Project Report
Build A 'Portrait' Mode Image Enhancement Tool

Eric Dat Le
Bingcheng Chen
Matthieu Michel René Larnouhet
Yu-Ping Hsu

October 16, 2023

Chalmers University of Technology
Image processing - RRY025

1 Introduction

In a modern word of photography and image processing, 'Portrait' mode has become a familiar feature in our phones and cameras. This mode captures typically one or multiple human faces, it then blurs the background to create a striking depth effect, known as "bokeh", achieving this effect is more than just a click of a button; it's a complex realization of image enhancement techniques and algorithms.

This project aims to build a face detector using image processing tools such as edge detection filters, smoothing filters, and morphological operations. We explore two methods to achieve this goal. The first one uses morphological operations to find bounding box candidates for the face and then filters out unsuitable bounding boxes. The second combines Sobel filters with averaging filters to detect where the face is located.

2 Theoretical Background

2.1 Face/Object Detection using Edge Detection

2.1.1 Edge detection

The first step in creating a "portrait" mode tool involves identifying the primary subject (typically a face or head) within the image. To achieve this, one can use edge detection, which highlights regions of significant intensity variation in an image and can be used to detect the edges of objects in the photo. There are various edge detection techniques, but one could use Wavelets or edge detection operators.

The Sobel operator is chosen for its computational efficiency and noise reduction capabilities in this project. Gradient-based operators, like Sobel, employ convolution with horizontal and vertical masks, represented as $\mathbf{M}_x$ and $\mathbf{M}_y$, to compute gradients $\mathbf{G}_x$ and $\mathbf{G}_y$ from the source image $\mathbf{A}$. Mathematically the horizontal $\mathbf{G}_x$ and vertical $\mathbf{G}_y$ gradients can be described as

$$\mathbf{G}_x = \mathbf{M}_x * \mathbf{A},\ \mathbf{G}_y = \mathbf{M}_y * \mathbf{A} \tag{1}$$

where * is the convolution operator. In the case of the Sobels, the x and y-direction kernels are

$$\mathbf{M}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix},\ \mathbf{M}_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}. \tag{2}$$

Combining these gradients yields the magnitude of the gradient, $|\mathbf{G}| = \sqrt{\mathbf{G}_x + \mathbf{G}_y}$, which highlights edges.

However, Sobel operators can introduce noise in edge detection. A simple solution is to smooth the image by applying an averaging filter, such as $\mathbf{M}_{\text{avg}}$, before using the Sobel operator. The idea of mean filtering is to replace each pixel value in an image with its

neighbors' mean ('average') value, including itself. This has the effect of eliminating pixel values that are unrepresentative of their surroundings. This step enhances the accuracy of edge detection. An example of a weighted average filter is

$$\mathbf{M}_{\text{avg}} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}. \tag{3}$$

2.1.2 Morphology

Once edges are detected, further processing may be needed to identify and isolate the object of interest (face or head). Then, one could use morphological operations to connect the large proportions of edges to form a mask.

Morphological image processing is a broad set of operations of image processing techniques related to the shape of features. The operations rely only on the relative ordering of pixel values rather than their numerical values, which makes them suited for processing binary images.

The idea is to use a structural element and do convolution on an image. The structural element is a small matrix of pixels with binary values that can take any form of shape determined by the binary values' patterns and any size determined by the matrix's size. Its origin is usually one of its pixels, but a common practice is to have odd dimensions of the structuring matrix to have the origin defined as the center of the matrix.

It is advisable to introduce specific terminology to provide a clear explanation of fundamental morphological operations like erosion and dilation. As a demonstration, consider a squared structuring element traversed over an image; see Figure 1. There are three cases to consider when doing the convolution between the image and structuring element:

1. **Fit**: Occurs when all the pixels in the structuring element cover all the pixels in the object.
2. **Hit**: Occurs when at least one of the pixels in the structuring element covers the object's pixels.
3. **Miss**: Occurs when no pixel in the structuring element covers the object's pixels.

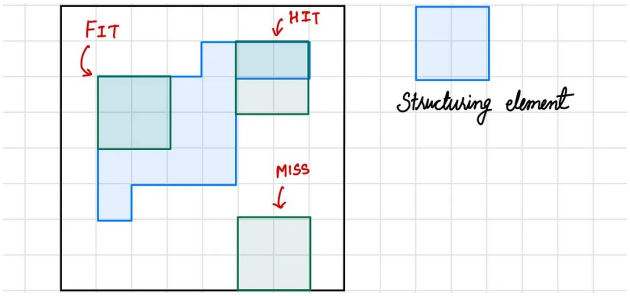


Figure 1: Illustration of terminology. Fetched from [5].

Now that the terminology is in place, the erosion operation can be explained as

$$\text{Pixel } (x, y) = \begin{cases} 1, & \text{if FIT} \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

where (x, y) are the pixel coordinates that align with the origin of the structure matrix. The erosion process increases the non-object of pixels and decreases the object pixels. Figure 2 illustrates the result of an erosion process with a square matrix as a structural element.

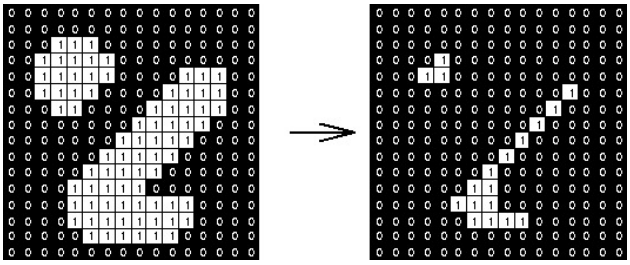


Figure 2: Erosion with a 2x2 square structural matrix. Image fetched from [7].

The dilation operation can be described as

$$\text{Pixel } (x, y) = \begin{cases} 1, & \text{if HIT} \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

where (x, y) are the pixel coordinates that align with the origin of the structure matrix. It can be seen as the reverse process of erosion where the dilation operation instead increases the number of pixels of the object and decreases the number of pixels for the non-object. Figure 3 illustrates the result of an erosion process with a square matrix as a structural element.

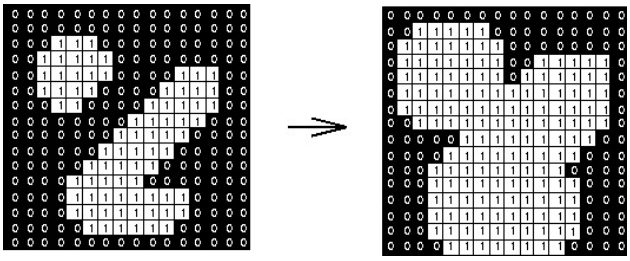
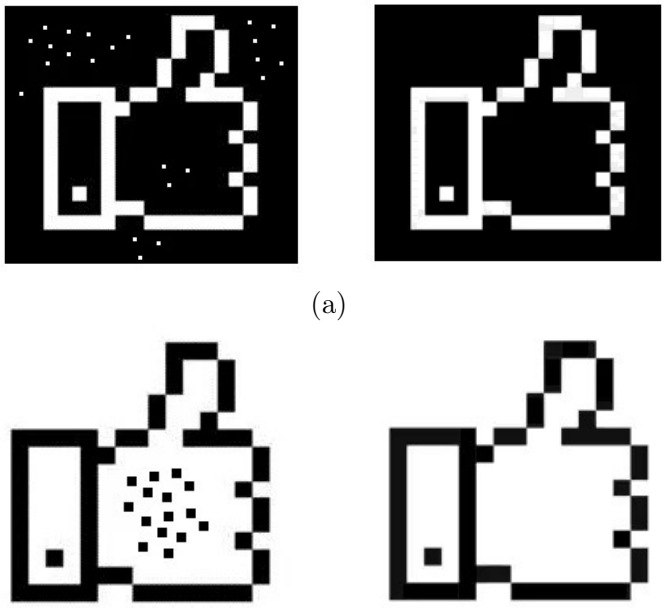


Figure 3: Dilation with a 2x2 square structural matrix. Image fetched from [7].

Another valuable morphological operation is opening. This technique is particularly effective for noise reduction in an image. It operates by performing erosion and then dilation, ensuring the preservation of the original object pixels while eliminating minor background noise. Essentially, opening eliminates small objects from an image while maintaining the shape and size of larger objects, see Figure 4 a). In contrast, the closing operation is employed to fill small holes in the foreground, converting tiny background areas into the foreground, see Figure 4 b).



(a)

(b)

Figure 4: a) Opening. Image fetched from [5]. b) Closing. Image fetched from [6].

2.2 Background Blur

A common technique to blur an image is to use what is called Gaussian blurring. The Gaussian blur feature is obtained by convolving an image with a Gaussian kernel. In 2D form, the Gaussian kernel is expressed as

$$G_{2D}(x, y, \sigma) = \frac{1}{2\pi\sigma}\exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \tag{6}$$

where x and y are the location indices and σ is the standard deviation of the distribution. The value of σ controls the variance around a mean value of the Gaussian distribution, which determines the extent of the blurring effect around a pixel. One way to view it is as an uneven low-pass filter that retains lower spatial frequencies while diminishing image noise and less significant details within the image [8]. Figure 5 shows an example of Gaussian blur applied to an image.

Figure 5: Gaussian blur applied on an image with standard deviation $\sigma = 0.5$.

2.3 Object sharpness

The purpose of image sharpening is to enhance the clarity of blurry images. There are several methods available to achieve this goal.

2.3.1 Differential method

One common approach is the differential method, which measures the rate of change in a signal. This method accentuates high-frequency components, thereby emphasizing edges and object contours. Applying filters like Sobel and Laplacian to images is a prevalent technique within the differential method. The Sobel operator is a discrete difference operator used to estimate the approximate absolute gradient magnitude at each pixel in the image. On the other hand, the Laplacian operator utilizes second-order differences to detect edges within the image.

2.3.2 Unsharp masking

Another technique is Unsharp Masking (USM). In this method, the image is initially smoothed using a blur filter. Then, the blurred image is subtracted from the original image to create a mask. Finally, this mask is added back to the original image. It is worth noting that simply using a high-pass filter to sharpen edges can amplify noise. USM enhances the visual quality of an image by emphasizing its high-frequency components, thereby improving overall image clarity.

3 Methodology

In this section, two image processing techniques are explored: Morphology and Sobel Filter Edge Detection.

3.1 Method 1: Morphology [1]

3.1.1 Step 1: Find the largest connected domain in the image and identify it as a face.

The input image is first converted to grayscale. In order to reduce the noise in the image, the mean filter is applied to the image. Convert a grayscale image into a black-and-white image with the help of a binarization function.

Figure 6: The result from (a) Applying mean filter and binarization processing (b) Extracting edges processing (c) Thickening and filling holes in edges (d) Detecting vertical edge and eliminate horizontal edges (e) Iterating small areas to prepare input for finding bounding boxes (f) Finding the mask for image processing.

Figure 6 (a) shows the result from performing the mean filter and binarize function on the image. Set one vector to perform erosion operation on the binarized image. Subtract the result of the erosion operation from the original image to get the boundary. Figure 6 (b) shows the extracted edges of the input image. Use the morph function to thicken the boundary. bwareaopen function is applied to eliminate the area with a pixel value equal to 0 within the boundary (see Figure 6 (c)). Because most outlines of the human face, neck, and body are vertical, perform vertical dilation to extract the human shape. Then, a horizontal erosion operation is performed to separate the human face from the body. The result of applying vertical dilation and horizontal erosion is shown in Figure 6 (e))

Theoretically, faces will not appear at the edge of the image, so the value 0 is given to the pixels at the edge of the image. In order to exclude irrelevant connected domains, the

ratio of black pixels to white pixels in each area of a given size is checked. If the ratio of black pixels to white pixels in the region is smaller and equal to 50%, the entire region's pixel values are set to 0. Figure 6 (e)) shows the result from the above processing. Use the bwlabel function to label all connected domains. Perform the regionprops function and the parameter BoundingBox is given to find the smallest bounding boxes covering the corresponding connected domains. Then, remove bounding boxes that are too wide or too narrow. Figure 7) shows the bounding boxes where the face is located. Select the bounding box containing the largest connected domain to create a mask.

Figure 7: All found bounding boxes

3.1.2 Step 2: Smoothen background.

Use the position, width, and height of the bounding box, which is found in step 1, to create an ellipse-shaped mask. Figure 6 (f) shows the found mask. Give value 0 to the pixel inside the ellipse mask and value 1 to the pixel outside the ellipse mask. Perform a Gaussian filter on the RGB channel of the image. Concatenate three matrices to a 3D matrix.

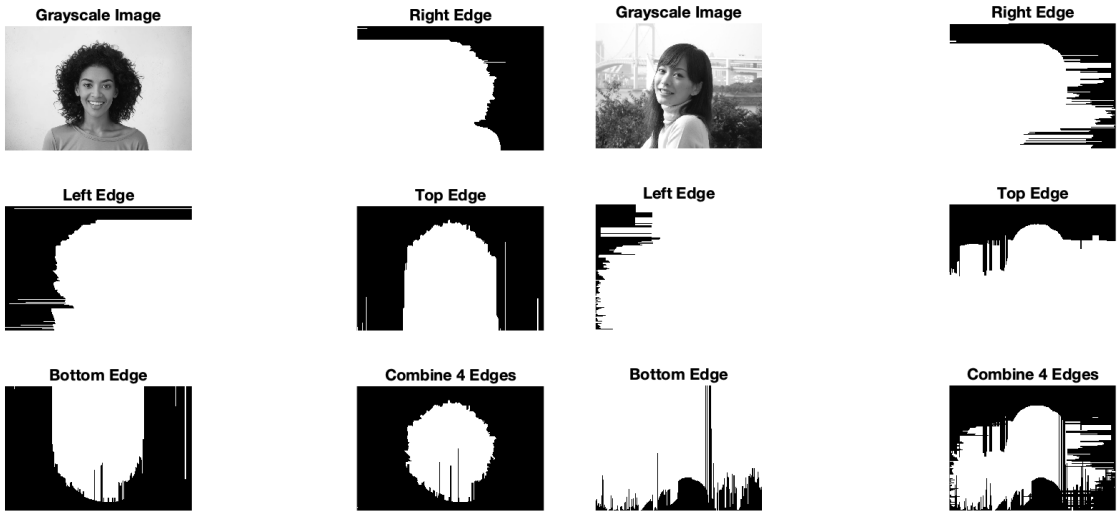
3.1.3 Step 3: Sharpen face.

First, blur the image using an average filter. Subtract the blurred image from the original image to create a mask. Finally, add the mask to the original image to get the target image. Subtract one from all values of the mask, which is created in step 2, and then take the absolute value of the mask to create a new mask. Apply the new mask on the target value to sharpen the face. Reconstruct the output image by adding the image with a sharpened face to the image with a blurred background.

3.2 Method 2: Sobel Filter Edge Detection

3.2.1 Step 1: Detect edges from 4 directions

This method first applies Sobel filters to detect edges within an image. The method involves the application of four Sobel kernels; these filters are designed to identify edges in four distinct directions: right, left, top, and bottom. The resulting edge images are then combined and refined using a threshold.



(a) A simple example: 4 edges detection using Sobel filter.

(b) A more complex example: 4 edges detection.

Figure 8: Edge detection.

We used two example images. The first image is a simple one with no complex background, providing an easy scenario for edge detection. The second example image is relatively more complex, containing challenging background elements.

Figure 8 a) and 8 b) illustrate the detection of edges in these four directions. As we can see, in the case of the first, more straightforward image, combining the four edge detection

images effectively captures a distinct outline of the human subject. However, there is some stripe noise that requires additional further processing. However, for the second image, the background cancellation is less effective, but the part of the human edges has been caught.

3.2.2 Step 2: Add a circle mask at the center of 1s pixels

After completing step 1, a pixel matrix composed of binary values is generated. Within this matrix, the 1s represent the potential regions of interest, although some noise may exist in it.

In step 2, we compute the centers of these 1s pixels. Additionally, we calculate the average distance between each 1s pixel and the computed center. The intuition behind this approach lies in the observation that the area of interest, such as a facial feature, tends to be centrally located in the 1s pixels. After achieving the location of the 1s' center, a circle boundary is then established. The radius of this circle is determined as the average distance multiplied by a parameter, denoted as α, α is set to 0.75 in our case.

By adding this circle layer, we aim to constrain the area for detecting the edges. In the middle of Figure 9 and 10, it is noticeable that the interested area is constrained into a smaller space.

3.2.3 Step 3: Add averaging filter

Observing the middle 'Add Circle Mask' figures of Figure 9 and 10, it is apparent that black stripes are present within the areas of interest. This is undesirable, and thus, in Step 3, we implement an averaging filter to eliminate these black stripes effectively.

This filter aims to smooth the regions of interest previously defined by the circle mask. By averaging the pixel values over a local kernel, the filter effectively mitigates the presence of black stripes.

The effect of this filtering process can be seen in the 3rd 'Add Average Filter' figures of Figure 9 and 10.

Figure 9: A simple example: Adding circle mask and averaging filter.

Figure 10: A more complex example: Adding circle mask and averaging filter.

3.2.4 Step 4: Smooth the background

After successfully detecting the edges in Step 3, we obtain the mask for the background by applying a complementary operation using (1-mask). In Step 4, our objective is to achieve a smooth background. To accomplish this, we disassemble the original image into its three color channels (red, green, and blue - rChannel, gChannel, and bChannel). Subsequently, we employ an averaging filter independently on each of these channels; then, these three matrices are concatenated to convert it back to the full-color image. Afterward, by multiplying this smoothed image with the background mask, we effectively create a blurred background.

The next step involves combining this blurred background with the edge-detected area, and we arrive at our final result. The final result is presented in the following section of our report.

4 Results

In this section, the results of our two image processing methods are presented respectively after explanation of the methodology in the previous sections.

4.1 Result of Method 1: Morphology

Figure 11, 12 and 13 show outputs from image processing using morphology algorithm.

Figure 11: The original image, mask, processed image for image 1.

Figure 12: The original image, mask, processed image for image 2.

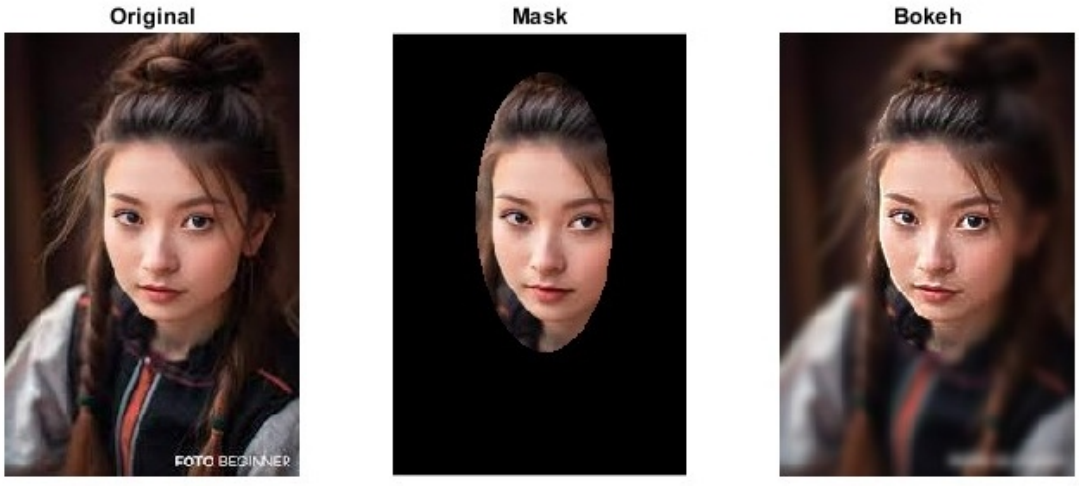


Figure 13: The original image, mask, processed image for image 3.

4.2 Result of Method 2: Sobel Filter Edge Detection

Figure 14 and 15 shows the final results of the Methods 2 in detecting the edges in two different scenario.

Figure 14: A simple example: final result.

Figure 15: A more complex example: final result.

5 Discussions

5.1 Method 1

The morphology algorithm can generally be used to find faces and generate masks for smoothing backgrounds and sharpening faces. However, in some cases, the found masks cannot cover the whole face, or the found masks are not in the correct sizes. Taking Figure 16 (f) as an example, the thickness of the horizontal structure of the building in the background is too big, so the algorithm cannot eliminate the effect from the horizontal edge. Figure 17 (e) shows that the connected domain of the face and horizontal structure of the building are connected. This makes the maximum bounding box offset.

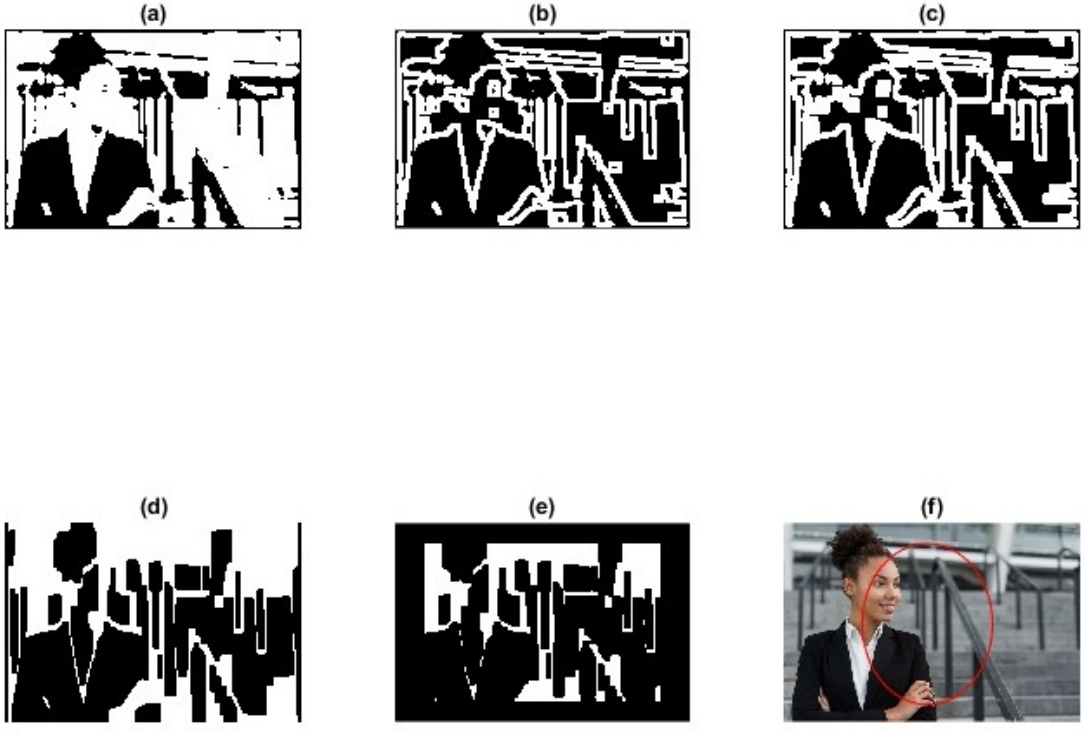


Figure 16: An obtained offset mask by using the morphology algorithm. (a) Apply mean filter and binarization processing (b) Extract edges processing (c) Thicken and fill holes in edges (d) Detect vertical edge and eliminate horizontal edges (e) Iterate small areas to prepare input for finding bounding boxes (f) Find the mask for image processing.

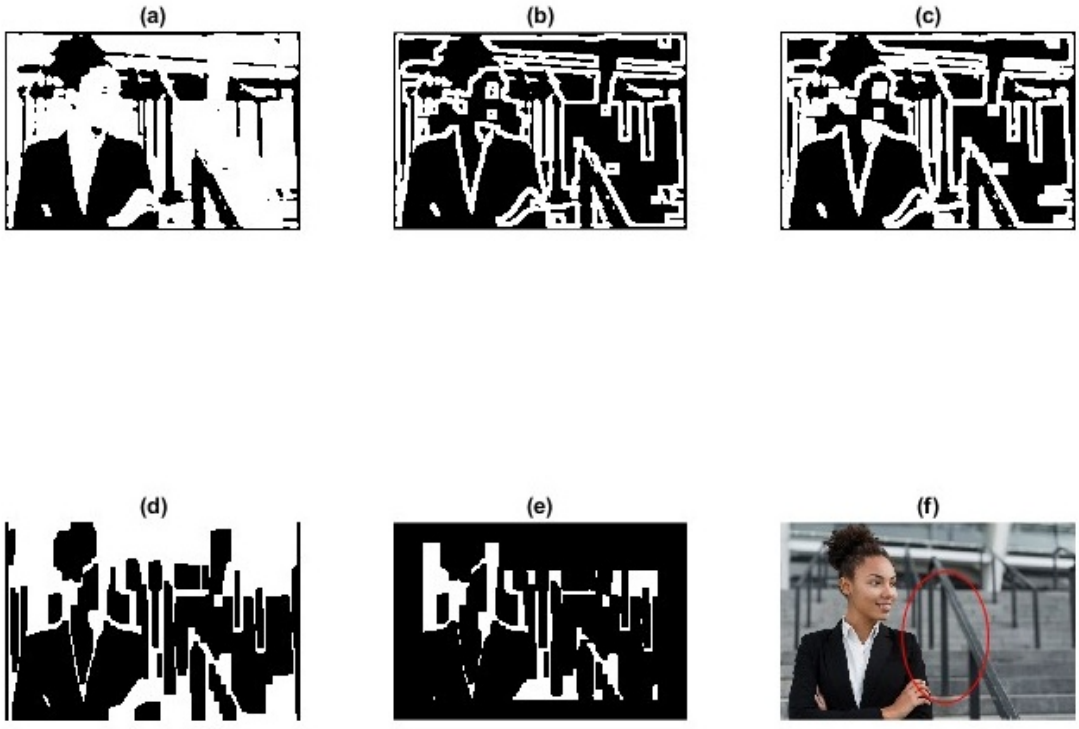


Figure 17: Mask shifts wrong direction after removing some connected domain of building.

Figure 17 (f) shows that the range of the mask is reduced and shifted after manually removing the connected domain of the horizontal structure of the building. The most extensive connected domain becomes the handrails of the stairs. This indicates that the morphology algorithm does not help eliminate beveled edges. Figure 18 (f) shows the size and location of the mask after removing the connected domain of the handrails of the stairs. The size and location of mask in 18 (f) is more appropriate than 16 (f) and 17 (f). The experiment shows that whether a suitable mask can be generated by finding the largest connected domain in an image depends on the composition of the image background. Using the morphology algorithm to generate a mask is difficult if the background is too complicated.

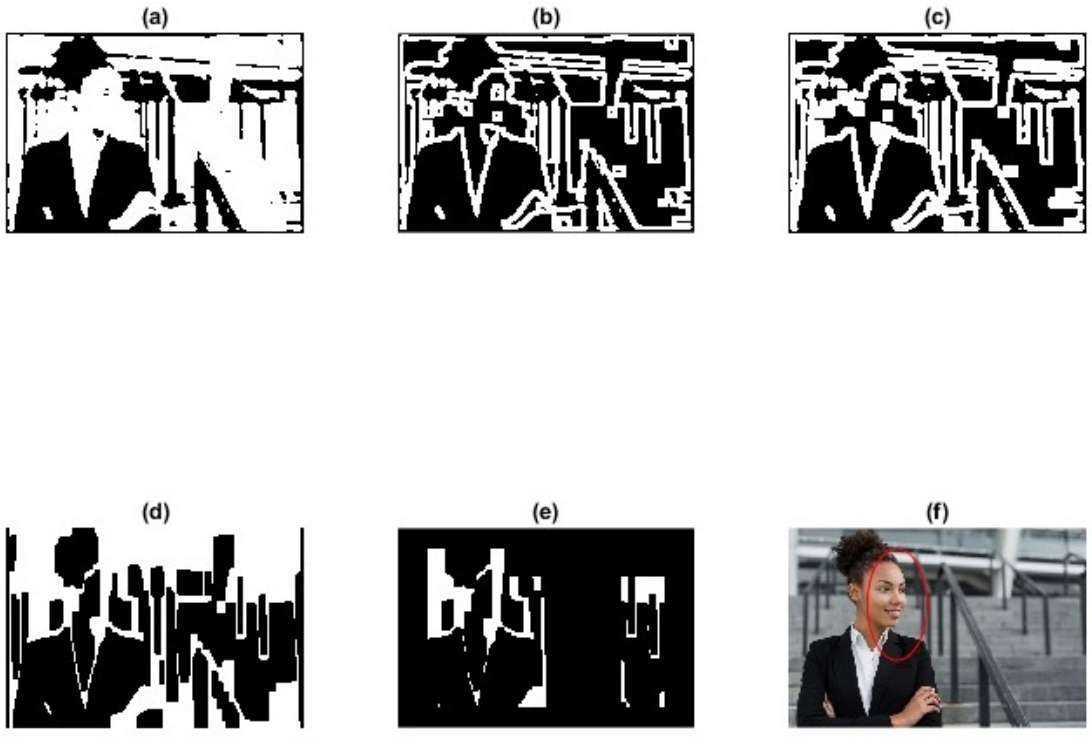


Figure 18: One more appropriate mask is generated after removing some connected domains of the building and handrails of stairs.

5.2 Method 2

Method 2 has limited capability in edge detection, as is shown in Figure 19. When faced with a more complex background or in situations where the background shares similar pixel characteristics with the region of interest, the Sobel filter edge detection may encounter difficulties, failing to detect edges. This then leads to inaccurate determination of the center of the face. In such scenarios, the undesired stripes can also be hard to cancel.

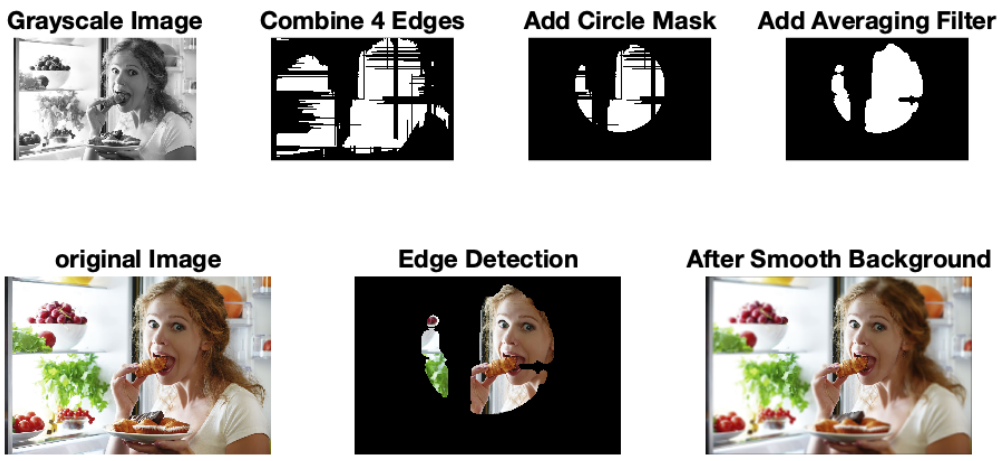


Figure 19: Method 2: Failure example.

5.2.1 Possible improvements

Depth of field refers to the distance at which an object or image can be displayed in the picture. The depth or shallow depth of field is usually determined by the aperture and focal length of the lens, as well as the subject's distance. Generally speaking, the larger the aperture, the shallower the depth of field; the longer the focal length, the shallower the depth of field; the closer the distance to the subject, the shallower the depth of field. In this project, the method of generating bokeh is to find the outline of the portrait using various methods and then apply the sharpening filter to the region of interest in the image (face) while applying the smoothing filter to the region of uninterest (background). This is an approximation method to create depth of field. Because the found mask is not always on the same plane, and objects on the same plane may not be included in the found mask. Figure 11 shows that part of the background is included in the sharpened range of the image. In addition, applying the same smoothing filter to the background does not meet the definition of depth of field. According to the definition of depth of field, objects closer to the subject are slightly more transparent than objects farther away. The improvement can be to use the roipoly function to select the region of interest within an image to generate a mask for masked filtering, which may be a better way to improve bokeh. Concerning improving the smoothing background, it may be possible to create several ring masks of different sizes and then apply different smoothing filters for each ring mask to obtain a smoothing effect that is gradually enhanced from the center point of the face toward the edge of the image.

6 Conclusion

In summary, our exploration into "Build A 'Portrait' Mode Image Enhancement Tool" has used two different methods: Morphology and Sobel Filter Edge Detection. While these methods are powerful, they both face challenges when dealing with more complex backgrounds, making a general portrait mode of image processing tricky.

Our investigation has shown that generating a perfect mask for detecting the edges could be a challenge, it calls for more innovative solutions to address this difficulties. Our exploration marks the beginning of this journey, and we look forward to study more advanced work from others.

References

[1] Gonzalez, R. C., Woods, R. E. (2022). Digital Image Processing, 4th edition. Pearson. ISBN-13: 9780137848560

[2] Image Edge Detection Operators in Digital Image Processing. GeeksforGeeks, 10 May 2020, https://reurl.cc/Myjl5X. Accessed 10 Oct. 2023.

[3] An Implementation of Sobel Edge Detection. Sean Sodha, 10 May 2020, https://reurl.cc/Myjlxn. Accessed 11 Oct. 2023.

[4] Olufunke Rebecca Vincent, and Olusegun Folorunso. "A Descriptive Algorithm for Sobel Image Edge Detection." Informing Science and IT Education Conference, 1 Jan. 2009, https://doi.org/10.28945/3351. Accessed 11 Oct. 2023.

[5] "Medium." Medium, https://reurl.cc/WvN2Z5. Accessed 11 Oct. 2023.

[6] "Medium." Medium, https://reurl.cc/edyN4R. Accessed 11 Oct. 2023.

[7] "Morphological Image Processing", https://reurl.cc/3e42lV. Accessed 12 Oct. 2023.

[8] Siddharth Misra, Yaokun Wu, Chapter 10 - Machine learning assisted segmentation of scanning electron microscopy images of organic-rich shales with feature extraction and feature ranking, Editor(s): Siddharth Misra, Hao Li, Jiabo He, Machine Learning for Subsurface Characterization, Gulf Professional Publishing, 2020, Pages 289-314, ISBN 9780128177365, https://doi.org/10.1016/B978-0-12-817736-5.00010-7.

7 Appendices

7.1 Method 1: Morphology

7.1.1 boken.m

```
clear all; clc;
rgb = imread('girl7.jpeg');
I = rgb2gray(rgb);
% Create mean filter and apply it on image
h = ones(9)/81;
I = uint8(conv2(I,h));
% Apply Binarization
BW = imbinarize(I);

% Extrat edges
B = ones(21);
BW = -imerode(BW,B) + BW;

% Thicken edges
BW = bwmorph(BW,'thicken');
% Fill holes in edges
BW = not(bwareaopen(not(BW), 300));

% Extract the vertical edge
B = strel('line',50, 90);
BW = imdilate(BW,B);
BW = imerode(BW,B);

% Eliminate the horizontal edge(10)
B = strel('line',10,0);
BW = imerode(BW,B);

[nr, nc] = size(I);

% Set borthers of image = 0
Border_size_h = round(nr*.1);
Border_size_v = round(nc*.1);
BW(1:Border_size_h, :) = 0;
BW(nr-Border_size_h: nr, :) = 0;
BW(:, 1:Border_size_v) = 0;
BW(:, nc-Border_size_v:nc) = 0;

```

```
% Divide image to many blocks, each blocks = p**2 pixels
p = 3;

% Label block
for r=Border_size_h+1:3:nr-Border_size_h-1
    for c = Border_size_v+1:3:nc-Border_size_v-1
        rate = mean(BW(r:r+p, c:c+p), 'all')*100;
        if rate <= 50
            BW(r:r+p, c:c+p) = 0;
        end
    end
end
figure
imshow(BW)
% Find the 8-connected objects
L = bwlabel(BW,8);
% Find the bounding boxes for 8-connected objectes
% Get [xLeft, yTop, width, height]
BB = regionprops(L,'BoundingBox');
% Converts cell arrays to ordinary arrays
BB = cell2mat(struct2cell(BB));
[s1,s2] = size(BB);
BB = reshape(BB,4,s1*s2/4)';
% Compute width height ratio for bounding box
ratio = BB(:,3)./BB(:,4);
shapeind = BB(0.3 <ratio & ratio < 3,:);
[~,arealind] = max(shapeind(:,3).*shapeind(:,4));
% Create an ellipse shaped mask
% Ellipse center point (y, x)
c = [shapeind(arealind,2)+shapeind(arealind,4)/2,
    shapeind(arealind,1)+shapeind(arealind,3)/2];
% Ellipse radii squared (y-axis, x-axis)
r_sq = [shapeind(arealind,3)/2, shapeind(arealind,4)/2] .^ 2;
[X, Y] = meshgrid(1:size(I, 2), 1:size(I, 1));
% Create mask for background
ellipse_mask = (r_sq(2) * (X - c(2)) .^ 2 + ...
                r_sq(1) * (Y - c(1)) .^ 2 >= prod(r_sq));
ellipse_mask = double(ellipse_mask(5:nr-4, 5:nc-4));
% Create smoothing mask
ellipse_mask_neg = double(abs(ellipse_mask-1));
%ellipse_mask_neg = imgaussfilt(ellipse_mask_neg, 3);
% Apply Gaussian filter on image and mask
```

```
[r, g, b] = imsplit(rgb);

sigma = 3;

smoothr = double(imgaussfilt(r, sigma)).*ellipse_mask;
smoothg = double(imgaussfilt(g, sigma)).*ellipse_mask;
smoothb = double(imgaussfilt(b, sigma)).*ellipse_mask;

% Sharpen the face
ag = fspecial('average', [5, 5]);
g = imfilter(rgb, ag);
g = imsubtract(rgb, g);
sharpenImg = uint8(double(imadd(rgb, g)).*ellipse_mask_neg);

% Plot the images
blurredImage = uint8(cat(3, smoothr, smoothg, smoothb)) +
   sharpenImg;
figure1=figure('Position', [0, 0, 800, 600]);
subplot(1,2,1)
imshow(rgb)
title('Original')
subplot(1,2,2)
imshow(blurredImage)
title('Bokeh')
```

7.2 Method 2: Sobel Filter Edge Detection

7.2.1 main.m

```
clear all;
% Read an image containing the object (head/face)
originalImage = imread('./img/portrait_2.png');
% originalImage = imread('./img/portrait_2.png');

% Convert the image to grayscale (if it's not already)
gray_img = rgb2gray(originalImage);

% Define 4 edge detection filter kernel
vertical_kernel_1   = [-1  0   1; -2  0  2; -1   0   1];
vertical_kernel_2   = [ 1  0  -1;  2  0 -2;  1   0  -1];
Horizontal_kernel_1 = [ 1  2   1;  0  0  0; -1  -2  -1];
Horizontal_kernel_2 = [-1 -2  -1;  0  0  0;  1   2   1];
```

```
14
15 % Perform convolution to detect vertical edges
16 edge_response_vertical_1   = filter2(vertical_kernel_1, double(
      gray_img));
17 edge_response_vertical_2   = filter2(vertical_kernel_2, double(
      gray_img));
18 edge_response_horizontal_1 = filter2(Horizontal_kernel_1,
      double(gray_img));
19 edge_response_horizontal_2 = filter2(Horizontal_kernel_2,
      double(gray_img));
20
21 % Apply a threshold to the edge response to extract edges
22 threshold = 100; % Adjust this threshold as needed
23 binary_edge_image_vertical_1 = edge_response_vertical_1 >
      threshold;
24 binary_edge_image_vertical_1 = convert_matrix(
      binary_edge_image_vertical_1, 1);
25
26 binary_edge_image_vertical_2 = edge_response_vertical_2 >
      threshold;
27 binary_edge_image_vertical_2 = convert_matrix(
      binary_edge_image_vertical_2, 2);
28
29 binary_edge_image_horizontal_1 = edge_response_horizontal_1 >
      threshold;
30 binary_edge_image_horizontal_1 = convert_matrix(
      binary_edge_image_horizontal_1, 3);
31
32 binary_edge_image_horizontal_2 = edge_response_horizontal_2 >
      threshold;
33 binary_edge_image_horizontal_2 = convert_matrix(
      binary_edge_image_horizontal_2, 4);
34
35 % Display the original image and the binary edge map
36 figure(1);
37 subplot(3, 2, 1);
38 imshow(gray_img);
39 title('Grayscale Image');
40
41 subplot(3, 2, 2);
42 imshow(binary_edge_image_vertical_1);
43 title('Right Edge');
```

```
44
45 subplot(3, 2, 3);
46 imshow(binary_edge_image_vertical_2);
47 title('Left Edge');
48
49 subplot(3, 2, 4);
50 imshow(binary_edge_image_horizontal_1);
51 title('Top Edge');
52
53 subplot(3, 2, 5);
54 imshow(binary_edge_image_horizontal_2);
55 title('Bottom Edge');
56
57 add_all_image1 = binary_edge_image_vertical_1 +
      binary_edge_image_vertical_2 +
      binary_edge_image_horizontal_1 +
      binary_edge_image_horizontal_2;
58
59 subplot(3, 2, 6);
60 imshow(add_all_image1>3);
61 title('Combine 4 Edges');
62
63 add_all_image2 = filter_mask(add_all_image1);
64 add_all_image2 = add_all_image1 + add_all_image2 +find_circle(
      add_all_image1);
65
66 add_all_image2 = add_all_image2 > 5;
67 add_all_image = filter_mask(add_all_image2);
68
69 figure(2);
70 subplot(1, 3, 1);
71 imshow(add_all_image1>3);
72 title('Combine 4 Edges');
73
74 subplot(1, 3, 2);
75 imshow(add_all_image2);
76 title('Add Circle Mask');
77
78 subplot(1, 3, 3);
79 imshow(add_all_image);
80 title('Add Averaging Filter');
81
```

```
figure(3);
subplot(1, 3, 1);
imshow(originalImage);
title('original Image');
% subplot(1, 3, 2);
% filtered_image = uint8(add_all_image) .* gray_img;
% imshow(filtered_image);

subplot(1, 3, 2);
mask = repmat(add_all_image, [1, 1, 3]);
three_dimension_filter_image = uint8(mask) .* originalImage;
imshow(three_dimension_filter_image);
title('Edge Detection');

% blur the background
blurred_background = background_smooth(originalImage, mask);
smooth_image = double(blurred_background) + double(
   three_dimension_filter_image) ./ 256;
subplot(1, 3, 3);
imshow(smooth_image)
title('After Smooth Background');
```

7.2.2 convert_matrix.m

```
function A = convert_matrix(A, type)
    switch type
        case 1
            % Iterate through each row and find the largest
               index of 1
            for row = 1:size(A, 1)
                indices = find(A(row, :) == 1); % Find indices
                   of 1 in the current row
                if ~isempty(indices)
                    A(row, 1:max(indices)) = 1;
                end
            end

        case 2
            % vertical_2
            for row = 1:size(A, 1)
                indices = find(A(row, :) == 1); % Find indices
                   of 1 in the current row
```

```
                if ~isempty(indices)
                    A(row, min(indices):end) = 1;
                end
            end

        case 3
            % horizontal_1
            for col = 1:size(A, 2)
                indices = find(A(:,col) == 1); % Find indices
                    of 1 in the current row
                if ~isempty(indices)
                    A(min(indices):end, col) = 1;
                end
            end


        otherwise
            % horizontal_2
            for col = 1:size(A, 2)
                indices = find(A(:,col) == 1); % Find indices
                    of 1 in the current row
                if ~isempty(indices)
                    A(1:max(indices), col) = 1;
                end
            end
    end
end
```

7.2.3 filter_mask.m

```
function filtered_mask = filter_mask(A)
    % define filter
    filter = fspecial("average", 30);
    filtered_mask = filter2(filter, A);
    filtered_mask = filtered_mask > 0.3;
end
```

7.2.4 background_smooth.m

```
function blurred_background = background_smooth(image, mask)
    % extracting rgb channels individually
    rChannel = image(:, :, 1);
```

```
    gChannel = image(:, :, 2);
    bChannel = image(:, :, 3);
    % define filter
    filter = fspecial("average", 15);
    % applying filter to the extracted matrices separately
    rChannelNew = filter2(filter, rChannel);
    gChannelNew = filter2(filter, gChannel);
    bChannelNew = filter2(filter, bChannel);

    % concatenate matrices along 3rd dimension
    outputImg = cat(3, rChannelNew, gChannelNew, bChannelNew);
    % convert the numeric matrix to image
    outputImg = mat2gray(outputImg);

    blurred_background = outputImg .* (1-mask);
end
```

7.2.5 find_circle.m

```
function new_mask = find_circle(add_all_image)
    % Find row and column indices of ones
    [rowIndices, colIndices] = find(add_all_image);

    % Calculate the center row and column
    centerRow = round(mean(rowIndices)*0.8);
    centerCol = round(mean(colIndices));

    fprintf('Center of 1s is at row %f and column %f\n',
       centerRow, centerCol);
    % Initialize a variable to store the sum of distances
    sumDistances = 0;

    % Calculate the distance of each 1 to the center and sum
       them up
    for i = 1:length(rowIndices)
        distance = sqrt((rowIndices(i) - centerRow)^2 + (
           colIndices(i) - centerCol)^2);
        sumDistances = sumDistances + distance;
    end

    % Calculate the average distance
    averageDistance = sumDistances / length(rowIndices);
```

```
21
22      fprintf('Average distance of 1s to the center is %f\n',
           averageDistance);
23
24      new_mask = zeros(size(add_all_image));
25
26       for j = 1:length(rowIndices)
27           distance2 = sqrt((rowIndices(j) - centerRow)^2 + (
                colIndices(j) - centerCol)^2);
28           if distance2 > averageDistance * 0.75
29               new_mask(rowIndices(j), colIndices(j)) = 0;
30           else
31               new_mask(rowIndices(j), colIndices(j)) = 1;
32           end
33       end
34  end
```