Poisson Image Editing: Code Implementation report

Leonardo Pesce 223100001
CIE6004: Image Processing and Computer Vision

***Abstract*—The paper Poisson Image Editing [1] introduces a range of tools for seamless image editing using a generic interpolation approach. In this report, we will analyze how this works and has been implemented in Python.**

I. Introduction

The tools provided can be distinguished into two different classes. The first set of tools allows for the smooth importation of normal, opaque, and transparent image regions into a designated area. In contrast, the second set uses similar mathematical concepts to enable users to alter an image's appearance within a specified region, including changes to texture, illumination, and color. An essential feature of these tools is that they do not necessitate precise object delineation, making them versatile for various editing tasks, from simple touch-ups to complex photomontages. Furthermore, it's possible to combine these editing tools with cloning capabilities, and there is potential for expanding the range of editing options, such as adjusting the sharpness of objects to manipulate the focus in images. In total there are 5 tools: seamless cloning, texture flattening, local illumination change, local color change, and seamless tiling.

II. General functioning

We will now analyze the mathematical formulation behind these tools. All of them rely on image interpolation using a guidance vector field (for RGB images this is simply applied to each single color channel). We define S, a closed subset of $\mathbb{R}^2$ (the image definition domain), Ω a closed subset of S, with boundary $\partial\Omega$, f^* a scalar function defined over S except in Ω, and f the unknown part of f^* defined in Ω.

The easiest interpolation of f is defined by the solution to the following minimization problem:

$$\min_f \iint_\Omega |\nabla f|^2 \quad with\ f|_{\partial\Omega} = f^*|_{\partial\Omega} \tag{1}$$

where $\nabla = \left[\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right]$ is the gradient. The minimization problem has associated Euler-Lagrange equation:

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

where $\nabla = \frac{\partial^2}{\partial x^2}, \frac{\partial^2}{\partial y^2}$ is the Laplacian operator.

At this point adding further constraints will result in the desired modifications of the image. This constraints are expressed in the form of a guidance field vector $\mathbf{v}$. This changes the previous equations to

$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \quad with\ f|_{\partial\Omega} = f^*|_{\partial\Omega} \tag{3}$$

which has a unique solution given by the following Poisson equation with Dirichlet boundary conditions.

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

where $\nabla = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$ is the divergence of $\mathbf{v} = (u, v)$.

Since we are talking about discrete images, we can find this solution with discrete Poisson equations.

Playing with $\mathbf{v}$ will result in the various different effects, let's see how these are done.

III. Seamless Cloning

In this case the guidance field $\mathbf{v}$ is taken to be the gradient of another image (g). This allows us to import the gradient of g and copy it to the original image in a smoother way (the color may be slightly distorted).

The problem becomes:

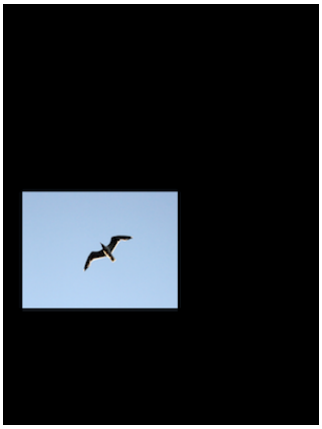
$$\Delta f = \nabla g\ over\ \Omega \quad with\ f|_{\partial\Omega} = f^*|_{\partial\Omega} \tag{5}$$

The Python implementation follows is described in algorithm 1

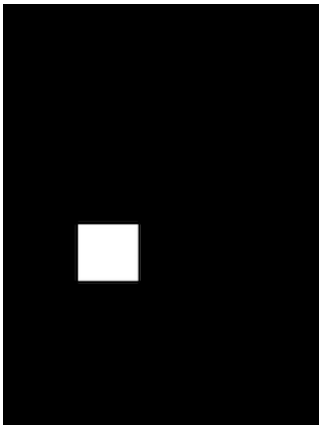
Algorithm 1 pseudocode for seamless cloning

```
compute matrix A (Discrete Poisson equation)
for number of color channels in picture do
    computes ∇g and ∇f
    if selected mode = alpha then
        ∇g = α∇f + (1 − α)∇g
    else
        ∇g_{i,j} = max(∇f_{i,j}, ∇g_{i,j})
    end if
    construct vector b with the newly computed gradient
    compute the solution to Ax = b
end for
reconstruct the image from the solutions
```

The following images represent the result of the algorithm. For the first set we will show also the mask, while for the other set, we will only show the source, target, and final result.



(a) image that we want to clone



(b) mask of the first image

(c) Target image for the cloning

(d) Image with added seagull

Fig. 1: Seagull in the sky

The tool works excellently in importing the image to the new background and blending with it. It also does a great job of maintaining the features of the original image (this is given by the fact that it imports the gradient which is what permits us to distinguish features in an image).

(a) target image

(b) source image

(c) result image

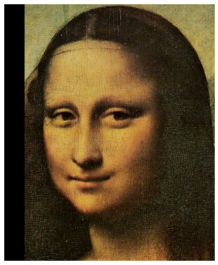
Fig. 2: Fighting jet in the sky

(a) target image

(b) source image

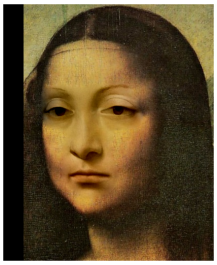
(c) result image

Fig. 3: Statue of Liberty under the sea



(a) target image

(b) source image

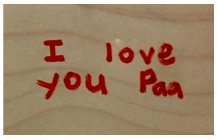


(c) result image

Fig. 4: Mona Lisa and Ginevra de' Benci

(a) target image

(b) source image



(c) result image

Fig. 5: The phrase "I love you Paa" on wood

(a) target image

(b) source image

(c) result image

Fig. 6: gradient of f on brick wall

The images are made with both the method, mixing gradient and max, based on the best visual result obtained.

IV. Texture flattening

To flatten the feature of an image we pass the ∇f^* (the image gradient) through a filter function $M(x)$, in this case $\mathbf{v}$ becomes

$$\mathbf{v}(x) = M(x)\nabla f^* \tag{6}$$

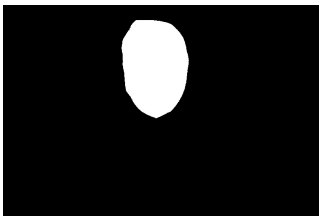
As $M(x)$ we used an edge detection tool to detect the image's most prominent feature inside the mask. In this way, we keep only the most important features while flattening the other ones.

Here's the pseudocode

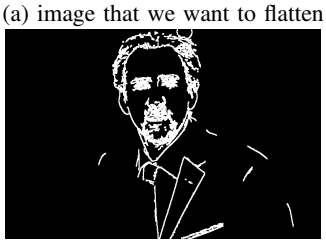
Algorithm 2 pseudocode for texture flattening

compute matrix A (Discrete Poisson equation)
compute edge of the image $M(x)$ (inside the mask)
for number of color channels in picture **do**
  computes ∇g and ∇f
  $\mathbf{v} = M(x)\nabla f^*$
  construct vector b with $\mathbf{v}$
  compute the solution to $Ax = b$
end for
reconstruct the image from the solutions

(a) image that we want to flatten



(b) mask of the image flattening



(c) Edge of the image

(d) Flattened image

Fig. 7: Flattening of the image of a person

As we can see the main features of the faces, which are visible in white in 7c are preserved in the flattening filter, while the rest is blended together.

V. Illumination change

By applying a non-linear transformation to the gradient (such as the logarithm), before solving the discrete Poisson equations, it is possible to highlight hidden features and reduce reflection. In this case, the guidance field defined in the log-domain becomes:

$$\mathbf{v} = \alpha^{\beta}|\nabla f^*|^{-\beta}\nabla f^* \tag{7}$$

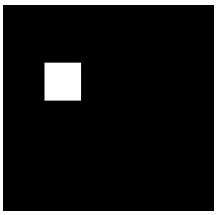
where $\beta = 0.2$ and $\alpha = 0.2 * average(\|\nabla f^*\|)$

Here's the pseudocode followed by the results

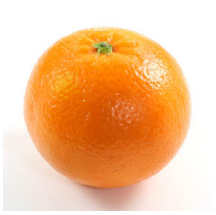
Algorithm 3 pseudocode for illumination change

compute matrix A (Discrete Poisson equation)
for number of color channels in picture **do**
  change the image to the log domain
  compute ∇f and $|\nabla f|$
  compute α and β
  $\mathbf{v} = \alpha^{\beta}|\nabla f^*|^{-\beta}\nabla f^*$
  construct vector b with $\mathbf{v}$
  compute the solution to $Ax = b$
  revert the image to the original domain
end for
reconstruct the image from the solutions

(a) source image



(b) mask image



(c) result image

VI. Color change

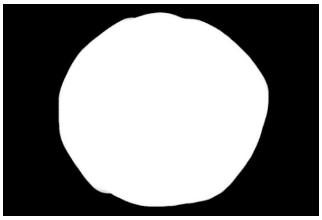
The method is also applicable to color manipulations. Given an original image and a color-changed version of itself, the two can be mixed to obtain a new image with only the masked part that changed color. One image provides the destination function f^* outside Ω while the other provides the function g to be modified inside Ω

Algorithm 4 pseudocode for color change

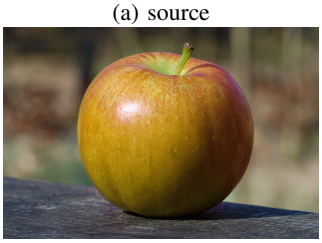
compute matrix A (Discrete Poisson equation)
compute image in the new color spectrum
for number of color channels in picture **do**
  computes ∇f
  construct vector b with ∇f and the color changed image
  compute the solution to $Ax = b$
end for
reconstruct the image from the solutions

The method as shown in 9 proves to be functioning even if the mask for the object is not precise. This saves a lot of time in image modification since we can skip the step of precisely contouring the object before changing its color.

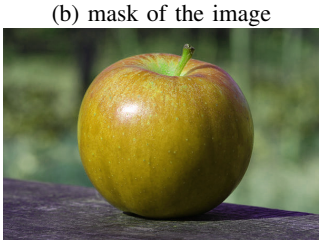
(a) source



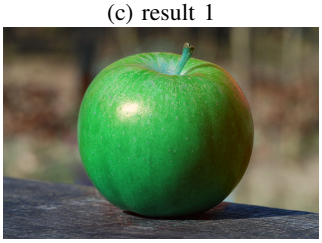
(b) mask of the image



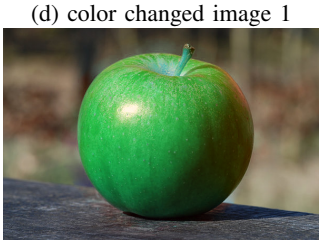
(c) result 1



(d) color changed image 1



(e) result 2



(f) color changed image 2

Fig. 9: Changing color of the apple without affecting the rest of the image

VII. Seamless Tiling

In the case of a rectangular domain Ω, if we enforce periodic boundary conditions, the image becomes tileable when the Poisson solver is applied. In the paper, were chosen

$$f^*_{north} = f^*_{south} = 0.5(g_{north} + g_{south}) \tag{8}$$

$$f^*_{east} = f^*_{west} = 0.5(g_{east} + g_{west}) \tag{9}$$

This condition as visible in 10 is smoothened at the border to soften the transition between two tiles.

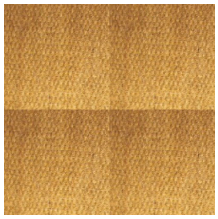
Algorithm 5 pseudocode for seamless tiling

```
compute matrix A (Discrete Poisson equation)
for number of color channels in picture do
    compute ∇f
    change the border of the image
    construct vector b with ∇f and the new boundaries
    compute the solution to Ax = b
end for
reconstruct the image from the solutions
tile the image with itself
```

VIII. Conclusions

The paper discusses the use of guided interpolation to edit image selections seamlessly. It offers various tools for making changes to selected image regions, including replacement, mixing with other images, and altering aspects like texture, illumination, or color. The key feature is that precise object delineation is not required, distinguishing it from classic tools. These tools can be used for both minor touch-ups and complex photo editing. A Python implementation was created and run on all of the experiments proposed, proving in practice the theoretical discoveries.

(a) source image



(b) tiled source

(c) tiled result image

Fig. 10: Making an image tileable

References

[1] P. Pérez, M. Gangnet, and A. Blake, "Poisson image editing," *ACM Trans. Graph.*, vol. 22, no. 3, p. 313–318, jul 2003. [Online]. Available: https://doi.org/10.1145/882262.882269