# Project Report

Boyuan Yao 19307110202

October 9, 2021

## 1 Introduction

I gained inspiration from a paper "poisson image editing" by Pérez [1]. Solving a poisson equation, this method can simply insert part of sources image into the target one, which can cover the case that drastic differences between the souces and destination exist. Moreover, with appropriate modifying of the so called *guidance filed*, we could even retain the feature of the background of target image, that is to say we could insert some transparant sources into the destination. In the second sector, I'm going to show the basic idea of this method. In the third sector, I'll show several experiments using the FDM. We will see a fourier approximation approach to solve the poisson equation in the fourth section, which is included in the work of J. Matías Di Martino[2]. I will give two brief pseudocodes in the last section.

## 2 Basic idea

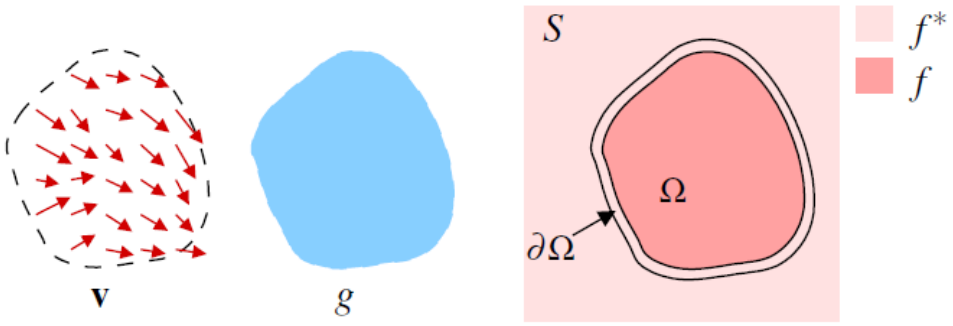



Figure 1: **Guided interpolation notations.** Unknown function $f$ interpolates in domain $\Omega$ the destination function $f^*$, under guidance field $\boldsymbol{v}$ [1]

The basic idea of the approach is to interpolate the destination function $f^*$ on a subdomain $\Omega$ with an unkown function $f$. The simplest interpolant $f$ of $f^*$ is called membrane interpolant, it is the

[1] P. Pérez, M. Gangnet, and A. Blake, *Poisson image editing*, ACM Transactions on Graphics, 22 (2003), p.314.

solution of the minimization problem:

$$\min_f \iint_\Omega |\nabla f|^2 \quad \text{s.t.} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega} \tag{1}$$

The solution of problem (1) is:

$$\Delta f|_\Omega = 0 \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega} \tag{2}$$

If a *guidance field* $\boldsymbol{v}$ is given, the problem (1) could be extended as

$$\min_f \iint_\Omega |\nabla f - \boldsymbol{v}|^2 \quad \text{s.t.} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega} \tag{3}$$

Then the solution is

$$\Delta f|_\Omega = \text{div}\boldsymbol{v} \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega} \tag{4}$$

Where $\text{div}\boldsymbol{v} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$ is the divergence of $\boldsymbol{v} = (u, v)$. Therefore, the interpolation problem could be converted into a poisson function with Dirichlet boundary condition.
Then back to the problem we focus, in the poisson image editing, the *guidance field* is given by calculating the gradient of the part of sources image that is to be inserted into the target one together with the destination domain, and the boundary condition could be attained from the target image. I implement two types of Seamless cloning in my project, with importing gradients and mixing gradients.

### 2.1 Importing gradients

Importing gradients is a basic choice for the *guidance field*, which has the following form

$$\boldsymbol{v} = \nabla g \tag{5}$$

Where $g$ denotes the source image, then problem (4) could be presented by this

$$\Delta f|_\Omega = \Delta g \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega} \tag{6}$$

With the above equations, we are able to insert the source image into the target image seaminglessly. But notice that in the domain $\Omega$, the information of gradient of the background image has been completely replaced by the source image, we can not hold the properties of target image inside the domain. The shortcoming of the improting gradients implementation will display when you want to insert a transparent item or you want to hold the texture of the background image to some degree. To solve this, we have the following implementation called mixing gradients.

### 2.2 Mixing gradients

In this part we simply reformulate the *guidance field* like this

$$\boldsymbol{v} = \begin{cases} \nabla f^* & ||\nabla f^*|| > ||\nabla g|| \\ \nabla g & \text{Otherwise} \end{cases} \tag{7}$$

With the above reformulation, we are able to insert some transparent items into the destination image or stop the object in the target image bleeds into the domain when they are close enough.

## 3 Finite Difference Method

To solve the given poisson equation, I first implement a solver using finite difference method to solve the discretization poisson equation. With Gaussian Seidel iteration, it could solve some of the problem with small cases, but works slowly when the number of pixels growth to certain degree. The followings are some results of my experiments.

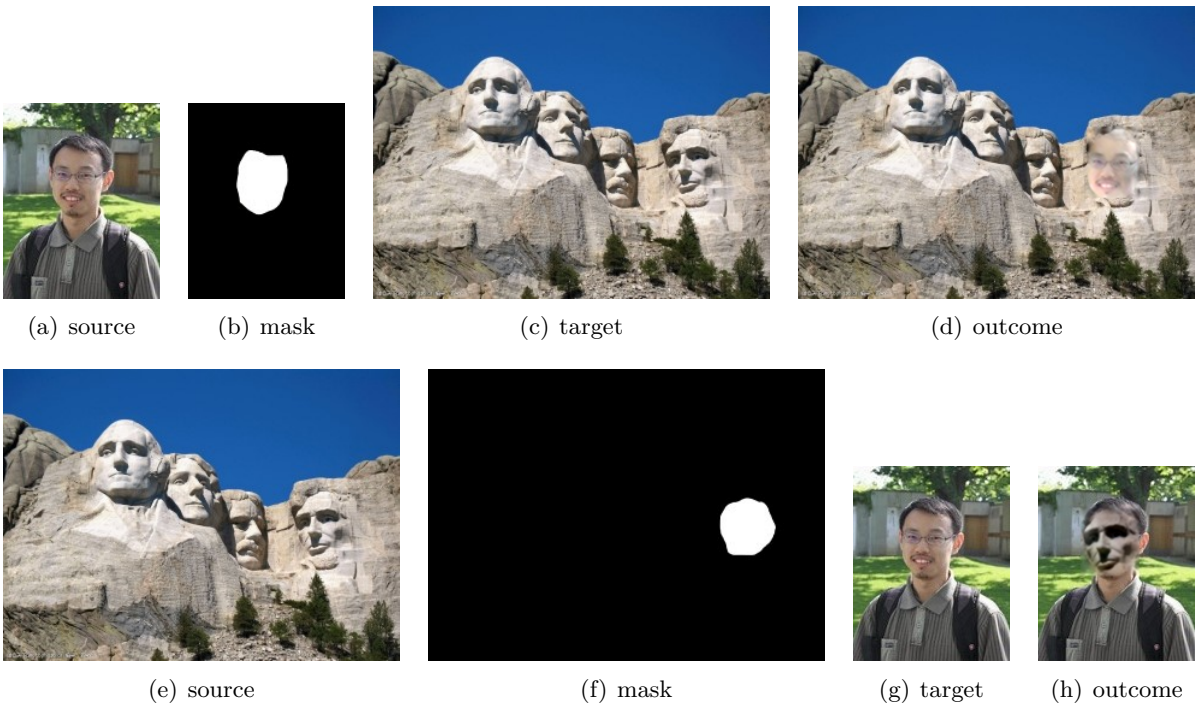
(a) source (b) mask (c) target (d) outcome

Figure 2: Importing gradients[2]

(a) source (b) mask (c) target (d) importing gradients (e) mixing gradients

Figure 3: **Importing gradients vs Mixing gradients** We could see from the experiments above that when inserting an transparent object into the destination mixing gradients performes better than simply importing gradients

[2]Some of the pictures are from http://cs.brown.edu/courses/csci1950-g/results/proj2/pdoran/



(a) source (b) mask (c) target (d) outcome

(e) source (f) mask (g) target (h) outcome

Figure 4: A funny case, just enjoy!

## 4 Fourier Approach [3]

To work on the problem using Fourier approach, we first need to transform the original problem (1) into the following approximation problem

$$\Delta f|_S = \text{div}\boldsymbol{V} \quad \text{with} \quad \nabla f \cdot \nu|_{\partial S} = 0 \tag{8}$$

As we solve the problem on the entire domain $S$, the *guidance field* $\boldsymbol{V}$ has the following form

$$V = \begin{cases} v & \text{in domain}\Omega \\ \nabla f^* & \text{Otherwise} \end{cases} \tag{9}$$

To use the Fourier approach, we should first consider the discrete bidimentional Fourier transformation of 2-D signal $Z$ with size $H \times W$

$$\hat{Z}(u,v) = \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} Z(x,y) e^{-2\pi i(\frac{xu}{W} + \frac{yv}{H})} \tag{10}$$

and its inverse,

[3] J. Matías Di Martino, Gabriele Facciolo, Enric Meinhardt-Llois, *Poisson Image Editing*, Image Processing On Line, 6 (2016), p.309-311.

$$Z(x,y) = \frac{1}{WH}\sum_{u=0}^{W-1}\sum_{v=0}^{H-1}\hat{Z}(u,v)e^{2\pi i(\frac{xu}{W}+\frac{yv}{H})} \tag{11}$$

Therefore the DFT has the following properties

$$\widehat{\frac{\partial Z}{\partial x}} = \frac{2\pi i}{W}u\hat{Z}, \widehat{\frac{\partial Z}{\partial y}} = \frac{2\pi i}{H}v\hat{Z} \tag{12}$$

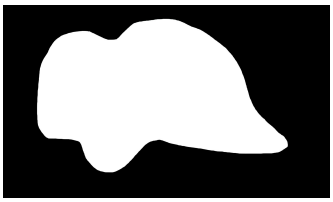
Then we are able to convert the problem (8) into the following form

$$\left[(\frac{2\pi i}{W})^2u^2 + (\frac{2\pi i}{H})^2v^2\right]\hat{f}_{uv} = \frac{2\pi i}{W}u\widehat{V^x}_{uv} + \frac{2\pi i}{H}v\widehat{V^y}_{uv} \tag{13}$$

Where $\widehat{V^x}, \widehat{V^y}$ denotes the partial derivatives of field $\boldsymbol{V}$. When $u = v = 0$, we could simply let $\hat{f}_{uv} = 0$, cause zero frequency just influence the solution by a constant value. After calculating the inverse of $\hat{f}$, we apply a shift of the solution to recover the color and the brightness of the image. Moreover, to hold the Neumann boundary conditions $\nabla f \cdot \nu|_{\partial S} = 0$, we could extend the input gradient field $\boldsymbol{V}$ so that the components $V^x$ and $V^y$ are antisymmetric with respect to $x$ and $y$ axis respectively. Then we have extended gradient fields which are four times bigger and become symmetric and periodic. After solving the reformulated system, we restrict the solution to the original domain and attain the solution we want. The details could be found in the source code of my project.

This approach is much faster than FDM written by myself. With the aid of this approach I am able to implement poisson image editing on images with higher quality(more pixels). The followings are some of my results.

(a) source



(b) mask

(c) target

(d) outcome

Figure 5: Importing gradients

(a) source (b) mask

(c) target (d) outcome

Figure 6: Mixing gradients

## 5 Algorithms

The following two pseudocodes are brief conclusion of my codes, for more details, just take a look at the directory "src".

**Algorithm 1** Poisson image editing with FDM

**Input:** source image, target image, mask image, rectangular domain $\Omega$, location
**Output:** interpolated image

1: **for** pixels in $\Omega$ **do**
2: **if** the pixel is not being masked by mask image **then**
3: Compute the gradient using the information of source image(importing gradients) or Compute the gradient using the information of both source and target image and pick the one with largest length(Mixing gradients)
4: **end if**
5: **end for**
6: Solve the poisson equation on the given $\Omega$ using Gauss-Seidel iteration
7: **Return:** the interpolated image

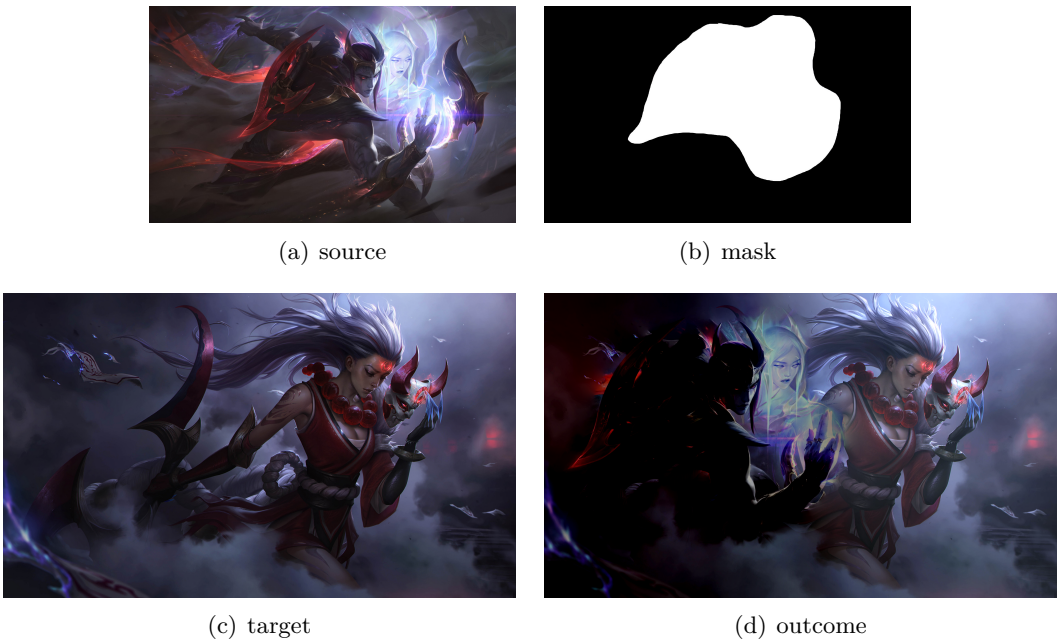
**Algorithm 2** Poisson image editing with FFT

**Input:** source image, target image, mask image, rectangular domain $\Omega$, location
**Output:** interpolated image

**for** pixels in the entire domain **do**
  **if** the pixel is not inside $\Omega$ **then**
    Compute the gradient using the information of target image
  **else**
    **if** the pixel is not being masked by the mask image **then**
      Compute the gradient using the information of source image(importing gradients) or Compute the gradient using the information of both source and target image and pick the one with largest length(Mixing gradients)
    **else**
      Compute the gradient using the information of target image
    **end if**
  **end if**
**end for**
extend the gradient fields to make it antisymmetric
compute the DFT of extended $f$ entry-wisly using the equation (13) and set the zero frequency as 0
compute extended $f$ through inverse Fourier transformation, compute the gap between average value of the target image and average value of the solution(outside of $\Omega$), then shift the solution by this gap.
**Return:** the interpolated image

# 6 Conclusion

The poisson image editing is sometimes a useful and funny ways to interpolate part of a image to another one, which gives a good way to recreate images. During the implementation of this project, I learnt new ideas for solving the poisson equations on a given domain and have so much fun creating lots of wierd(sometimes unbelievable) images. Feel free to check the directory "image" to see demos of my project. But apart from all the good things, we could still see that the approach sometimes leads to bad image quality. There are also some incompletion parts of my job, such as the Gauss-Seidel solver I wrote is not that efficent, the interfaces of my function are not that perfect, the codes I wrote are not that fluent and easy to understand.



(a) source (b) mask

(c) target (d) outcome

Figure 7: A case that the interpolant becomes too dark that it is hard to see the details of it. I guess the problem is the drastic brightness difference between selected area and outside the area of the target image. Moreover, the lack of brightness of the source image might also contributes to the bad quality.

# 7 References

[1] P. Pérez, M. Gangnet, and A. Blake, *Poisson image editing*, ACM Transactions on Graphics, 22 (2003), p.314-318.
[2] J. Matías Di Martino, Gabriele Facciolo, Enric Meinhardt-Llois, *Poisson Image Editing*, Image Processing On Line, 6 (2016), p.300-325.
[3] http://cs.brown.edu/courses/csci1950-g/results/proj2/pdoran/
[4] https://www.ipol.im/pub/art/2016/163/?utm_source=doi