Contents

1 Introduction 2

2 Theory 2

3 Deficiencies and Improvements 4

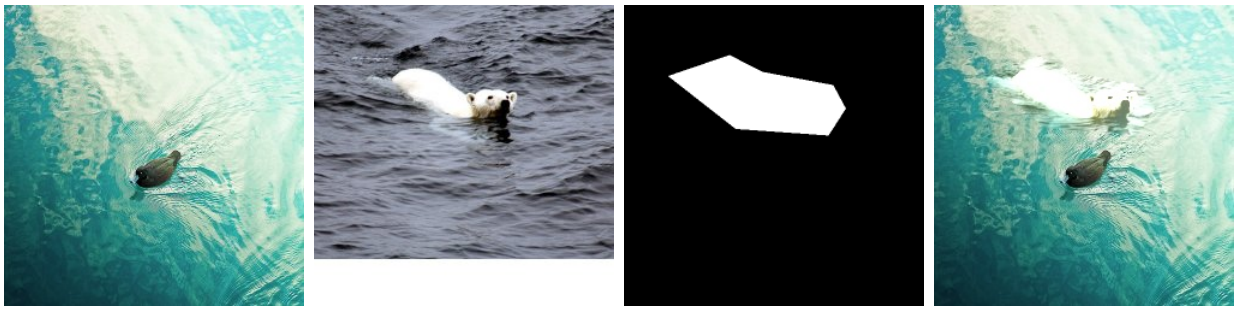
4 Python Codes 5

5 References 7

1 Introduction

The idea of Poisson Image Editing (PIE) was first proposed in a 2003 paper. [1] Based on solving discrete Poisson equations, it is a generic interpolation technique that can achieve seamless editing of image regions without " precise object delineation". [1] The paper suggests two main applications of PIE: seamless cloning and selection editing (i.e., local color changes). The implementer only implemented a naive form of seamless cloning.

Seamless cloning is to copy an image region from a foreground image and seamlessly paste it into another image region $\mathbf{\Omega}$ of a background image. Fig. 1 suggests a naive presentaion of seamless cloning. The background image ($\mathbf{B}$), foreground image ($\mathbf{F}$) and matte image ($\mathbf{M}$) are three 250×250 8-bit RGB images [2]. The matte image indicates $\mathbf{\Omega}$. Overlay $\mathbf{M}$ on $\mathbf{F}$, then pixels of $\mathbf{F}$ that overlaps with the white pixels (255, 255, 255) of $\mathbf{M}$ are to be copied from. Overlay $\mathbf{M}$ on $\mathbf{B}$, then pixels of $\mathbf{B}$ that overlaps with the white pixels of $\mathbf{M}$ are to be pasted into. In the context of $\mathbf{B}$, $\mathbf{F}$ and $\mathbf{M}$, the polar bear will be copied and pasted into above the duck.



(a) Background Image **(b)** Foreground Image **(c)** Matte Image **(d)** Output Image

Figure 1: A naive example of seamless cloning

2 Theory

Fig. 2 specifies the mathematical presentation of seamless cloning, i.e., the process of implementing interpolation through guidance vector field. [1] $\boldsymbol{f}^*$ is the function of $\mathbf{B}$ outside $\mathbf{\Omega}$ and $\boldsymbol{f}$ is the unknown function of $\mathbf{B}$ in $\mathbf{\Omega}$. $\boldsymbol{\partial\Omega}$ is the rim of $\mathbf{\Omega}$. Let $\boldsymbol{g}$ be the function of of $\mathbf{F}$ corresponding to $\mathbf{\Omega}$. We would like to replace $\mathbf{S}$ with $\boldsymbol{g}$ in the region of $\mathbf{\Omega}$. The solution is to extract a vector field $\mathbf{v}$ from $\boldsymbol{g}$ and let $\mathbf{v}$ "guide" the gradients in $\mathbf{\Omega}$.

There are two expectations of seamless cloning:

1. The gradients in $\mathbf{\Omega}$ should be as close to $\mathbf{v}$ as possible.
2. The pixel values of $\boldsymbol{\partial\Omega}$ should not be changed.

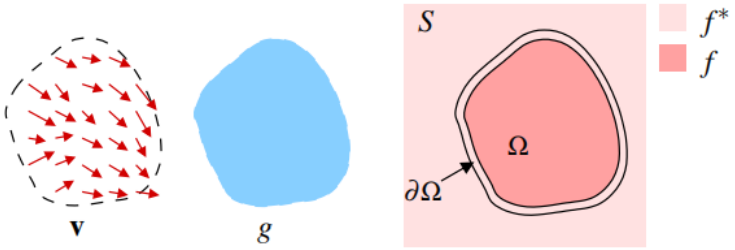


Figure 2: Guidance vector field

The first requirement accomplishes the mission of "copying and pasting" $\boldsymbol{g}$, and the second aims to minimize the existence of seams around $\boldsymbol{\partial\Omega}$. Combine them together, we get:

$$\begin{aligned} &\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \\ &\text{s.t.} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega} \end{aligned} \tag{1}$$

For the above minimization problem, the Poisson equation (solution) with Dirichlet boundary conditions is:

$$\Delta f = \text{div}\mathbf{v} \text{ over } \Omega \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega} \tag{2}$$

where $\Delta.$ is the Laplacian operator $\frac{\partial^2 .}{\partial x^2} + \frac{\partial^2 .}{\partial y^2}$. Under 2-D discrete conditions, arbitrarily choose one vector $w = (x, y)$ from $\boldsymbol{v}$. Then we have

$$\begin{aligned} \frac{dw}{dx} &= g(x+1, y) - g(x-1, y) = G_x(x, y) \\ \frac{dw}{dy} &= g(x, y+1) - g(x, y-1) = G_y(x, y) \end{aligned} \tag{3}$$

$$\begin{aligned} \frac{\partial^2 w}{\partial x^2} &= G_x(x, y+1) - G_x(x, y-1) \\ &= g(x, y+2) - g(x, y) - g(x, y) + g(x, y-2) \\ \frac{\partial^2 w}{\partial y^2} &= g(x+2, y) - g(x, y) - g(x, y) + g(x-2, y) \end{aligned} \tag{4}$$

$$\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} = -4g(x, y) + g(x+2, y) + g(x-2, y) + g(x, y+2) + g(x, y-2) \tag{5}$$

Under discrete conditions, dx should be expressed as $x + 1$ instead of $x + 2$. Equation (5) can be further written as:

$$\Delta w = -4g(x, y) + g(x+1, y) + g(x-1, y) + g(x, y+1) + g(x, y-1) \tag{6}$$

and div$w$ can be similarly expressed.

A discrete example is shown as following [3]: say X is a 4×4 grayscale image expressed as Equation (7).

$$X = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \\ x_{13} & x_{14} & x_{15} & x_{16} \end{bmatrix} \tag{7}$$

We would like to use (divx_6, divx_7, divx_{10}, divx_{11}), which are obtained from the guidance gradient field, to update (x_6, x_7, x_{10}, x_{11}). Then the Poisson equations will be:

$$\begin{cases} x_2 + x_5 + x_7 + x_{10} - 4x_6 = \text{div}x_6 \\ x_2 + x_5 + x_7 + x_{10} - 4x_6 = \text{div}x_7 \\ x_2 + x_5 + x_7 + x_{10} - 4x_6 = \text{div}x_{10} \\ x_2 + x_5 + x_7 + x_{10} - 4x_6 = \text{div}x_{11} \end{cases} \tag{8}$$

When implementing the codes, express the Poisson equations in a matrix form $Ax = b$ and use numpy to solve them.

3 Deficiencies and Improvements

As mentioned in the above content, this implementation is still naive. This implementation worked well in Fig. 1, but it might not obtain excellent overall performance. As shown in Fig. 3, when trying to "copy" the Bently and paste it into the park, much irrelevant content around the Bently was also pasted on. This kind of deficiency can be attributed to the similarity between the textures of **F** and **B**. To overcome this deficiency, "mixed gradients seamless cloning" has to be adopted. [1]

(a) Background Image **(b)** Foreground Image **(c)** Matte Image **(d)** Output Image

Figure 3: A bad output example of this implementation. a, b and c were resized from (4032, 3024) to (200, 150) before d was output.

4 Python Codes

```
from cv2 import imwrite
import numpy as np
import cv2
from scipy import sparse
from scipy.sparse.linalg import spsolve

F = 'D:\\Research_Sam\\Poisson_Image_Editing\\foreground.jpg'
B = 'D:\\Research_Sam\\Poisson_Image_Editing\\background.jpg'
M = 'D:\\Research_Sam\\Poisson_Image_Editing\\matte.png'

F_src = cv2.imread(F)
B_src = cv2.imread(B)
M_src = cv2.imread(M)

# Remember to normalize F, B, M's sizes

assert F_src.shape == B_src.shape == M_src.shape

W, L, _ = F_src.shape
n = 0

omega = [[False for _ in range(L)] for _ in range(W)]
omega_coords = []
idx_dict = {}
count = 0
for i in range(W):
    for j in range(L):
        if M_src[i][j][0] == 255:
            omega[i][j] = True
            n += 1
            omega_coords.append((i, j))
            idx_dict[(i, j)] = count
            count += 1

nbhd_omega = []
for _ in omega_coords:
    y = _[0]
    x = _[1]
    tmp = [False, False, False, False]
    if omega[y - 1][x] == True: tmp[0] = True
    if omega[y + 1][x] == True: tmp[1] = True
    if omega[y][x - 1] == True: tmp[2] = True
    if omega[y][x + 1] == True: tmp[3] = True
    nbhd_omega.append(tmp)
```

```
div = [np.zeros(n) for _ in range(3)]
for i in range(3):
    for _ in range(n):
        x, y = omega_coords[_]
        div[i][_] = -4 * F_src[x][y][i] + F_src[x + 1][y][i] + F_src[x - 1][y][i]
            + F_src[x][y + 1][i] + F_src[x][y - 1][i]

A = [sparse.lil_matrix((n, n)) for _ in range(3)]

for _ in range(3):
    for idx in range(n):
        A[_][idx, idx] = -4

b = [np.zeros(n) for _ in range(3)]

for cl in range(3):
    for i in range(n):
        b[cl][i] = div[cl][i]
        nbhd_coords = [(omega_coords[i][0] - 1, omega_coords[i][1]),\
                       (omega_coords[i][0] + 1, omega_coords[i][1]),\
                       (omega_coords[i][0], omega_coords[i][1] - 1),\
                       (omega_coords[i][0], omega_coords[i][1] + 1)]
        if all(nbhd_omega[i]):
            for _ in range(4):
                A[cl][i, idx_dict[nbhd_coords[_]]] = 1
        else:
            for _ in range(4):
                if nbhd_omega[i][_] == False:
                    b[cl][i] -= B_src[nbhd_coords[_]][cl]
                else:
                    A[cl][i, idx_dict[nbhd_coords[_]]] = nbhd_omega[i][_] * 1

for _ in range(3):
    A[_] = A[_].tocsc()

X = [[] for _ in range(3)]
for _ in range(3):
    X[_] = spsolve(A[_], b[_])


new = np.zeros((W, L, 3))
for _ in range(3):
    new[:, :, _] = B_src[:, :, _]

for _ in range(n):
    for cl in range(3):
        x, y = omega_coords[_]
```

```
        new[x][y][cl] = min(255, X[cl][_])

new = np.array(new, dtype = 'uint8')
cv2.imwrite('D:\\Research_Sam\\Poisson_Image_Editing\\Output_Sam.jpg', new)
```

5 References

[1] Pérez, P., Gangnet, M., and Blake, A. 2003. Poisson Image Editing. *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, 313-318.

[2] link

[3] link