Assignment 2 Poisson Image Editing

fanghzh1@shanghaitech.edu.cn

October 30, 2020

Contents

1 Introduction

Achieve a basic Poisson image editing pipeline with three parts,which can blending the source image to the target image with designated location with the Poisson equation method. For the first part,based on the given general boundary $\partial\Omega_0$, than optimized a better boundary $\partial\Omega_{opt}$,secondly, use Poisson to adjust the each pixel value of the source image inside $\partial\Omega_{opt}$,finally ,blending this into the target image location then output the final result.

2 Implement

2.1 Input/Output Format

2.1.1 Input

Roughly speaking, Poisson image blending is embedding the source image into the target image. As showing in 1, we want to embedding the 2 into the Ω place, so we need to give

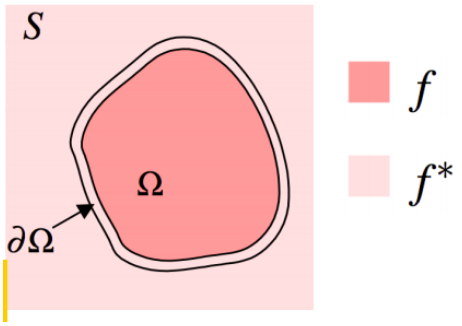

Figure 1: Target

Figure 2: Source

the $\partial\Omega$. Before that, first we should give a place mapping function from source image to the target image.
Given an assumption that the S have a larger size of width and height than g:

1 Discuss the simplest case, both S and g are rectangles, which means we only need to build the mapping relationship of the points on the 4 corners.

2 Discuss adding $\partial\Omega$. In this case, g is a 2D plane enclosed by $\partial\Omega$ and surrounded by a rectangle G. At this point, we need to use two parts to describe g, mask and basic.

3 Regard G as an image, mask is an image of the same size as G, and all values set to 0 or 1, 0 indicate that these pixels are not included in g, 1 means that it belongs to g. In particular, the pixels $\in \partial\Omega$ are set to 1.

2.2 Build and solve linear system

2.2.1 Vector field and scalar field

Only consider a single channel of image, we can use a function I, assigning the coordinate to pixel value, which is a scalar, then define a scalar field as follow:

$$\begin{aligned} I(x,y):\ \mathcal{R}^2 &\to \mathcal{R} \\ \frac{\partial I}{\partial x} = u(x,y) &: \mathcal{R}^2 \to \mathcal{R} \\ \frac{\partial I}{\partial y} = v(x,y) &: \mathcal{R}^2 \to \mathcal{R} \end{aligned}$$

More over:

$$\begin{aligned} \text{Curl of the gradient} \nabla \times \nabla I(x,y) &= \frac{\partial^2}{\partial y \partial x} I(x,y) - \frac{\partial^2}{\partial x \partial y} I(x,y) \\ \text{Divergence of the gradient} \nabla \cdot \nabla I(x,y) &= \frac{\partial^2}{\partial x^2} I(x,y) + \frac{\partial^2}{\partial y^2} I(x,y) \\ &= Laplacian = \Delta I(x,y) \end{aligned}$$

2.2.2 Poisson blending

The key idea is that when blending, retain the gradient information as best as possible, back to the format given as 1, in which we define the source image g as a integrate vector field , form the Poisson blending as integrate vector field ∇g with Dirichlet boundary conditions f^*
So, we need to

$$\begin{aligned} \nabla f = \nabla g \text{ inside } \Omega \\ f = f^* \text{ at the boundary of } \partial\Omega \\ \min_f \iint_\Omega |\nabla f - v|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega} \end{aligned}$$

For the final formula,the first part would make the gradient of f looks like the vector field v,f is equivalent to f^* at the boundaries, for now we induce the Poisson equation:

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

For discrete images:

$$\begin{aligned} \text{For each piexl:} (\nabla f)(x,y) &= (\nabla . v)(x,y) \\ -4f_p + \sum_{q \in N_p} f_q &= (u_x)_p + (v_y)_p \end{aligned}$$

Equivalent to a linear system $Af = b$

2.2.3 Solving the linear system

Convert the system to a least-squares problem:

$$E_{LLS} = \|Af - b\|^2 = f^T(A^TA)f - 2f^T(A^Tb) + \|b\|^2$$
$$\frac{\partial E}{\partial f} = (A^TA)f - A^Tb$$

Where we use the conjugate gradient descent to solve this problem.

2.3 Boundary optimization

2.3.1 Target of optimization

For we want to reduce the color variance along the boundary, so the target should have

$$minE(\partial\Omega, k) = \sum_{p\in\partial\Omega}((f_t(p) - f_s(p)) - k)^2$$
$$s.t. \in \partial\Omega$$
$$f(p) : (x, y) \in R^2 \to (r, g, b) \in R^3$$

E represent the color deviation of the boundary.

2.3.2 Iterative method

1 Initialize Ω as Ω_0

2 Given the current boundary $\partial\Omega$

$$\frac{\partial E(\partial\Omega, k)}{\partial k} = 0$$
$$k = \frac{1}{|\partial\Omega|}\sum_{p\in\partial\Omega}(f_t(p) - f_s(p))$$

$|\partial\Omega|$ is the length of the boundary, so k mean average color difference on the boundary.

3 Given the current k, we optimize the boundary $\partial\Omega$.In this part, we should find the shortest path which have the least with with given k.To deal with this problem, we should determine the basic element.

Node Regarded as each pixel as a node with at most 4 edge connect to the nearby pixel with 4 edges.

Edge The value of edge is compute by the distance of two nodes with the color space, form like the L2-norm.

Start Obviously, all pixels are connected with the assumption showing before. For we want to draw a enclose $\partial\Omega$, we using a cut C connect the Ω_{obj} and Ω_0, then remove all edges crossing the cut.

End All path should ending at the neighboring pixels on the other of cut.

4 Repeat steps 2 and 3 until the E does not decrease.

2.3.3 Complex

Review all steps listed before, assume we have M pixels in the cutting C, for each $m \in M$, we need to find a best enclose $\partial\Omega_m$, finally get the globally $minE$ as the $\partial\Omega_{best}$.
Assume we do x_i iteration for $m_i \in M$, even when m_i only have 1 ending target, we still need $\sum_i^{m \in M} x_i$ times finding process with the complexity $O(nlogn)$ with direct using Dijkstra.We use a simple parallel work to do this, much faster.

3 Experiments

3.1 Boundary optimization

3.1.1 Getting the basic outline

Using a simple gui, draw a list of points in order, than get a convex poly as the mask part.

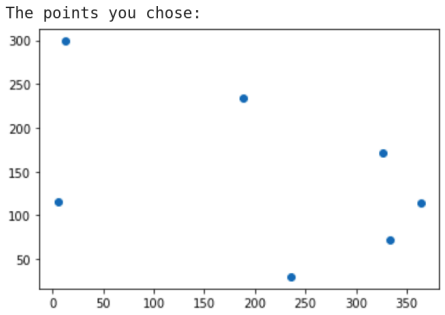

Figure 3: points

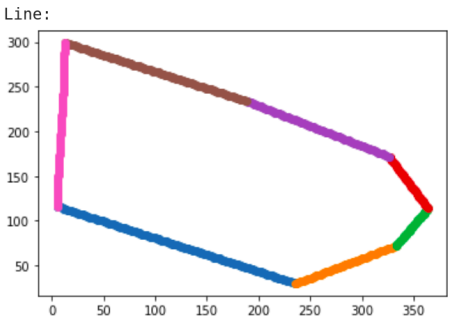

Figure 4: line

3.1.2 Start with K

Do this twice, get the inline and outline, then using the E function to start with k. As in 5, where the green is the object, the red part is fine to adjust, and the black was not selected part.Then the close connect between from black to green is the cut.Then as showing in 6, the blue and white line is the two side of in this cut.

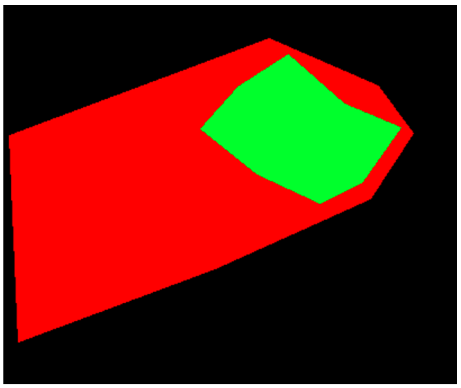

Figure 5: Outline

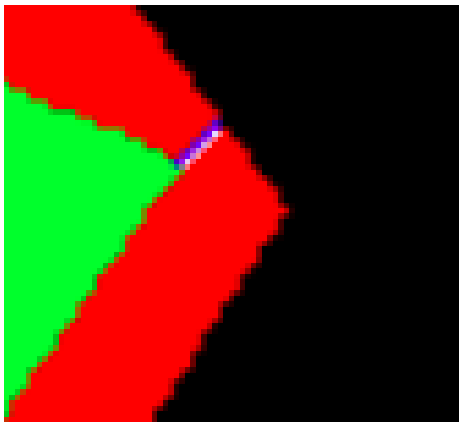

Figure 6: cut

3.1.3 Optimization

After this, write the information to the file as $Node$,$Edge$,$task$, using go version program to find the path as:

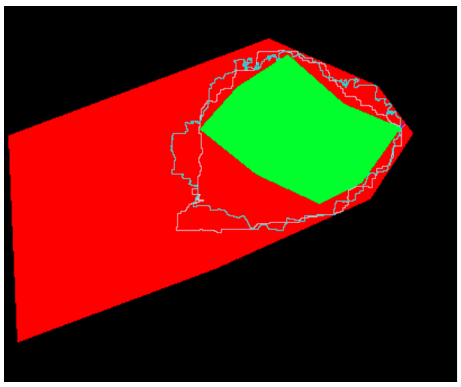

Figure 7: Opt1

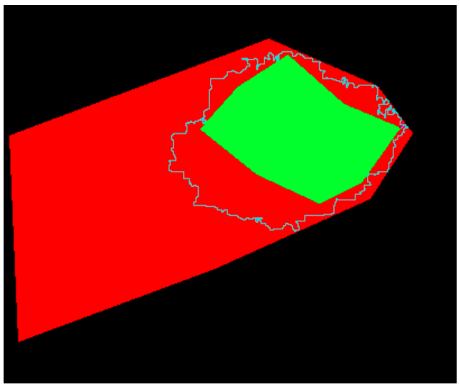

Figure 8: Opt2

3.1.4 Get mask

Using the best k, we get could get the mask we need, for we do not draw a good basic mask, the result is just not well, but should say that this algorithm basically converges once.

Figure 9: Mask

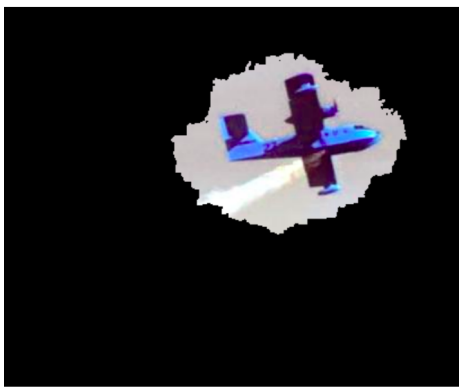

Figure 10: photo

3.2 Blending

3.2.1 Form

Only a few tricks need to talk about:

Matrix For computation, instead of build each line of the linear matrix, use $Lf = Lg$ instead of $Af = g$.

Solver *spare.linalg.cg* is reasonable, but *spsolve* is more quick in testing(almost $\times 3$)

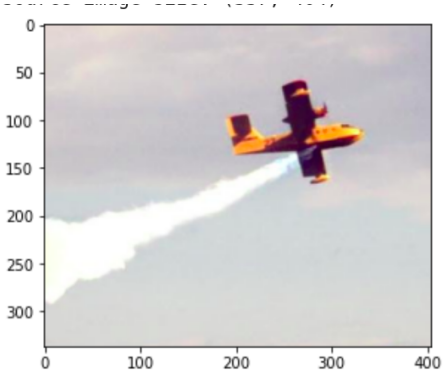

Figure 11: Basic

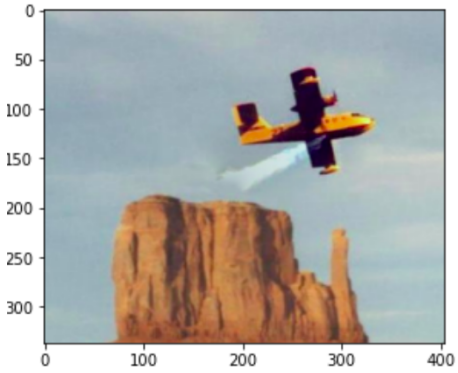

Figure 12: Blending

4 Conclusion

4.1 algorithms

There several special algorithm:

Draw a line Define at the first part of line.ipynb, Bresenham's algorithm, achieve to draw the outline of the mask showing as before.

Finding the other side of a line To make the other side of the cut. we use the cross product to define if a point is on the target side of a line, the detail write in markdown in ipynb.

4.2 performance

Parallel Easy to see that for each m_i in cut, finding the $\partial\Omega_{m_i}$ with $minE$ is independent, using a go version code to do a multithreads work, detail see the "path-find.md"

5 Code

https://github.com/fangwater/cs276lab/tree/main/assignment2