HO CHI MINH UNIVERSITY OF SCIENCE

APPLIED DIGITAL IMAGE AND VIDEO PROCESSING

LAB01

March 22, 2024

Author

Nguyen Viet Kim - 21127333

Contents

1 Algorithm Description

1.1 Required Libraries

1.2 Binary Dilation

Given a binary image A and a structuring element B, the dilation operation $A \oplus B$ results in a new binary image where each foreground pixel (usually represented as 1) in A is replaced by 1 if there is at least one foreground pixel under the structuring element B.

In mathematical terms, dilation can be defined as the union of A with the translated kernel B, where the translation is performed over all possible positions in the image. This can be expressed as:

$$(A \oplus B)(x, y) = \bigcup_{(i,j)\in B} A(x-i, y-j)$$

In words, this equation states that the value of the pixel at position (x, y) in the dilated image is determined by taking the union (or maximum) of the pixel values in the input image A at positions $(x-i, y-j)$ for all points (i, j) in the structuring element B.

Alternatively, dilation can be defined as the set of all points (x, y) such that for at least one point (i, j) in the structuring element B, the pixel at $(x+i, y+j)$ in A is foreground:

$$A \oplus B = \{(x, y) \mid \text{for some } (i, j) \in B, A(x-i, y-j) = 1\}$$

The pseudo code for the custom dilation function:

```
function dilation(input_image, kernel):
    // Define output image
    output_image = create_empty_image_like(input_image)

    // Get kernel offset
    offset_x = kernel.width / 2
    offset_y = kernel.height / 2

    // Loop through each pixel
    for each white pixel (x, y) in input_image:
        for each kernel element (i, j):
            new_x = x + i - offset_x
            new_y = y + j - offset_y

            if kernel[i, j] != 0:
                output_image[new_x, new_y] = 255

    return output_image
```

Listing 1: Set up console

1.3 Binary Erosion

Given a binary image A and a structuring element B, the erosion operation $A \ominus B$ results in a new binary image where each foreground pixel (usually represented as 1) in A is replaced by 0 unless all the pixels under the structuring element B are also foreground pixels.

In mathematical terms, erosion can be defined as the intersection of A with the translated kernel B, where the translation is performed over all possible positions in the image. This can be expressed as:

$$(A \ominus B)(x, y) = \bigcap_{(i,j)\in B} A(x+i, y+j)$$

Alternatively, erosion can be defined as the set of all points (x, y) such that for all points (i, j) in the structuring element B, the pixel at $(x+i, y+j)$ in A is foreground:

$$A \ominus B = \{(x, y) \mid \text{for all } (i, j) \in B, A(x+i, y+j) = 1\}$$

In practical terms, erosion removes pixels from the boundaries of objects in the image, effectively reducing the size of the objects. It is useful for operations such as noise reduction, object separation, and image segmentation.

```
function erosion(input_image, kernel):
    // Define output image
    output_image = create_empty_image_like(input_image)

    // Get kernel offset
    offset_x = kernel.width / 2
    offset_y = kernel.height / 2

    // Loop through each pixel
    for each white pixel (x, y) in input_image:
        // Check if the center pixel is white (255)
        if input_image[y][x] == 255:
            // Check if all neighboring pixels covered by the kernel are white
            erosion_flag = True
            for each kernel element (i, j):
                new_x = x + i - offset_x
                new_y = y + j - offset_y

                if kernel[i, j] == 1 and input_image[new_y][new_x] != 255:
                    erosion_flag = False
                    break
            // If any neighboring pixel is not white, set the center pixel to black (0) in the output image
            if erosion_flag:
                output_image[y][x] = 255

    return output_image
```

Listing 2: Set up console

1.4 Binary Opening

Given a binary image A and a structuring element B, the opening operation $A \circ B$ results in a new binary image where each pixel is set to 1 if it remains unchanged under the erosion operation followed by the dilation operation.
Mathematically, opening can be defined as:

$$A \circ B = (A \ominus B) \oplus B$$

Alternatively, opening can be expressed as the set of all points (x, y) such that for every point (i, j) in the structuring element B, the pixel at $(x+i, y+j)$ in A is foreground:

$$A \circ B = \{(x, y) \mid \text{for all } (i, j) \in B, A(x+i, y+j) = 1\}$$

Binary opening is useful for removing small objects, smoothing object contours, and breaking narrow bridges or connections between objects.

```
function open(input_image, kernel):
    // Perform erosion using custom implementation
    eroded_img_custom = erosion(input_image, kernel)

    // Perform dilation using custom implementation on the eroded image
    opened_img_custom = dilation(eroded_img_custom, kernel)

    return opened_img_custom
```

Listing 3: Set up console

1.5 Binary Closing

Given a binary image A and a structuring element B, the closing operation $A \bullet B$ results in a new binary image where each pixel is set to 1 if it remains unchanged under the dilation operation followed by the erosion operation.
Mathematically, closing can be defined as:

$$A \bullet B = (A \oplus B) \ominus B$$

Alternatively, closing can be expressed as the set of all points (x, y) such that for at least one point (i, j) in the structuring element B, the pixel at $(x+i, y+j)$ in A is foreground:

$$A \bullet B = \{(x, y) \mid \text{for at least one } (i, j) \in B, A(x+i, y+j) = 1\}$$

Binary closing is useful for filling small holes in objects, joining narrow gaps or breaks in object contours, and smoothing object boundaries.

```
function close(input_image, kernel):
    // Perform dilation using custom implementation
    dilated_img_custom = dilation(input_image, kernel)

    // Perform erosion using custom implementation on the dilated image
    closed_img_custom = erosion(dilated_img_custom, kernel)

    return closed_img_custom
```

Listing 4: Set up console

2 Result

The original image:

Figure 1: Original Image

2.1 Binary Dilation

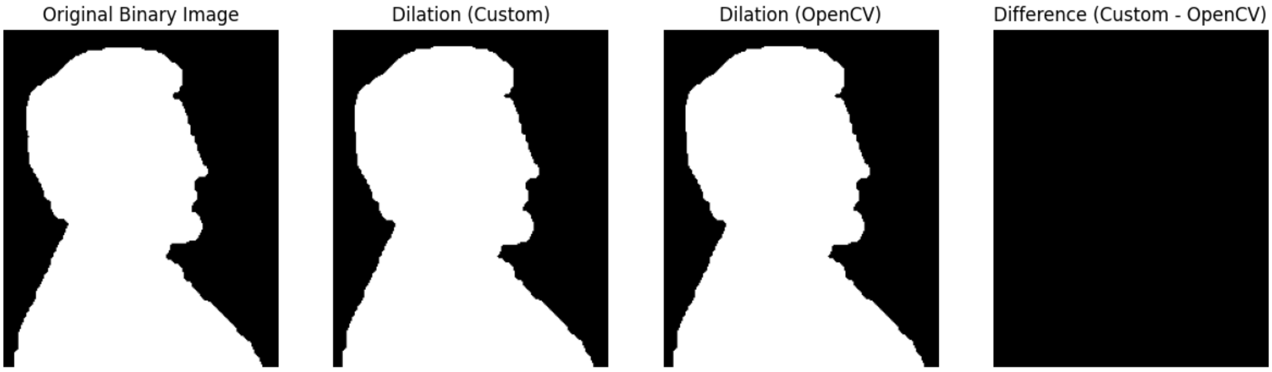

Figure 2: Binary Dilation comparing with openCV

2.2 Binary Erosion

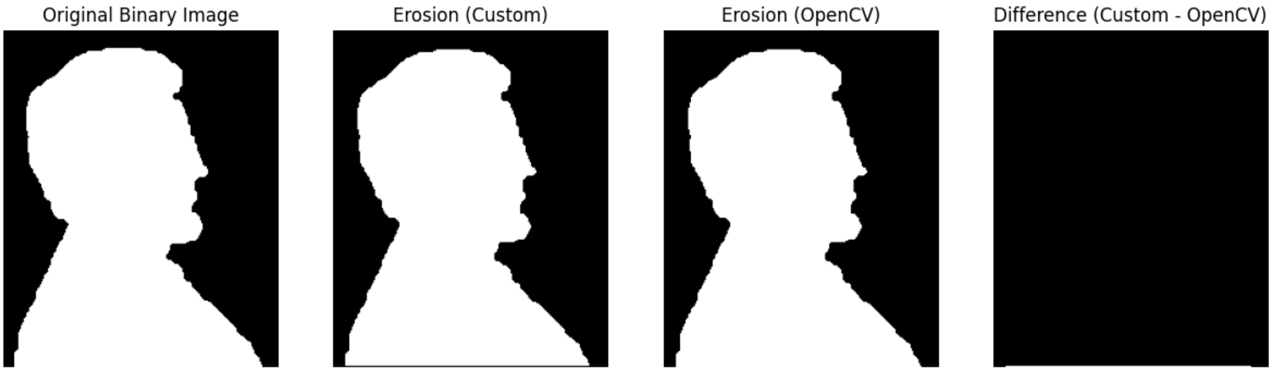

Figure 3: Binary Erosion comparing with openCV

2.3 Binary Opening

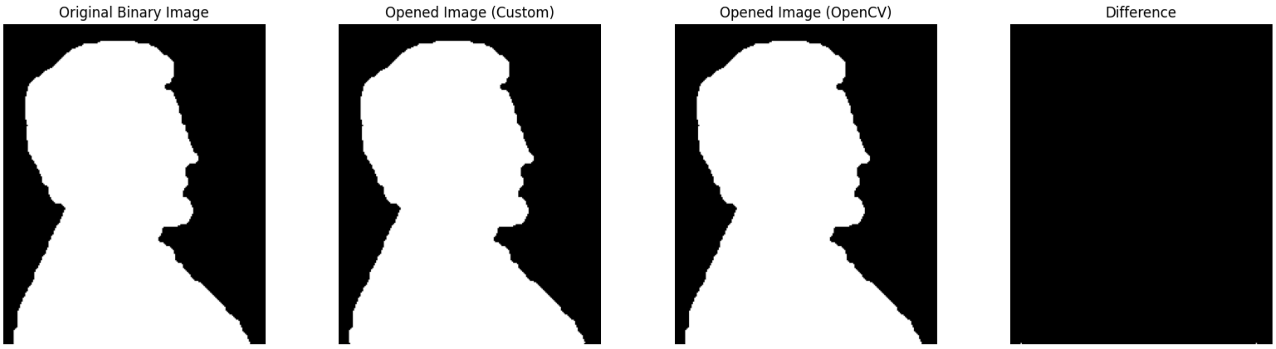

Figure 4: Binary Opening comparing with openCV

2.4 Closing

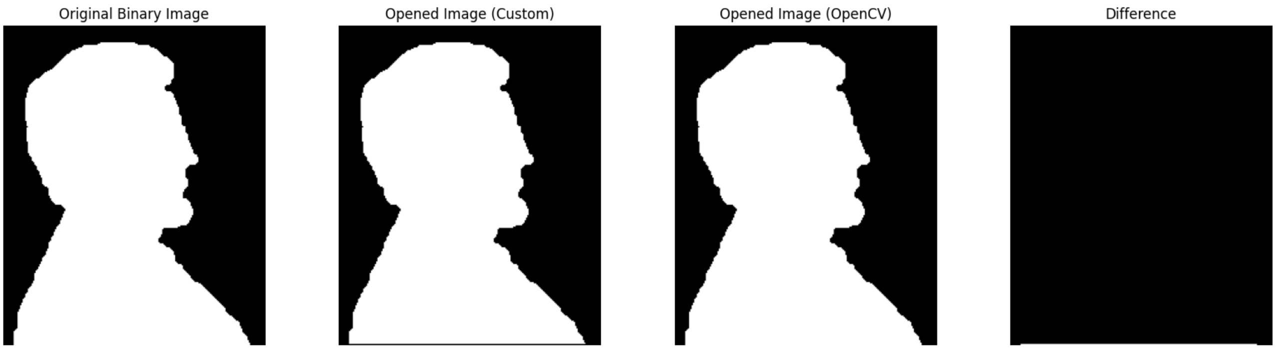

Figure 5: Binary Closing comparing with openCV