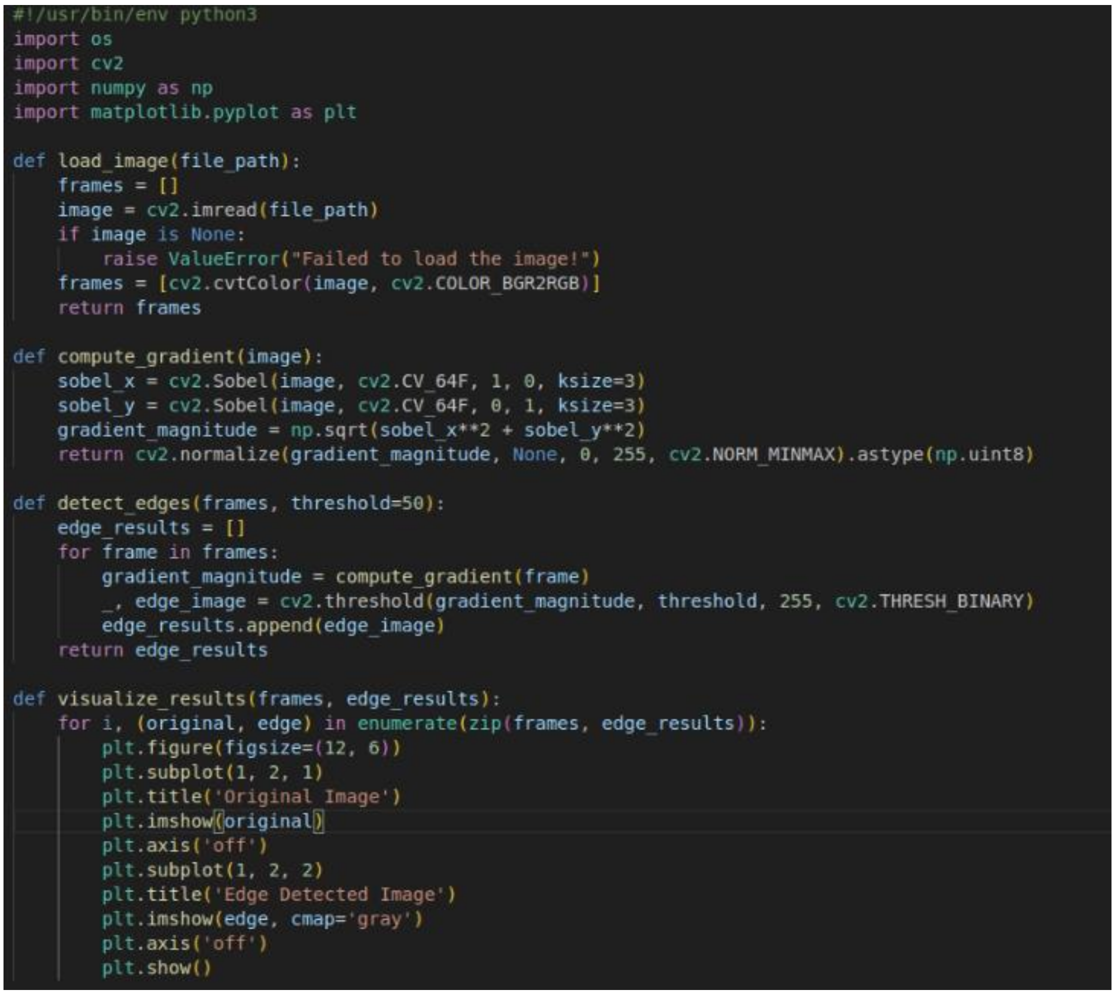
1. Please use a gradient computed in RGB color vector space to detect the edge for the image, 'lenna-RGB.tif' as Fig. 6.44(b) in pp.453. Please describe your method, procedures, final gradient image and print out the source code? (3X10=40)

Code:



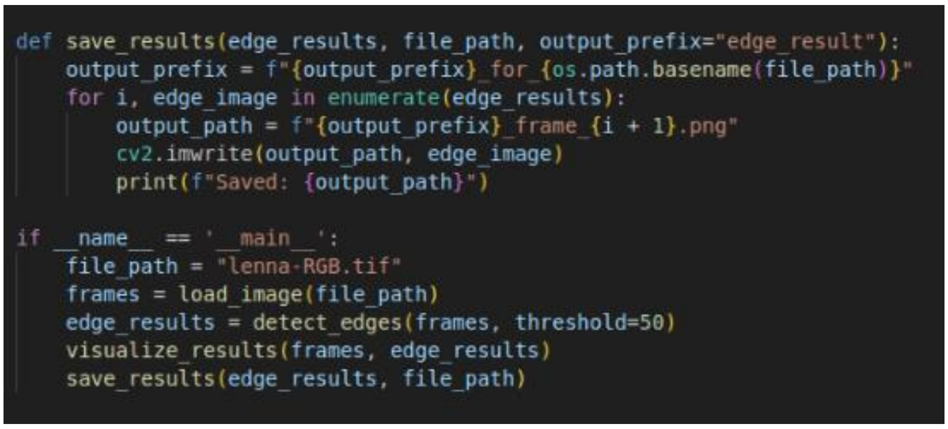
```python
#!/usr/bin/env python3
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt

def load_image(file_path):
    frames = []
    image = cv2.imread(file_path)
    if image is None:
        raise ValueError("Failed to load the image!")
    frames = [cv2.cvtColor(image, cv2.COLOR_BGR2RGB)]
    return frames

def compute_gradient(image):
    sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
    gradient_magnitude = np.sqrt(sobel_x**2 + sobel_y**2)
    return cv2.normalize(gradient_magnitude, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

def detect_edges(frames, threshold=50):
    edge_results = []
    for frame in frames:
        gradient_magnitude = compute_gradient(frame)
        _, edge_image = cv2.threshold(gradient_magnitude, threshold, 255, cv2.THRESH_BINARY)
        edge_results.append(edge_image)
    return edge_results

def visualize_results(frames, edge_results):
    for i, (original, edge) in enumerate(zip(frames, edge_results)):
        plt.figure(figsize=(12, 6))
        plt.subplot(1, 2, 1)
        plt.title('Original Image')
        plt.imshow(original)
        plt.axis('off')
        plt.subplot(1, 2, 2)
        plt.title('Edge Detected Image')
        plt.imshow(edge, cmap='gray')
        plt.axis('off')
        plt.show()
```



```python
def save_results(edge_results, file_path, output_prefix="edge_result"):
    output_prefix = f"{output_prefix}_for_{os.path.basename(file_path)}"
    for i, edge_image in enumerate(edge_results):
        output_path = f"{output_prefix}_frame_{i + 1}.png"
        cv2.imwrite(output_path, edge_image)
        print(f"Saved: {output_path}")

if __name__ == '__main__':
    file_path = "lenna-RGB.tif"
    frames = load_image(file_path)
    edge_results = detect_edges(frames, threshold=50)
    visualize_results(frames, edge_results)
    save_results(edge_results, file_path)
```

Method Description

This program detects edges in an RGB image (lenna-RGB.tif) using a gradient-based approach in the RGB color space. It applies the Sobel operator to compute gradients along the x and y directions for each channel of the RGB image and calculates the gradient magnitude to identify edges. The process is completed by thresholding the gradient magnitude to produce a binary edge image.

Procedures

1. **Load the Image**:
   - The image is loaded using OpenCV's cv2.imread and converted from BGR to RGB format for proper color representation.
   - The loaded RGB image is stored as a frame for processing.
2. **Compute Gradient Magnitude**:
   - Sobel operators are applied separately to the x and y directions for each channel in the RGB image using cv2.Sobel.
   - The gradients are combined into a single gradient magnitude using the formula:

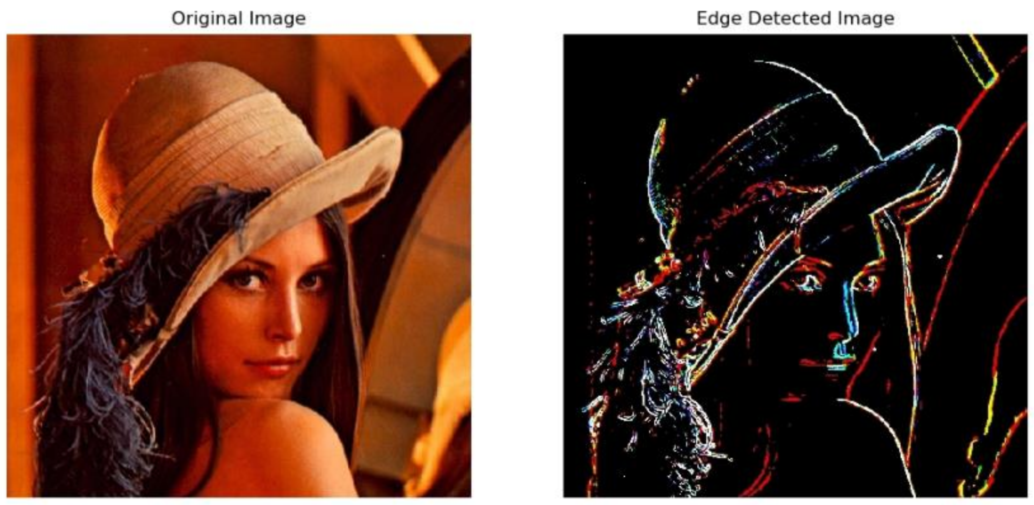
     $$G = \sqrt{G_x^2 + G_y^2}$$
   - The resulting gradient magnitude is normalized to the range [0, 255].
3. **Edge Detection**:
   - A thresholding operation is applied to the gradient magnitude. Pixels with values greater than the threshold (default: 50) are marked as edges (255), while others are set to 0.
4. **Visualization**:
   - The original image and the detected edge map are displayed side-by-side using Matplotlib.
5. **Saving Results**:

- The edge-detected images are saved as separate files with a prefix indicating the operation and original file name.

Final Gradient Image

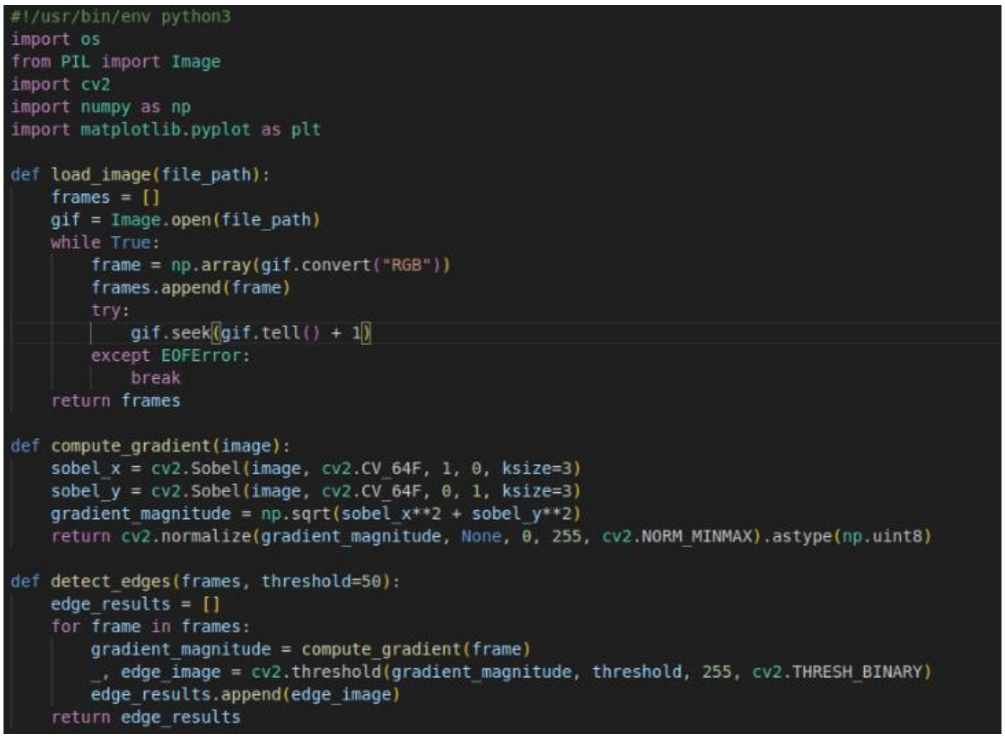
The resulting gradient image highlights edges in the RGB image based on intensity changes across all color channels. The edges are shown as binary (black and white), with white pixels indicating detected edges.

Result:



2. Repeat (1) steps in the image ‘Visual resolution.gif’? (3X10=40)

Code:

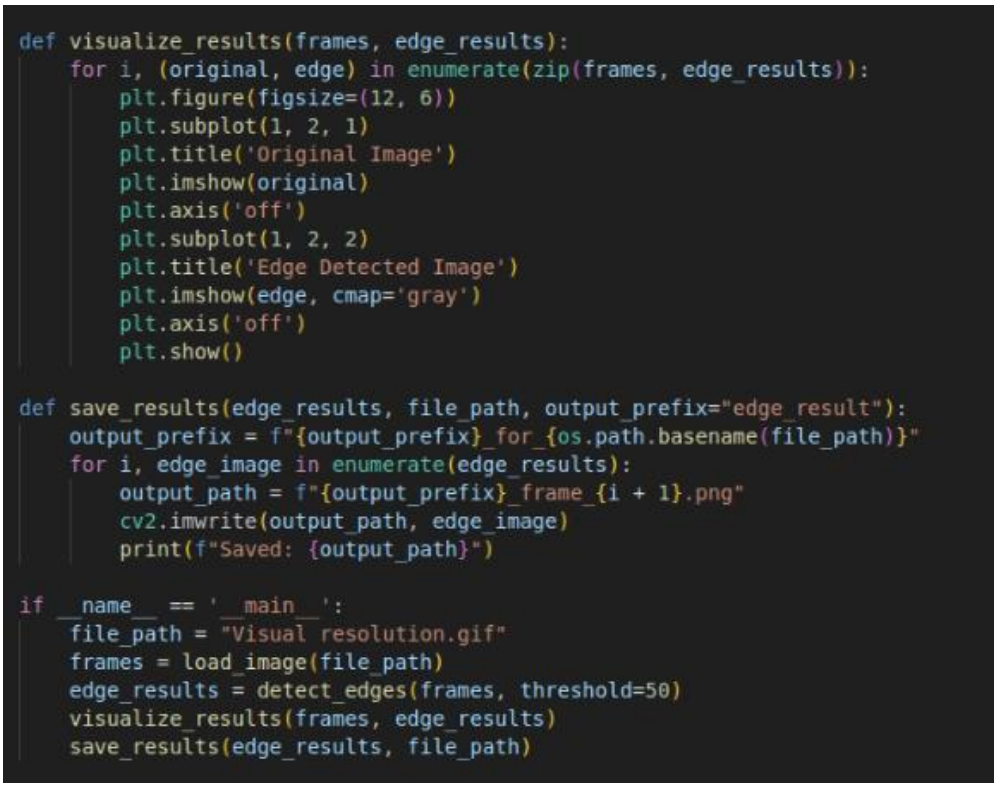


```python
#!/usr/bin/env python3
import os
from PIL import Image
import cv2
import numpy as np
import matplotlib.pyplot as plt

def load_image(file_path):
    frames = []
    gif = Image.open(file_path)
    while True:
        frame = np.array(gif.convert("RGB"))
        frames.append(frame)
        try:
            gif.seek(gif.tell() + 1)
        except EOFError:
            break
    return frames

def compute_gradient(image):
    sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
    gradient_magnitude = np.sqrt(sobel_x**2 + sobel_y**2)
    return cv2.normalize(gradient_magnitude, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

def detect_edges(frames, threshold=50):
    edge_results = []
    for frame in frames:
        gradient_magnitude = compute_gradient(frame)
        _, edge_image = cv2.threshold(gradient_magnitude, threshold, 255, cv2.THRESH_BINARY)
        edge_results.append(edge_image)
    return edge_results
```



```python
def visualize_results(frames, edge_results):
    for i, (original, edge) in enumerate(zip(frames, edge_results)):
        plt.figure(figsize=(12, 6))
        plt.subplot(1, 2, 1)
        plt.title('Original Image')
        plt.imshow(original)
        plt.axis('off')
        plt.subplot(1, 2, 2)
        plt.title('Edge Detected Image')
        plt.imshow(edge, cmap='gray')
        plt.axis('off')
        plt.show()

def save_results(edge_results, file_path, output_prefix="edge_result"):
    output_prefix = f"{output_prefix}_for_{os.path.basename(file_path)}"
    for i, edge_image in enumerate(edge_results):
        output_path = f"{output_prefix}_frame_{i + 1}.png"
        cv2.imwrite(output_path, edge_image)
        print(f"Saved: {output_path}")

if __name__ == '__main__':
    file_path = "Visual resolution.gif"
    frames = load_image(file_path)
    edge_results = detect_edges(frames, threshold=50)
    visualize_results(frames, edge_results)
    save_results(edge_results, file_path)
```

Method Description

This program uses a gradient computation in the RGB color space to detect edges for each frame of the GIF animation (Visual resolution.gif). It applies the Sobel operator to each frame to calculate gradients and uses the gradient magnitude to detect edges, ultimately producing a set of binarized edge images.

Steps

1. **Load the GIF Image**:
    - The program uses the PIL library's Image.open to load the GIF animation.
    - Each frame is converted to RGB format and stored as a NumPy array in a list.
2. **Compute Gradient Magnitude**:
    - The program applies OpenCV's cv2.Sobel to calculate the gradients in the x and y directions for each frame.
    - The gradients are combined into a gradient magnitude using the formula:

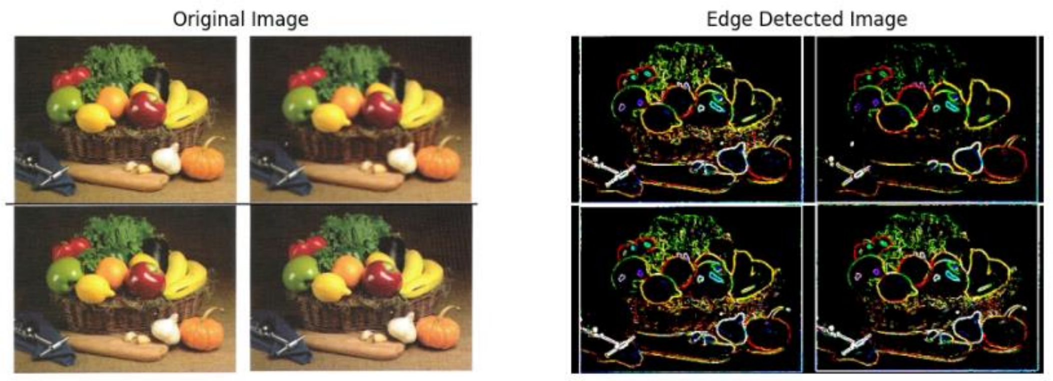
      $$G = \sqrt{G_x^2 + G_y^2}$$

    - The gradient magnitude is then normalized to the range [0, 255].
3. **Edge Detection**:
    - A threshold operation (default threshold: 50) is applied to the gradient magnitude for each frame.
    - Pixels above the threshold are marked as edges (255), while others are marked as background (0).
    - The edge-detected results are stored as binarized images.
4. **Visualize Results**:
    - Using Matplotlib, the program displays the original image and the corresponding edge-detected result side-by-side for each frame.
5. **Save Results**:
    - The edge-detected results for each frame are saved as individual PNG image files, with a prefix indicating the original file name and frame number.

Final Gradient Image

The final gradient image for each frame is a binarized representation of the gradient magnitude computed across the RGB color channels. White pixels represent detected edges, while black pixels indicate the background.

Result:



3. Please comment and compare your two designs? (20)

Comparison and Comments on the Two Designs

1. Input Format

- **First Program**:
    - Designed for a single static image (lenna-RGB.tif).
    - Handles and processes one RGB image at a time.
- **Second Program**:
    - Specifically built for animated GIFs (Visual resolution.gif).
    - Iterates over all frames in a GIF, treating each frame as an independent RGB image.

2. Gradient Computation

- **Both Programs**:
    - Use the Sobel operator to compute gradients in the x and y directions for each color channel in RGB.

    - Combine these gradients to calculate the gradient magnitude.
    - Normalize the gradient magnitude to a range of [0, 255].

The gradient computation method is identical in both programs, ensuring consistent edge detection quality regardless of input type.

3. Edge Detection

- **First Program**:
    - Performs thresholding on the gradient magnitude of the single input image.
    - Produces one edge-detected output.
- **Second Program**:
    - Applies thresholding to the gradient magnitude for each frame of the GIF.
    - Produces multiple edge-detected outputs, one for each frame.

4. Reusability and Versatility

- **First Program**:
    - Optimized for static RGB images. It is straightforward but limited to single-image tasks.
- **Second Program**:
    - Designed for dynamic content, making it suitable for a broader range of applications, including video or animation analysis.

Comment:
The second program is inherently more versatile due to its ability to handle multi-frame inputs. However, the first program is simpler and easier to adapt for single-image scenarios.

Summary:

The first program is specialized for static images, while the second program extends this logic to animated GIFs, offering greater flexibility.