=**[0, 0.03, 1, 0.95])

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

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

    plt.close()

**Processed Images:**



**Homework6: Implement Figure 6.44 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** vector\_gradient(image):

    img\_float **=** image.astype(np.float64)

    g\_xx **=** np.zeros\_like(img\_float[:, :, 0])

    g\_yy **=** np.zeros\_like(img\_float[:, :, 0])

**for** i **in** range(3):

        channel **=** img\_float[:, :, i]

        g\_x **=** ndimage.sobel(channel, axis**=**1)

        g\_y **=** ndimage.sobel(channel, axis**=**0)

        g\_xx **+=** g\_x **\*\*** 2

        g\_yy **+=** g\_y **\*\*** 2

    M **=** np.sqrt(g\_xx **+** g\_yy)

**return** M

**def** sum\_of\_gradients(image):

    img\_float **=** image.astype(np.float64)

    M\_sum **=** np.zeros\_like(img\_float[:, :, 0])

**for** i **in** range(3):

        channel **=** img\_float[:, :, i]

        g\_x **=** ndimage.sobel(channel, axis**=**1)

        g\_y **=** ndimage.sobel(channel, axis**=**0)

        M\_sum **+=** np.sqrt(g\_x **\*\*** 2 **+** g\_y **\*\*** 2)

**return** M\_sum

**def** scale\_to\_uint8(image\_float):

    max\_val **=** np.max(image\_float)

**if** max\_val > 0:

        scaled\_image **=** (image\_float **/** max\_val) **\*** 255

**else**:

        scaled\_image **=** image\_float

**return** scaled\_image.astype(np.uint8)

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

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

    output\_dir **=** "output\_image"

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

    image\_filename **=** "Fig0646(a)(lenna\_original\_RGB).tif"

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

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

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

    grad\_vector **=** vector\_gradient(original\_image)

    grad\_sum **=** sum\_of\_gradients(original\_image)

    grad\_diff **=** np.abs(grad\_vector **-** grad\_sum)

    display\_vector **=** scale\_to\_uint8(grad\_vector)

    display\_sum **=** scale\_to\_uint8(grad\_sum)

    display\_diff **=** scale\_to\_uint8(grad\_diff)

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

    fig.suptitle(f'RGB Edge Detection (Fig 6.44) - {image\_filename}', fontsize**=**16)

    axes[0, 0].imshow(original\_image)

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

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

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

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

    axes[1, 0].set\_title('c) Sum of Individual Gradients')

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

    axes[1, 1].set\_title('d) Difference (b) - (c)')

**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"{base\_name}\_rgb\_edge\_detection\_results.png")

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

    plt.close()

**Processed Images:**

