Report on Poisson Image Editing

1. Introduction

Poisson Image Editing is a powerful method in editing the images locally. Through solving Poisson equations in the gradient domain, this method enables tasks like blending images smoothly or modifying image regions in a way that is consistent with the surrounding pixels. It has various applications, such as object removal, image stitching, and even complex manipulations like changing lighting conditions in a photo.

In this project is to two essential Poisson Image Editing tasks are implemented: Seamless Cloning and Selection Editing based on the guided interpolation mechanism of poisson image editing method.

2. Background and Mathematical Theory: Poisson Equation

Poisson equation is a fundamental equation based on scalar and vector field. Given a function f^* and a vector field $\vec{v}$ on 2-dimension. For the minimization problem as follows:

$$\min_f |\nabla f - \vec{v}|^2 \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega} \qquad (1)$$

in which Ω indicates an area in the 2-dimension domain and $\partial\Omega$ is its boundary. The solution f of the problem satisfies the following equation:

$$\Delta f = \text{div}\vec{v} \quad \text{over} \quad \Omega, \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega} \qquad (2)$$

The "div" in the above Poisson equation stands for "divergence," which is a vector calculus operator that measures the rate at which a vector field "spreads out" from a given point. In simpler terms, it quantifies how much a field behaves as a source or a sink at a particular point.

In 2-dimensional space, the divergence of a vector field $\vec{F} = (F_x, F_y)$ is defined as:

$$\text{div}\,\vec{F} = \nabla \cdot \vec{F} = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} \qquad (3)$$

3. Problem Statement

In this project, 2 essential Posson Image Editing tasks are implemented: Seamless Cloning and Selection Editing.

In Seamless Editing, a part of the source image will be cloned to the corresponding position of the target image to generate an output image. The part to be cloned is indicated by a mask image. Poisson equation is used for calculating the RGB values of the cloned part on the output image.

The Selection Editing task is divided to 4 subtasks: Texture Flatening, Local Illumination Change, Local Color Change and Tiling. Texture flattening aims to lessen the texture within the selected region, creating a smoother appearance while preserving the edges. Local illumination change focuses on altering the lighting conditions in the region. Local color change is about modifying the colors in the specific area. Tiling involves replicating a selected texture across an image, in this projet, it is replicated 2 * 2.

4. Methodology

Either the seamless cloning or selection editing is based on the guided interpolation problem as demonstrated in formula (1) and (2), in which the function f and f^* denotes the value on the pixels of the images for R or G or B channel. The only difference is the way guiding vector $\vec{v}$ generated. Therefore, in the implementation, an guided interpolation solver is built and the latter problem solvers generate their own guiding vector field and make use of guided interpolation solver to solve their own questions.

5. Implementation Details

The project is implemented with python, together with opencv and numpy library. Opencv is only used to read and write image files. See requirement.txt for the version. The code can successly run on windows 11 platform with python 3.7.9. You can use run

```
python poisson_tasks.py -h
```

To now how to run the code and input parameters. Some instructions is prepared in the run.sh file.

5.0 Mask Preprocessing

For the mask matrix read from the mask image, it is preprocessed for the solve to use, all pixels in the indicated region is numbered so that it can be indexed when solving their value out. Here is the psudo code:

```
Initialize mask_o = ZeroMatrix(height, width)
Initialize mask_numpt = 0

// Filter the mask and boundary region; integer > 0 denotes mask, and 0.5 denotes
For i from 0 to height-1:
    For j from 0 to width-1:
    If mask[i, j] == [255, 255, 255]:
        Increment mask_numpt by 1
        mask_o[i, j] = mask_numpt

        // Check and mark boundaries
        If i > 0 and mask_o[i-1, j] < 1:
        mask_o[i-1, j] = 0.5
        If i < height-1 and mask_o[i+1, j] < 1:
        mask_o[i+1, j] = 0.5
        If j > 0 and mask_o[i, j-1] < 1:
        mask_o[i, j-1] = 0.5
        If j < width-1 and mask_o[i, j+1] < 1:
        mask_o[i, j+1] = 0.5
```

The "height" and "width" above is the height and width of the mask image, also of the source and target images. Then the preprocessed mask and the number of pixels in the mask is transported to the guided interpolation solver with other argumants.

5.1 Guided Interpolation Solver

Formula (2) shows the question needed to be solved in guided interpolation problem. The discrete variant of formula (2) in 2-dimensional domain S is as followings:

$$\text{for all } p=(i,j)\in\Omega,\ \sum_{q\in N_p\cap\Omega} f_q - 4f_p = \sum_{q\in N_p} v_{pq} - \sum_{q\in N_p\cap\partial\Omega} f_q^* \quad (4)$$

in which

$$\partial\Omega = \{p \in S\backslash\Omega : N_p \cap \Omega \neq \emptyset\}$$

$$f|_\Omega = f_p, \quad p \in \Omega.$$

$$N_p = \{(i-1, j), (i+1, j), (i, j-1), (i, j+1)\}$$

for every p in Ω there is an equation (4) for it, therefore, we can solve a system of equations including `mask_numpt` equations to solve all the pixels in region Ω. We can use a matrix notation for this: $Mn = b$, n are the values for a color channel at the pixels in selected region, When we analyse equation (4), we find the coefficient of f_p is always -4 and if a pixel in the region is its neigbour, then it has a coefficient 1. Therefore we get the M and b, for ith row of M, the ith element is -4 and if a pixel in the region is the neigbour of the ith pixel in the mask (Remenber that all the pixels are indexed in mask preprocessing), given that it is the jth pixel, then the jth element on ith row of M is 1. The ith index of b is part right to the equal sign in equation (4). Here is the psudo code for the solve function:

```
Function solve():
    Initialize M = -4 * IdentityMatrix(mask_numpt)
    Initialize N = ZeroMatrix(mask_numpt, 1)
    Initialize n_checked = 0

    For i from 1 to height-2:
        For j from 1 to width-2:
            If mask[i, j] >= 1:
            Increment n_checked
            omega_idx = mask[i, j]
            s = div_v[i, j]

            Handle_boundary_and_neighboring_points()  // encapsulate the long con

            N[mask[i, j] - 1, 0] = s

    ans = SolveLinearSystem(M, N)

    Initialize output = Copy(target)

    For i from 1 to height-2:
        For j from 1 to width-2:
            If mask[i, j] >= 1:
            channel_color = ans[mask[i, j] - 1, 0]
            channel_color = ClipToRange(channel_color, 0, 255)
            output[i, j] = channel_color
    Return output
```

In which the "target" is the corresponding channel for the target image. div_v is the matrix of the divergent for the guiding field.

5.2 Seamless Cloning

For seamless cloning, the guiding vector field for the guided interpolation problem is the gradient of the source image. The gradient in direction along height and direction along width is calculated seperately. Here is the psudocode for the gradient calculation:

```
// gradient_h : (height - 1, width -1)
// gradient_W : (height, width -1)
Function grad(M, height, width, axis):
    if axis is in {"w", "width", "wid"}:
        Initialize grad_ret as zeros of shape (width, height-1)
        Compute grad_ret as (M[:, 1:] - M[:, :-1]) / 2
        Return grad_ret
    else if axis is in {"h", "height"}:
        Initialize grad_ret as zeros of shape (width, height-1)
        Compute grad_ret as (M[1:, :] - M[:-1, :]) / 2
        Return grad_ret
    else:
        Return an empty array
```

Note that the gradients are not for the points that are the pixels of the images, but for those points between every two neighbour pixels.

For mixed gradient, the gradient of the target image is also calculated with the same function above. The gradient takes the larger one between the gradient of the source and the gradient of the target for each pixel.

For each color channel, the gradient is calculated from the source image. The target image in corresponding channel, the calculated gradient, the preprocessed mask, the number of pixels in selected region, height and width are fed to the guided interpolation solver to solve out the output imgae in corresponding channel, then the channels are stacked and writed to the output image file.

5.3 Texture Flattening

In texture flattening, the guiding vector field v_{pq} is the gradient of the target (f^*), but with an edge detector:

```
if an edge lies between p and q:
    $v_{pq} = f^{*}_p - f^{*}_q$
else:
    v_pq = 0
```

In this project, a threshold is set for decide whether there is an edge liew between p and q. if $f_p - f_q > \text{threshold}$ then there is an edge between.

For each color channel, the gradient is calculated from the target image. The target image in corresponding channel, the calculated gradient, the preprocessed mask, the number of pixels in selected region, height and width are fed to the guided interpolation solver to solve out the output imgae in corresponding channel, then the channels are stacked and writed to the output image file.

5.4 Local Illumination Change

In this subtask, the average gradient of all gradients over the selected region is calculated, noted **g_average**. Then

$$v_{pq} = \alpha^{\beta} |f^{*}_p - f^{*}_q|^{-\beta} (f^{*}_p - f^{*}_q)$$

α and β are parameters for local illumination.

5.4 Local Color Change

Local color change is similar to a seamless cloning proble. However, there is no source image. The source image is generated from the target image by changing the RGB values in the selected region by multipling the corresponding parameters (R_mul, G_mul, B_mul), denoted by the mask image. Here is the psudocode how the source is generated:

```
Function generate_source(R_mul, G_mul, B_mul):
    source_ret = copy(target)

    # Create an RGB multiplier array
    RGB_mul = array([R_mul, G_mul, B_mul])

    # Loop through each pixel in the image
    for i from 0 to height-1:
        for j from 0 to width-1:
            # Check if the pixel is within the mask
            if mask[i, j] >= 1:
                # Modify the source image's RGB values using the multipliers
                source_ret[i, j] = target[i, j] * RGB_mul
    return source_ret
```

The following steps is same to seamless cloning, given the source image.

5.4 Tiling

In tiling task, the source is the input patch. The target is generated from the input patch by averaging the edge pixels. The mask is all the pixels except those on the edge of the imge. The generation function for the target is as follows:

```
Function generate_target():
    # Initialize the modified target image as a copy of the patch image
    target_ret = copy(patch)

    # Handle four corners of the target image
    target_ret[0,0] = (patch[0,0] * 2 + patch[0, width-1] + patch[height-1,0]) /
    target_ret[0,width-1] = (patch[0,width-1] * 2 + patch[0, 0] + patch[height-1,
    target_ret[height-1,0] = (patch[height-1,0] * 2 + patch[0, 0] + patch[height-
    target_ret[height-1,width-1] = (patch[height-1,width-1] * 2 + patch[height-1,

    # Handle the left and right edges
    for i from 1 to height-2:
        mean = (patch[i, 0] + patch[i, width-1]) / 2
        target_ret[i, 0] = mean
        target_ret[i, width-1] = mean

    # Handle the top and bottom edges
    for j from 1 to width-2:
        mean = (patch[0, j] + patch[height-1, j]) / 2
        target_ret[0, j] = mean
        target_ret[height-1, j] = mean

    return target_ret
```

Psudocode for generating the mask:

```
Function generate_mask():
    # Initialize the mask array filled with 0.5
    mask_ret = full(height, width, 0.5)
    # Initialize index variable
    ind = 0
    # Loop through non-boundary pixels
    for i from 1 to height-2:
        for j from 1 to width-2:
            ind += 1
            mask_ret[i, j] = ind
    return mask_ret, ind
```

The following steps is similar to seamless cloning. The vector field it the gradient of the source image. The output image is replicated and spliced together.

6. Experimental Results

The output images are stored at the same path root of their source, target and masks. You can see them in the img folder

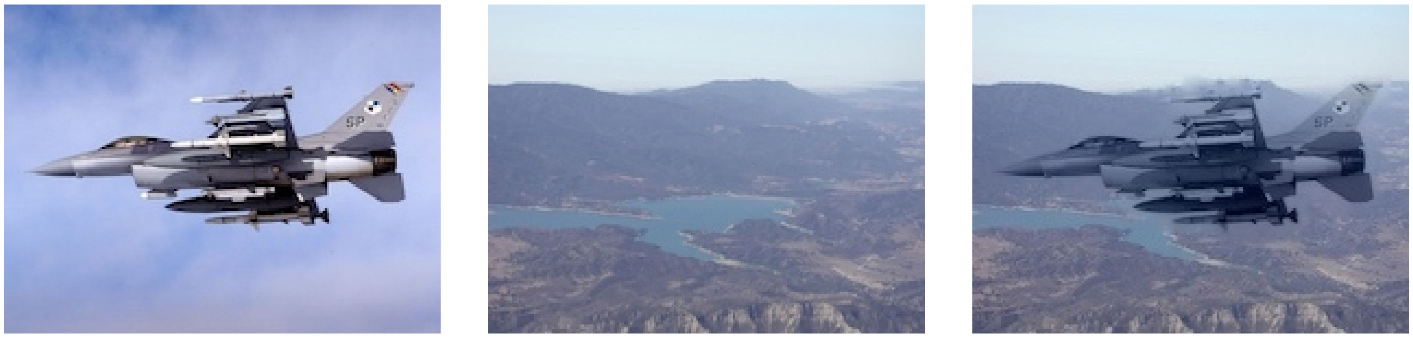

Figure 1: Example of Seamless Cloning

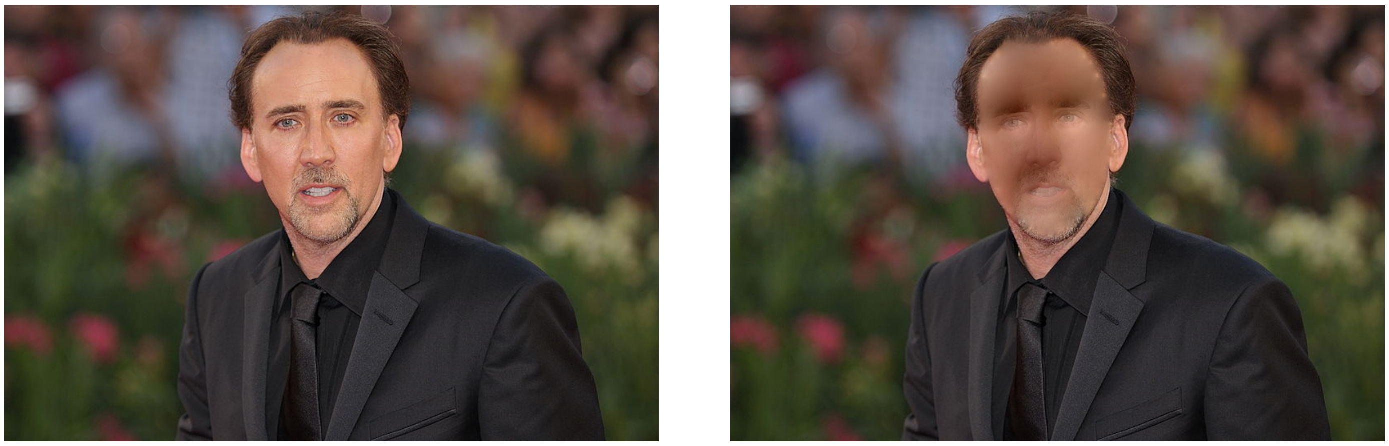

Figure 2: Example of Texture Flattening with threshold 1

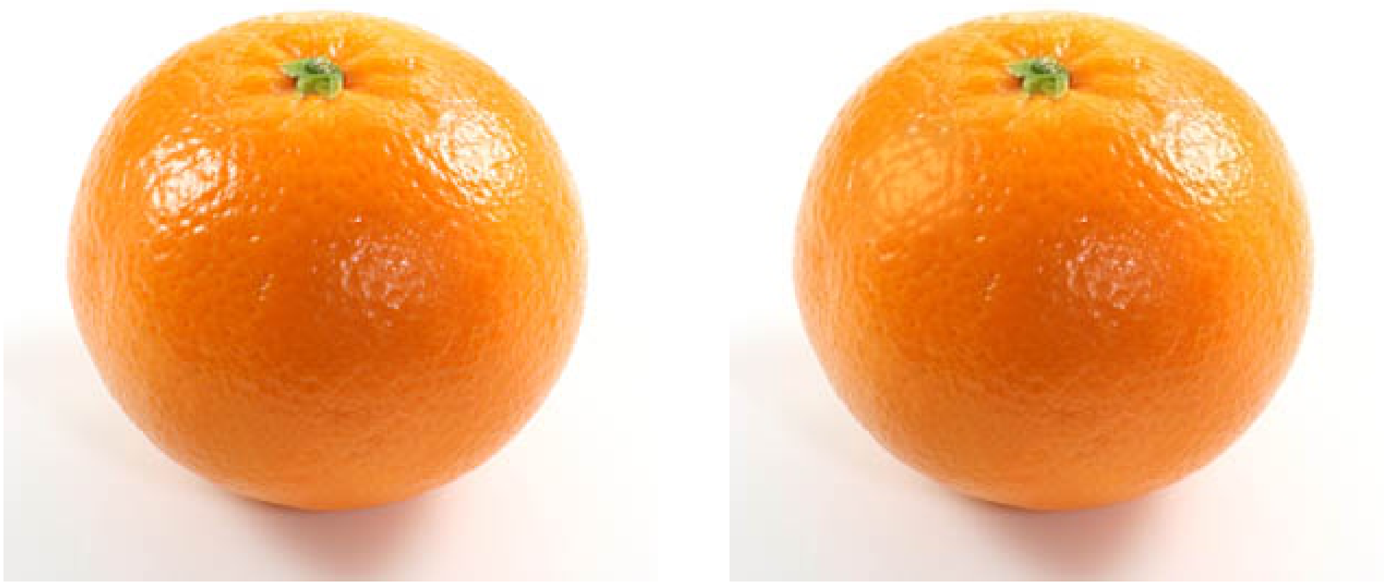
Figure 3: Example of Illumination Change with alpha and beta be 0.2

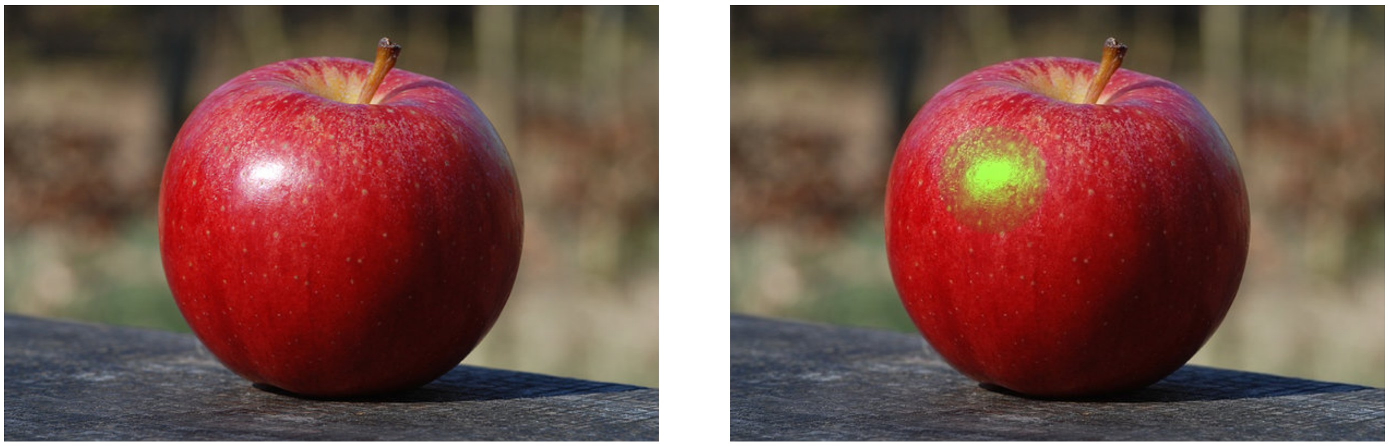
Figure 4: Example of Color Change with R_mul and B_mul be 0.5 and G_mul be 1.5

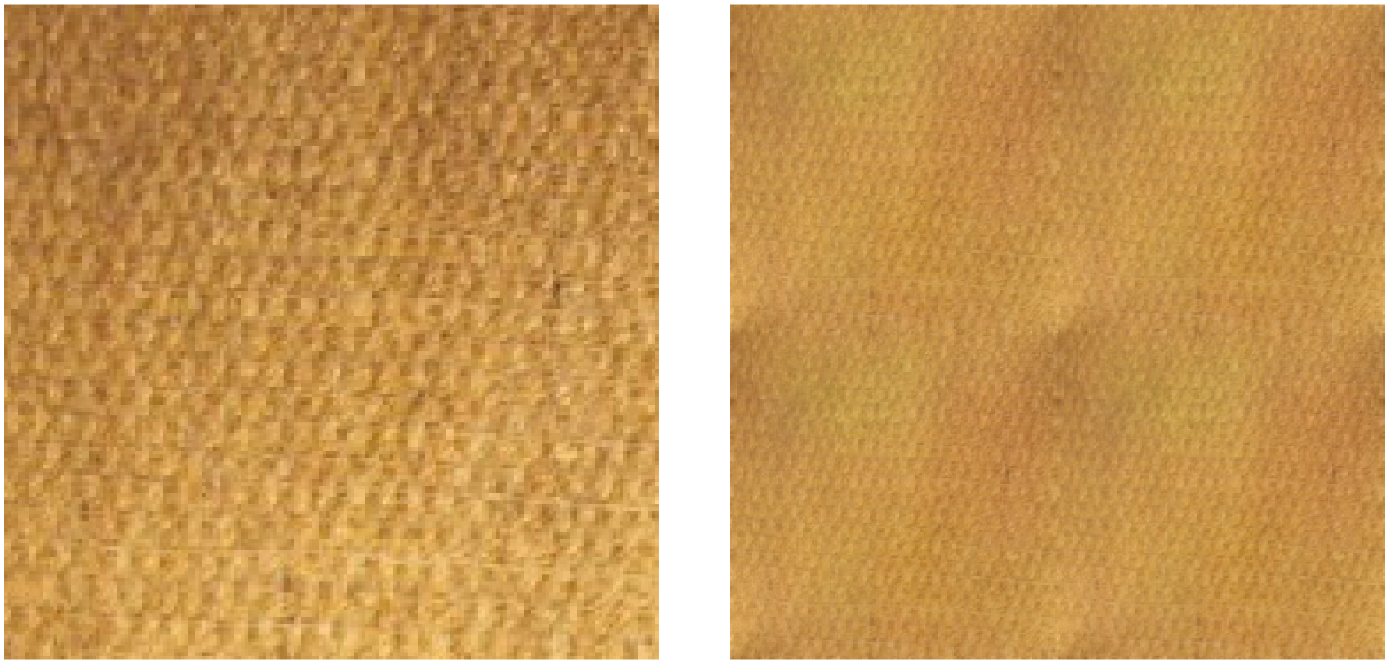
*Figure 4: Example of Tiling task, replicated (2, 2) *

8. References

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). A survey of collaborative filtering techniques. In Proceedings of the 10th International Conference on World Wide Web (WWW '01) (pp. 285-295). ACM.