Seamless Cloning Reproduced in Python

I. Theory

Seamless cloning, or Poisson image matting, is an image blending scheme proposed by Microsoft Research UK on SIGGRAPH 2003. The basic idea of the method is to address the problem of too obvious matting traces when traditional means are applied. By operating seamless cloning, the transition between foreground and background should be as imperceptible as possible, while the foreground's texture feature should be preserved. To meet both of the above requirements, a Poisson equation and a corresponding boundary condition are provided in the paper. Here I briefly describe the theoretical basis by introducing key variables and their notations at the very beginning.

Suggest that a foreground image g is to be overlaid to a region of a background image S. $f*$ stands for the known pixel values of S, and f stands for the unknown pixel values within the blended area Ω. The border of Ω which doesn't actually belong to Ω is notated with $\partial\Omega$. p is any single pixel, and one of its four-neighborhood pixels is expressed as q. As is aforementioned, the boundary condition is given right below:

$$f_p = f*_p \quad for\ every\ p \in \partial\Omega$$

The boundary condition indicates that, the border of the foreground area shares the same pixel values with the background pixels of corresponding positions. Since $f*_p$ is given, the border $\partial\Omega$ is definite. With this premise, the pixel values within Ω can be inferred by establishing the Poisson equation. The original form of the Poisson equation for this scenario is given as:

$$\nabla^2 f = div(grad(g))$$

$\nabla^2 f$ is the second derivative of f, the pixels within Ω which is to be solved, and $div(grad(g))$ represents the divergence of the gradient field of the original foreground image g, which is a known condition. Both sides of the equation essentially extract the texture information from the foreground image, yet the pixels within Ω is unknown beforehand, and the original values are used to guide the recovery of textures, which guarantees that the textures of the original foreground image can be recovered when blended. Note that we have to jointly consider the border smoothness and the texture preservation, hence this equation is solvable only when the border condition, as a constraint, is given or can be calculated.

The second derivative of pixels can be computed by applying the Laplacian to the space domain of the image. It is a convolution process, for Laplacian is a 3×3 kernel. Besides, Laplacian is a linear kernel and can be written as:

$$\nabla^2 f_p = f_{q_{up}} + f_{q_{down}} + f_{q_{left}} + f_{q_{right}} - 4f_p$$

In this case, the Poisson equation degenerates into a problem of solving a linear system, as is shown below:

$$\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b}$$

$\boldsymbol{A}$ is a square Laplacian matrix, a sparse matrix considering the nearest neighbors relationships. $\boldsymbol{b}$ is a constant vector where every element represents the second derivative of each pixel in g. The order of the elements in $\boldsymbol{A}$ and $\boldsymbol{b}$ should be perfectly corresponded. The result $\boldsymbol{x}$ can be obtained by solving the non-homogeneous sparse linear equations.

However, the above formulation is not rigorous enough because of the known pixel values composing the border $\partial\Omega$. In fact, when the linear equations are structured, $\boldsymbol{A}$ ignores all neighbors belonging to $\partial\Omega$, keeping only the nearest neighbors with unknown pixel values. The constants are moved to the right-hand side of the equation, which are subtracted by the corresponding constant in $\boldsymbol{b}$. For any pixel f_p of Ω, the expanded form of the equation can be represented by:

$$\sum_{q \in \Omega} f_q - 4f_p = \sum_{q \in g} g_q - 4g_p - \sum_{q \in \partial\Omega} f*_q$$

As is shown in the above equation, all the unknowns are placed left, while the constants are placed right. The Python program of Poisson matting can be realized based on this equation.

II. Program Implementation & Results

2.1 Program Implementation

Image Preprocess. The foreground as well as background image are read and resized so that the matrix $\boldsymbol{A}$ isn't too large for the calculation, and neither of them are larger than the display resolution. The matting center can be chosen arbitrarily by pressing the left mouse button at any point on the background image.

Border Assignment. A copy of the foreground will be created, of which the border pixels will be assigned at the beginning stage. As is mentioned, the values are equal to the background pixels of the same positions.

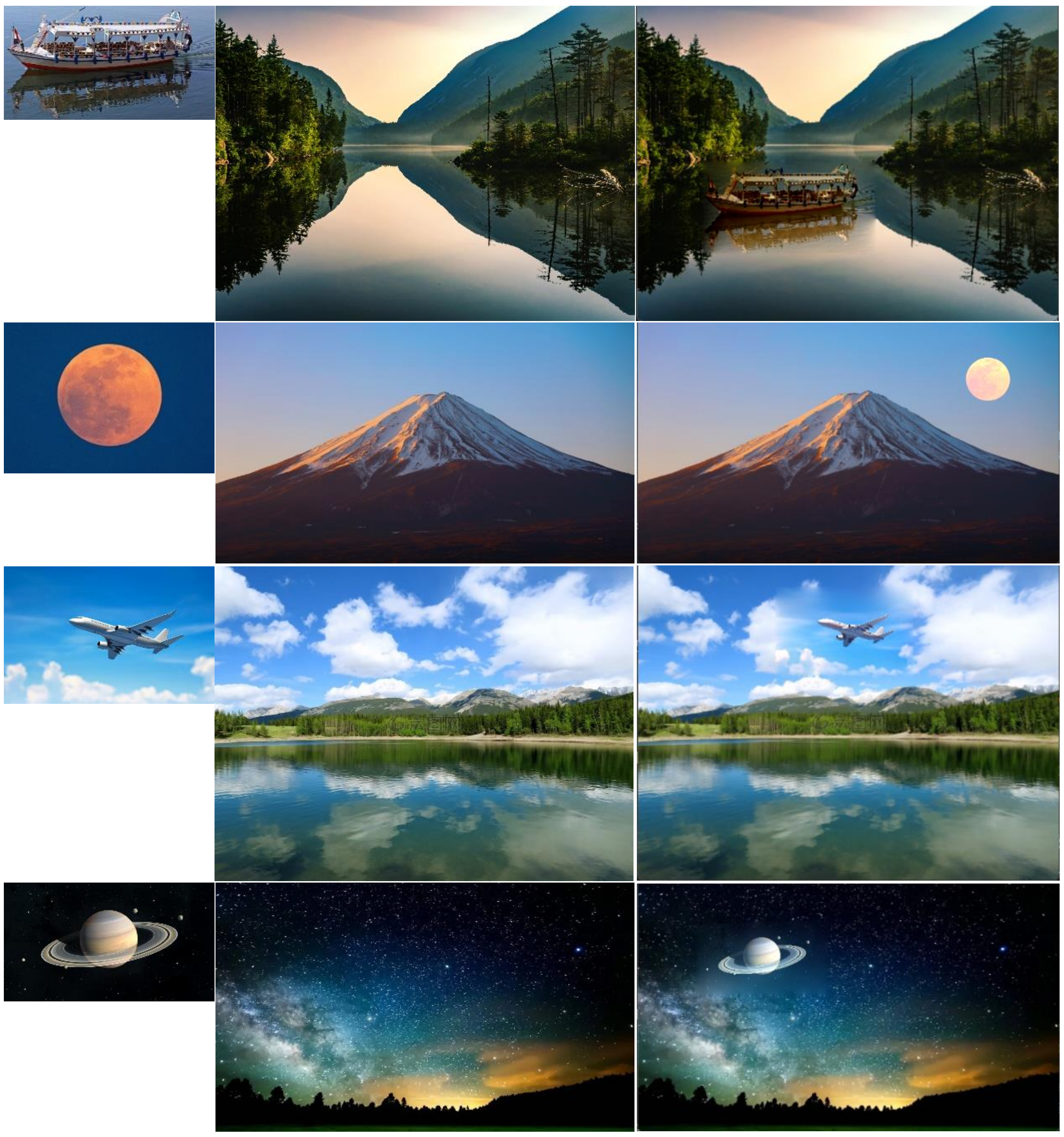
Matrix Construction. The matrix, or vector $\boldsymbol{b}$ is constructed first. The number of elements in $\boldsymbol{b}$ is equal to the number of pixels in Ω. For most of the pixels which have no neighbors belonging to the border, the values are identical to the second derivative of g_p. As for pixels whose neighborhoods intersect with $\partial\Omega$, i.e. pixels at the outermost of Ω while still within it, their values equal $div(grad(g_p))$ subtract the neighbors which belongs to $\partial\Omega$.

The matrix $\boldsymbol{A}$ has the same height and width, both identical to the number of pixels in Ω. Values along the back diagonal of $\boldsymbol{A}$ are set to -4, and for every $\langle p, q\rangle$ whose q is within Ω , $\boldsymbol{A}(p, q)$ is assigned with 1. All the other elements in $\boldsymbol{A}$ are zeroed.

Solving Equations. I utilize a SciPy function to solve the sparse linear equations, namely `scipy.sparse.linalg.spsolve()`. It is proved to be much quicker in finding solutions than a comparable function `numpy.linalg.solve()` in NumPy. The examples of the program output are demonstrated in 2.2.

2.2 Results and Analysis

Here I provide four examples of my program output.



The images from left to right are: the foreground, the background, and the blended image.

These examples show that the normal Poisson matting scheme can achieve satisfactory image blending results with holeless foreground images. For foreground images with much fewer pixels constituting the main objects, mixed Poisson matting suits better, as the output of normal method tend to be blurry at the rest part of the foreground, where the gradients are smaller than those of the same part in the background image.

You’re welcomed to try my program, and please make sure to change the file paths and adjust the images’ scaling factors before running the program if you try other foreground and background images. You can find the annotation at the beginning part of the program. Also, please make sure to select a blending center with your left mouse button at the start of operation and press ‘Esc’ or close all the existing image windows to finish selecting, otherwise the runtime will be terminated.