Contents

1 Part 1 - Requirement Analysis

Effective resizing of images should not only use geometric constraints but consider the image content as well. The purpose of this project is to implement an image operator called seam carving that supports content-aware image resizing for both reduction and expansion. In this project, we established an effective and user-friendly GUI to satisfy the requirements.

Effectiveness: Firstly, users can successfully shrink and expand the image based on the content-awareness. Secondly, users can protect or erase specific areas by drawing the areas using different colors.

User-friendly: Firstly, the result of the processed image will be shown clearly in the center of the GUI within a relatively short time. Secondly, the GUI supports the process containing multiple types of specific areas. Finally, users can continuously process the image, by controlling the increase and decrease buttons to quickly zoom in and out of the image in small increments.

2 Part 2 - Algorithm Analysis

2.1 Shrink the Image

2.1.1 Original Setting

The Seam-carving algorithm is used to shrink an image by removing seams in the image. Decreasing the width of the image refers to removing the horizontal seams in the image while decreasing the height of the image refers to removing the vertical seams in the image.

Figure 1: Seam Introduction

The red lines in the above image are a "vertical seam" and a "horizontal seam" as mentioned above. They are paths containing many pixels. Take the "vertical seam" as an example, along this path, each row in the image contains exactly one pixel (so that the image remains rectangle after this seam gets removed). If the coordinate of one of the pixels in this path is (x, y), then the possible coordinate of the pixel in the net line is $(x-1, y+1), (x, y+1), (x+1, y+1)$.

This algorithm remains the important areas in the image by carefully selecting the seams that go through unimportant areas of the image. The importance of a seam is defined by the sum of the pixels' energy along each seam. The smaller the energy, the less important the seam is. The equation of the energy of a pixel is defined as follows.

$$\sqrt{\Delta_x^2(x,y) + \Delta_y^2(x,y)} \tag{1}$$

Where $\Delta_x^2(x,y) = R_x(x,y)^2 + G_x(x,y)^2 + B_x(x,y)^2$. The variables R_x, G_x, B_x are the differences or "gradients" of red, green, and blue channels on the x-axix of position (x, y). $R_x(x, y)$ can be defined as

$r(x+1,y) - r(x-1,y)$, where $r(x,y)$ is the red value of the pixel on (x,y). Other variables can be defined similarly.

Moreover, in order to protect the border of the image, the energy pixels on the border of the image may be taken care of differently. In this project, we assign them as 1000 to protect them from being deleted.

To shrink an image by removing the least important seam of the image, the path of the seam with the minimal sum of the pixels' energy needed to be found. The detailed step is illustrated as follows.

1. Calculate the energy of all of the pixels in the image ($energy\_map$) using the equation (1).
2. Calculate the $accumulated\_energy\_map$ of the image. The detailed steps are illustrated in the following steps:
   - **The first row:** The value of the first row of the $accumulated\_energy\_map$ is the same as the first row of the $energy\_map$.
   - **Except the first row:**
     - **On the border:** The left border in row i(column index $== 0$): the accumulated energy is its energy plus the accumulated energy of the pixel whose index is $(i-1,1)$; The right border in row i(column index $== image\_width - 1$): the accumulated energy is its energy plus the accumulated energy of the pixel whose index is $(i-1, image\_width - 2)$.
     - **Inside the border:** The accumulated energy of the pixel whose index is (i,j) is its energy plus the minimal accumulated energy of the pixels whose index are $(i-1,j-1),(i-1,j),(i-1,j+1)$.
3. Verify the last index of the path of the minimal sum of pixels' energy by finding the minimal energy in the last row/column.
4. Retrace the path that contributes to the minimal sum of pixels' energy from the last index. Take "vertical seam" as an example, when retracing the next index of the path with its last pixel whose index is (i,j), find the minimal accumulated energy of the pixels in row $i-1$ with the column index $j-1, j, j+1$. The index of the minimal accumulated energy of the pixels is the next index of the path.

Shrinking the image to a specific size with the best performance means successively finding the optimal seam in both vertical and horizontal directions. By removing the optimal seam continuously, the image will be shrunk successfully while maintaining most of the important areas (content-based).

2.1.2 Suggested Protection Setting

Besides the original setting in terms of shrinking the image, the suggested protection setting is also being implemented. It means that the users can protect a specific area which is unlikely to be deleted by the seam carving algorithm.

The specific area is selected in two different interaction modes, the bonding box (select a regular area) and the scribble (select an irregular area).

According to the definition of the seam carving algorithm, to protect a specific area means making this area more important, which means assigning the pixels in this area a huge energy value. In this project, the energy of the pixels in this area is assigned as $Double.MAX\_VALUE - 10000$.

However, when changing the size of the image, the indices of the pixels in the protected area are likely not to match their original location in the image. Thus, this bug needs to be fixed when calculating the energy_map of the image.

2.1.3 Suggested Removal Setting

Besides the original setting in terms of shrinking the image, the suggested protection setting is also being implemented. It means that the users can delete a specific area more easily.

The specific area is selected in two different interaction modes, the bonding box (select a regular area) and the scribble (select an irregular area).

According to the definition of the seam carving algorithm, to protect a specific area means making this area more important, which means assigning the pixels in this area a small energy value. In this project, the energy of the pixels in this area is minus by 10000 from the original energy.

However, when changing the size of the image, the indices of the pixels in the protected area are likely not to match their original location in the image. Thus, this bug needs to be fixed when calculating the energy_map of the image.

2.2 Enlarge the Image

Different from the process of removing vertical and horizontal seams to shrink the image, to enlarge an image new "artificial" seams are inserted into the image.

Intuitively, to enlarge the size of an image by one we compute the optimal vertical (horizontal) seam and duplicate the pixels of the seam by averaging them with their left and right neighbors (top and bottom in the horizontal case). However, repeating this process will most likely create a stretching artifact by choosing the same seam, which breaks the balance between different parts in the image.

To achieve effective enlarging, it is important to balance between the original image content and the artificially inserted parts. Therefore, to enlarge an image by k, we find the first k seams for removal and duplicate them. Moreover, to continue the content-aware fashion, the sum of the seams used to duplicate needs to contribute to less than **30%** of the total energy. Thus, when the enlarging size is excessively large, the less important seams are duplicated multiple times.

In practice, to guarantee a content-aware fashion, the seams used to enlarge the image only accumulated up to **30%** of the total energy of the whole image. Assuming the number of seams we need to insert is N_1 and the number of the qualified seams is N_2.

On the one hand, when $N1 \leq N_2$, we choose the first N_1 minimal-energy seam in the group of qualified seams and duplicate them.

On the other hand, when $N_1 > N_2$, the qualified seams need to duplicate multiple times separately. The duplicate time for the qualified seams except the first minimal-energy seam is defined as

```
(int)(accumulated_last_col[i] / (total_energy * 0.1) * num_expand)
```

where i refers to the first i minimal-energy seam, $accumulated\_last\_col$ refers to the array of the $accumulatedenergy$, the $total\_energy$ refers to the total energy of the $accumulatedenergy$, 10% refers to our strategy of only used seams that accumulated up to 10% of the total energy of the whole image and the num_expand refers to the number of seams needed to be used to enlarge the image. Finally, the duplicate time for the first minimal-energy seam is defined as

```
num_expand - total_already_expand
```

where $total\_already\_expand$ is the sum of the duplicate time for the qualified seams except the first minimal-energy seam.

3 Part 3 - GUI Design

A GUI is designed to visualize the performance of resizing the image. It can resize the user's uploaded image based on the intended image size offered by the user. Moreover, when shrinking the image, two additional modes can be selected to delete or protect some specific areas chosen by the user.

The whole responding mechanism is illustrated in the following image.

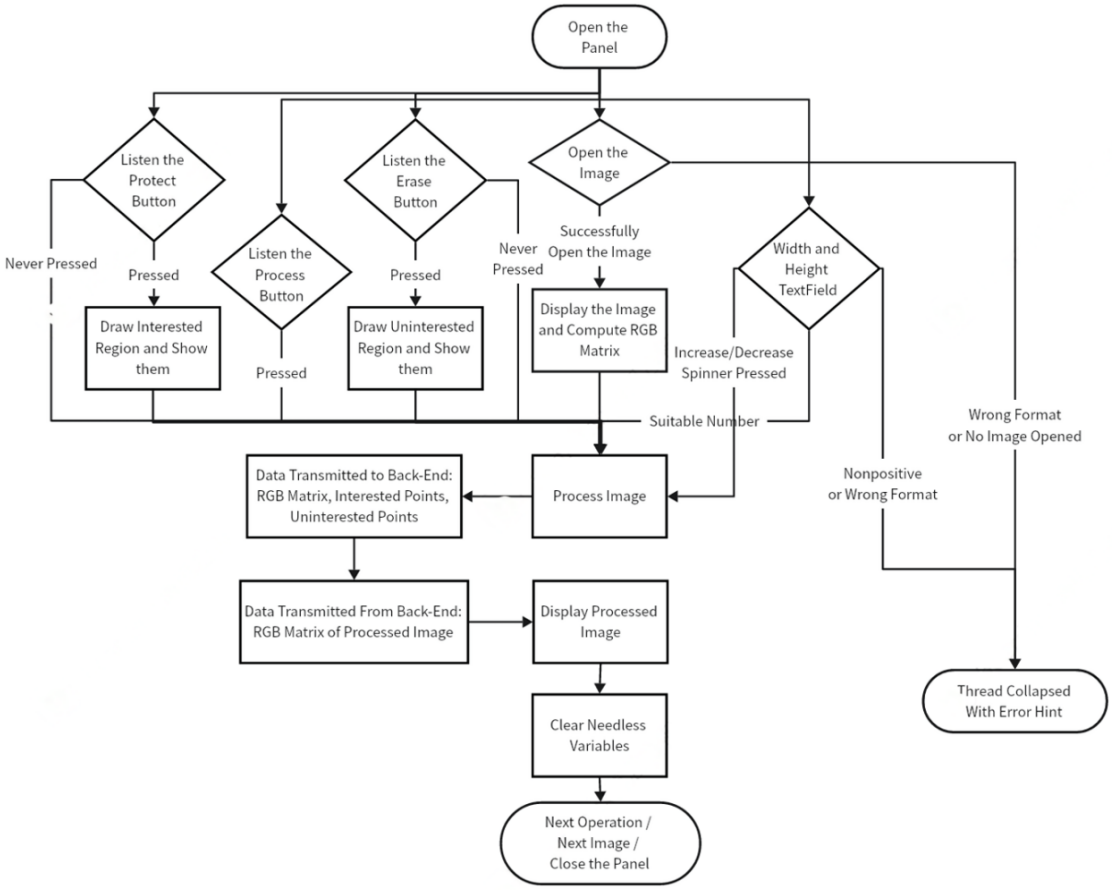

Figure 2: Responding Mechanism

3.1 Usage

The usage of the GUI is introduced in this section.

Open GUI: Run the file **Controller.py**

Open an Image: Click the **File** button on the upper left corner of the Before Open Image Interface(illustrated below) and choose an image from your device, then you can upload an image to the GUI.

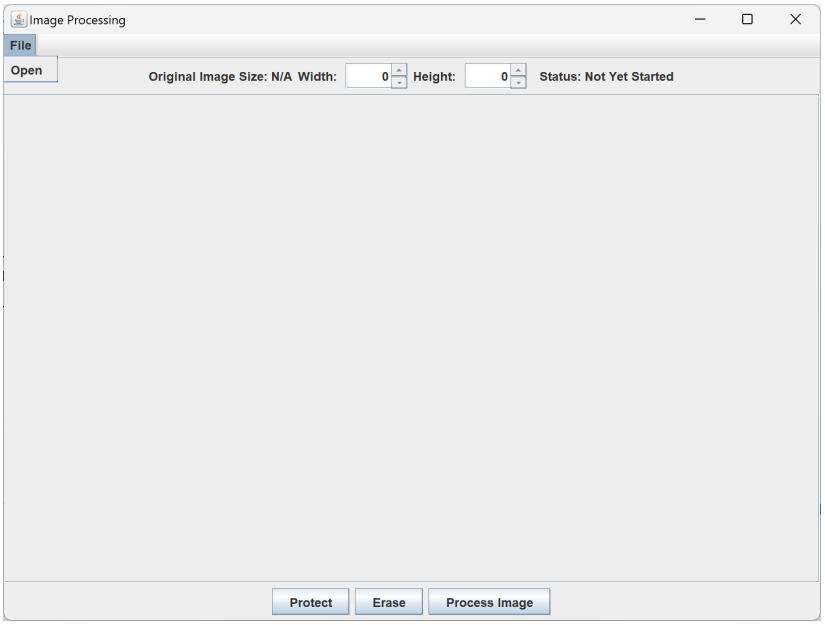

Figure 3: Before Open Image

Image Display: After uploading an image, it will be displayed in the center of the interface.

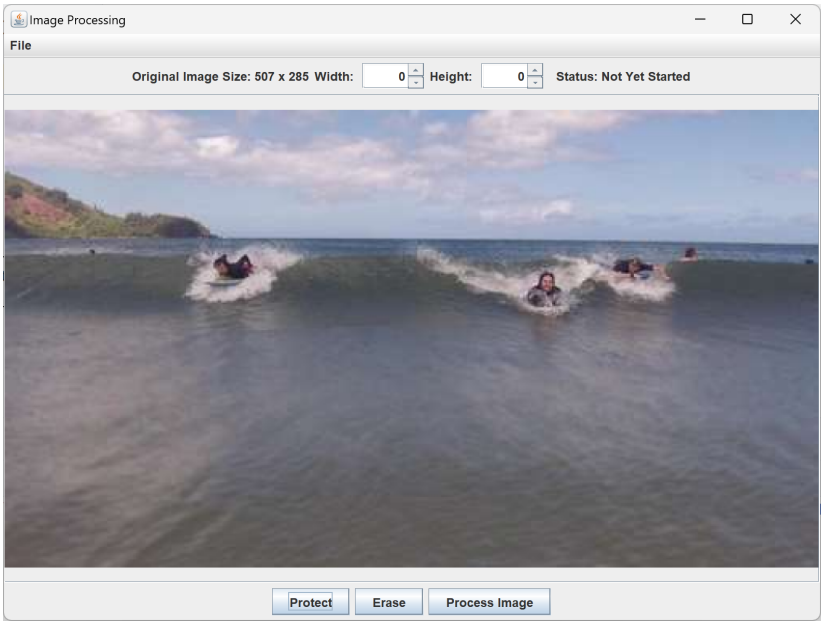

Figure 4: After Open Image

Resize the Image: Before you resize an image, the status displays **Not Yet Started.** There are two ways to resize the image. The first one is to enter the exact width and height of the image you expected and resize the image after you click the button **Process Image** at the bottom of the interface. The second one is to click the **Up and Down keys** next to the Width and Height window to resize the image in real time. (Each time it increases/decreases 5 pixels)

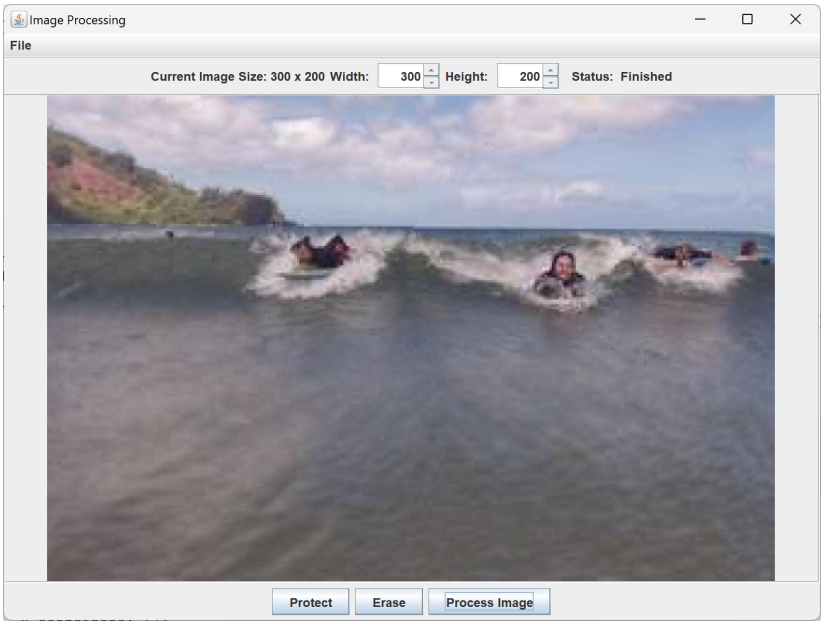

Figure 5: Processed Image

Processed Image Display: When the status displays **Finished**, the center of the interface displays the processed image. Otherwise, it displays the original image.

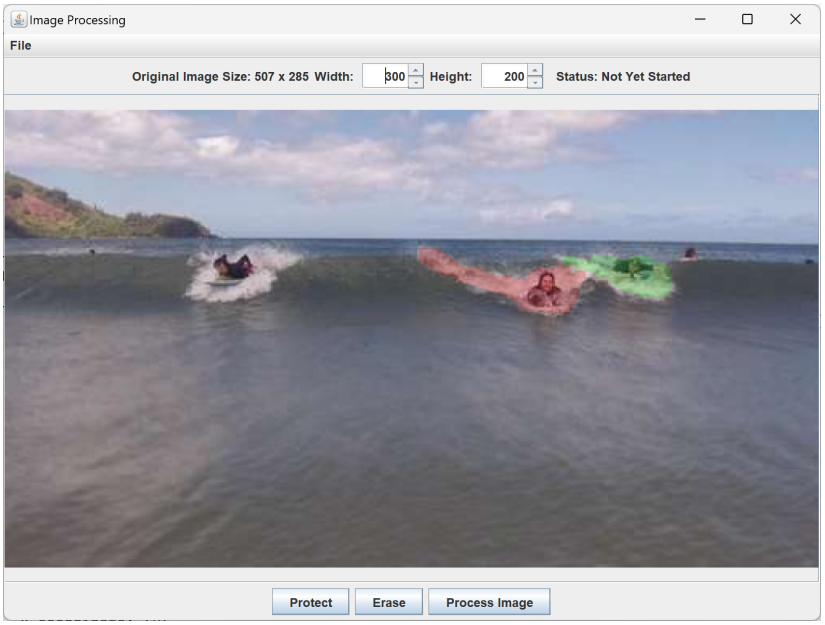

Figure 6: Choose Interested and Uninterested region

Protect specific areas: If you want to protect the specific areas in the image, click the **Protect** button at the bottom of the interface. Then you can use your mouse to draw some areas in the image. The drawn areas will be painted in **red**.

Delete specific areas: If you want to delete the specific areas in the image, click the **Erase** button at the bottom of the interface. Then you can use your mouse to draw some areas in the image. The drawn areas will be painted in **green**.

3.2 Panel

The panel is **BorderLayOut**. Thus, it's easy to put the image in the center of the panel.

At the **north** of the panel: We have a menu bar, with item in it to open the image. We have a textFieldPanel to indicate the original image size and current image size, indicate the status (Not Yet Started, Processing..., Finished), and control the expected width and expected height (with spinner to increase and decrease directly and process immediately).

At the **south** of the panel: There are three buttons, which will trigger protect, erase, and process, respectively.

At the **center** of the panel: The original image, and processed image are displayed here. At the same time, the accidents triggered by protect and erase are listening here, which can also display where to protect and where to erase.

3.3 Algorithm

Width and Height text field:

After pressing the increasing and decreasing spinner, directly process the image at that time. Thus, we can continuously process the signal by persistently pressing the button.

```
widthSpinner.addChangeListener(new ChangeListener() {
    @Override
    public void stateChanged(ChangeEvent e) {
        if (widthSpinnerHasFocus()) {
            buttonPressed = true;
            statusLabel.setText("  Status: Processing...");
        }
    }
});
```

Save the data:

The buffered image is difficult to compute, so we can transfer it into the RGB matrix, save the matrix, and transmit it to the back end.

```
for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
        Color color = new Color(image.getRGB(x, y));
        matrix[y][x][0] = color.getRed(); // Red channel
        matrix[y][x][1] = color.getGreen(); // Green channel
        matrix[y][x][2] = color.getBlue(); // Blue channel
    }
}
```

Display image:

For easier observation, we resize the image and display it at the center of the panel.

```
BufferedImage image = originalImage;
double widthRatio = (double) labelWidth / image.getWidth();
double heightRatio = (double) labelHeight / image.getHeight();
ratio = Math.min(widthRatio, heightRatio);
scaledWidth = (int) (imageWidth * ratio);
scaledHeight = (int) (imageHeight * ratio);
Image scaledImage = image.getScaledInstance(scaledWidth, scaledHeight, Image.SCALE_SMOOTH);
ImageIcon icon = new ImageIcon(scaledImage);
imageLabel.setIcon(icon);
imageLabel.setHorizontalAlignment(SwingConstants.CENTER);
imageLabel.setVerticalAlignment(SwingConstants.CENTER);
imageLabel.revalidate();
```

Select Interested Region & Select Uninterested Region:

Use an 8*8(can be adjusted at the head) rectangle to choose a region. For each point, label that point to be true, and adjust the color there.

```
int x = e.getX();
int y = e.getY();
int offsetX = (labelWidth - scaledWidth) / 2;
int offsetY = (labelHeight - scaledHeight) / 2;
for (int i = x; i < x + sizeOfRegion; i++) {
    for (int j = y; j < y + sizeOfRegion; j++) {
        int originalX = (int) ((i - offsetX) / ratio);
        int originalY = (int) ((j - offsetY) / ratio);
        if (originalX >= 0 && originalX < imageWidth && originalY >= 0 && originalY < imageHeight) {
            if (!interested[originalX][originalY]) {
                interestedPoints.add(originalX);
                interestedPoints.add(originalY);
            }
            interested[originalX][originalY] = true;
            overlayImage.setRGB(originalX, originalY, new Color(255, 0, 0, 50).getRGB());
        }
    }
}
BufferedImage combinedImage = new BufferedImage(imageWidth, imageHeight, BufferedImage.TYPE_INT_ARGB);
Graphics g = combinedImage.getGraphics();
g.drawImage(originalImage, 0, 0, null);
g.drawImage(overlayImage, 0, 0, null);
g.dispose();
Image scaledImage = combinedImage.getScaledInstance(scaledWidth, scaledHeight, Image.SCALE_SMOOTH);
ImageIcon icon = new ImageIcon(scaledImage);
imageLabel.setIcon(icon);
imageLabel.revalidate();
imageLabel.repaint();
```

3.4 Error Hint

If there are wrong operations, there exists pop-up windows to remind you of that.

Width and Height:

Figure 7: Wrong Width and Height

Open the wrong file:

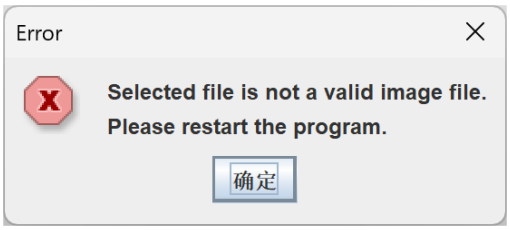

Figure 8: Wrong File Open

Process image without uploading an image:

Figure 9: Wrong Process Image

4 Part 4 - Result & Verification

In this section, we used the picture in the project document to verify our solution. The width of the picture is 507, and the height of it is 285. The original figure is shown in the figure below.

Figure 10: The original figure

4.1 Seam and Expand Vertically and Horizontally

It has been realized that the size of the picture can be set to any integer. As an example, we set the width to be 300, and the height to be 200 for seaming. And we set the width to be 600, and the height to be 320 for expansion. The results are shown in the following figures.

Figure 11: Resize the Picture into 300×200 for Seaming

Figure 12: Resize the Picture into 600×320 for Expanding

4.2 Seam with Protected Area

It has also realized to protect some of the areas in the picture to avoid being deleted during the seaming process. In the following example, the mountain in the left corner of the picture is protected, which is shown in Fig.13. As the same, the picture seems to be 300×200. From the result shown in Fig.14, it can be seen that the mountain isn't deleted at all.

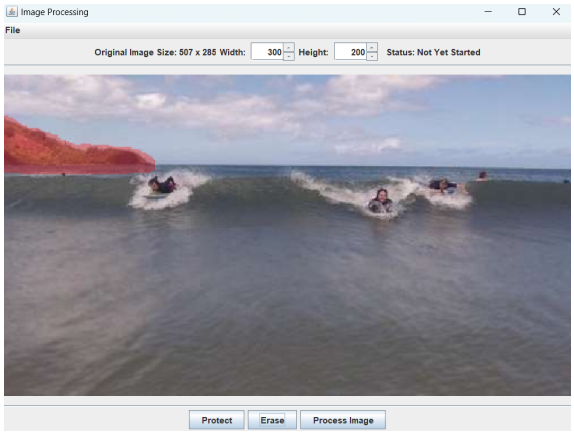

Figure 13: The Original Picture with Protected Area

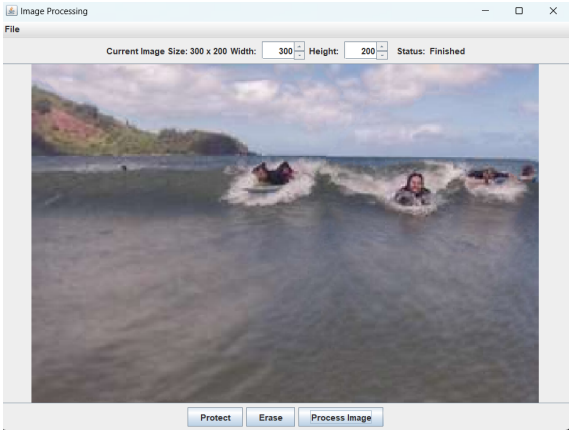

Figure 14: The Result Picture after Seaming with Protection

4.3 Seam with Erased Area

Besides the protection function, it can also erase particular areas that are difficult to be deleted, which means that they have high energy. In the following example, the middle person is erased, which is shown in Fig.15. As the same, the picture seems to be 300×200. From the result shown in Fig.16, it can be seen that the middle person is deleted.

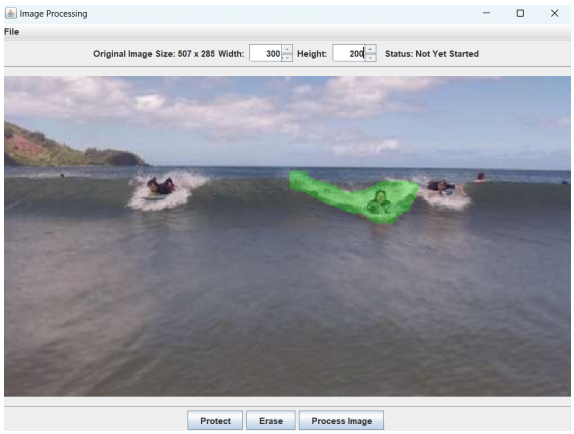

Figure 15: The Original Picture with Erased Area

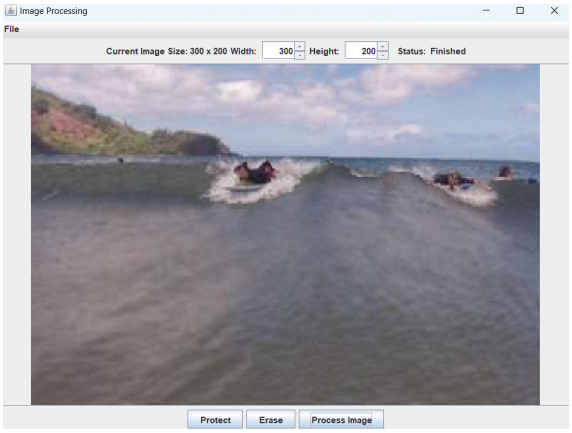

Figure 16: The Result Picture after Seaming with Erase

5 Part 5 - Difficulties & Solutions

5.1 Difficulties in designing the algorithm for enlarging the image

Enlarging the image while balancing the importance of the content and the smoothness of the image is a difficult task.

Intuitively, choosing the seam with minimal energy sum of pixels and inserting a seam by averaging the adjacent two seams is a way to enlarge the image. However, this results in always choosing the same seam, which significantly decreases the smoothness of the image.

Then, another method came to mind. If we need to enlarge the image by N pixels, then we choose the seams with the first N minimal energy sum of pixels and duplicate them. However, when we need to enlarge the image on a large scale, the content in the image will be distorted since the important parts in the image will be stretched.

Finally, a compromise was chosen. In order to protect the important part of the image, we assume the sum of the energy of the important part of the image taking 30% of the total energy. The seams we will choose to duplicate only come from the less important parts. In this way, we try our best to balance the content and the smoothness of the image.

5.2 Difficulties in precisely localizing the specific areas during resizing the image

Since shrinking an image will change the index of the remaining image, it brings some difficulties in precisely localizing the specific areas.

Thus, every time when we shrink the image, we need to update the indices of the pixels in the specific area in order to guarantee the correctness of the algorithm.

6 Part 6 - Conclusion

In this project, we have made an image scaling system using a seam-carving algorithm, which is able to scale the image according to the image size required by the user without affecting the important areas of the image, making the result of the process seem harmonious. Moreover, the program has high responsiveness and, a user-friendly interactive interface, and also supports the user to draw on the interactive interface to protect part of the image and suggest deleting part of the image.