Poisson Image Editing 的实现

I. 简介

泊松图像编辑是一种利用泊松方程解决图像编辑问题的方法，由Microsoft Research UK的Patrick Perez，Michel Gangnet,and Andrew Blake在论文“Poisson Image Editing”中首次提出。

II. 算法说明

A. 哈密顿算子、拉普拉斯算子、梯度与散度

哈密顿算子

$$\nabla = \frac{\partial}{\partial x}\vec{i} + \frac{\partial}{\partial y}\vec{j} + \frac{\partial}{\partial z}\vec{k}$$

拉普拉斯算子

$$\Delta = \nabla \cdot \nabla = \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$$

（拉普拉斯算子可以定义为梯度的散度）

梯度

$$grad(A) = \nabla A = \frac{\partial A}{\partial x}\vec{i} + \frac{\partial A}{\partial y}\vec{j} + \frac{\partial A}{\partial z}\vec{k}$$

散度

$$div(\vec{A}) = \nabla \cdot A = \frac{\partial A_x}{\partial x} + \frac{\partial A_y}{\partial y} + \frac{\partial A_z}{\partial z}$$

B. 图像中的运算

在图像中，通常使用查分来近似导数，最简单的一种梯度表示方式如下

$$Grad_x(x,y) = P(x+1,y) - P(x,y)$$

$$Grad_y(x,y) = P(x,y+1) - P(x,y)$$

因此我们对梯度求散度的话，有

$$Div(x,y) = Grad_x(x+1,y) - Grad_x(x,y) + Grad_y(x,y+1) - Grad_y(x,y)$$

拉普拉斯算子可以定义为梯度的散度,所以拉普拉斯算子在图像中的运算可以表示为如下形式

$$laplacian(x,y) = P(x+1,y) - P(x,y) - P(x,y) + P(x-1,y) + P(x,y+1) - P(x,y) - P(x,y) + P(x,y-1)$$

也即

$$laplacian(x,y) = P(x+1,y) + P(x-1,y) + P(x,y+1) + P(x,y-1) - 4P(x,y)$$

写成矩阵有如下形式，其中 P 代表像素点(x,y)的八邻域，这里并不是矩阵乘法运算，而是对应元素相乘然后累加。

$$laplacian(x,y) = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} P$$

C. 泊松图像编辑的基本思想

将背景图片记作 S，将前景图片记作 G，在 S 中插入 G 的对应区域Ω记作 f*，Ω的边界是∂Ω（不含有Ω），G 的梯度图记作ν。

在将 G 嵌入到 S 的Ω区域时，Patrick Perez 等人提出在保证边界不变（即∂Ω的像素点是属于 S 的）的前提下，使得合成后图片Ω区域（记作 f）的梯度值与原图的梯度值的差值最小。即

$$min \iint_{\Omega} |\nabla f - \nu|^2,\ f|_{\partial\Omega} = f^*|_{\partial\Omega} --①$$

（论文公式 3）

这个方程的解是满足狄里克雷边界条件的泊松方程，即

$$\Delta f = div v,\ \ f|_{\partial\Omega} = f^*|_{\partial\Omega} --②$$

（论文公式 4）

由于图像是二维离散的数据值，所以公式可以变化为

$$f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y) = div v --③$$

由此就可以构造出一个形如$Ax = b$的方程组。

当我们把 x 也即所有$f(x,y)$都求出来之后，将 f 覆盖到背景图片 S 上，就可以得出合成后的图像。

注意在（x,y）处于 f 边缘上时，上式会计算不属于 f 的像素点，此时该像素点取∂Ω上的像素点。

这里举个例子来说明计算过程。

假设有一个大小是 4*4 的 16 个像素点图片 S

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

我们要插入一个 2*2 的四个像素点的图像 G，插入位置为6、7、10、11四个像素点，那么根据公式③有下式成立，其中 f（x）代表的是合成后图片 f 在 x 处的像素点

$$\begin{cases} f(2)+f(5)+f(7)+f(10)-4*f(6)=div(G(6)) \\ f(3)+f(6)+f(8)+f(11)-4*f(7)=div(G(7)) \\ f(6)+f(9)+f(11)+f(14)-4*f(10)=div(G(10)) \\ f(7)+f(10)+f(12)+f(15)-4*f(11)=div(G(11)) \end{cases}$$

我们要求的是$f(6),f(7),f(10),f(11)$，对于边界如$f(2)$，我们取$f(2)=S(2)$（边界不变的条件）,这样我们就可以利用矩阵求解这个方程。

D. *程序算法流程*

1. 求解前景图像与背景图像的梯度，同时定义感兴趣的区域（ROI），该区域的大小就是前景图像的大小，该区域是在背景图像上的。
2. 将前景图像的梯度图覆盖到背景图像梯度图的 ROI 上，同时对覆盖后的梯度图求散度。

   前两部分主要是利用 OpenCV 中的 filter2D()函数来实现的，这部分相对比较简单，所以就直接调用函数实现。
3. 根据公式③构造$Ax=b$中的 A 与 b

   A 是一个大型稀疏矩阵，每一行中的非零元素不超过 5 个，这里构建了一个比较简陋的稀疏矩阵类 Sparse_matrix 来保存 A。

   （如果 ROI 的范围是 30*30，则 x 的维数是 900*1，那么 A 的维数就是 900*900）
4. 利用 Jacobi Method 求解 x

   在求解的过程中，注意迭代次数。在本次实现中暴力求解导致返程求解效率（应当是平方级别的）极其低下，迭代 100 次的解大约需要 30 分钟，迭代 1000 次，可以先去睡一觉，然后再回看效果。但同样的迭代次数少，求解出来的图像效果不好，曾试过只迭代一次，最后输出的图像就像是对 ROI 的区域进行边缘检测。
5. 显示结果

III. 运行结果

A. *Jacobi Method 迭代 100 次的结果*

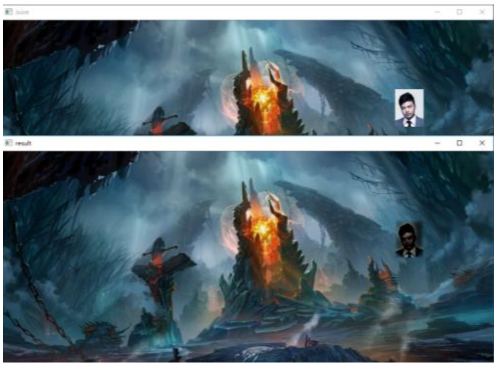
见图一

B. *Jacobi Method 迭代 1000 次的结果*

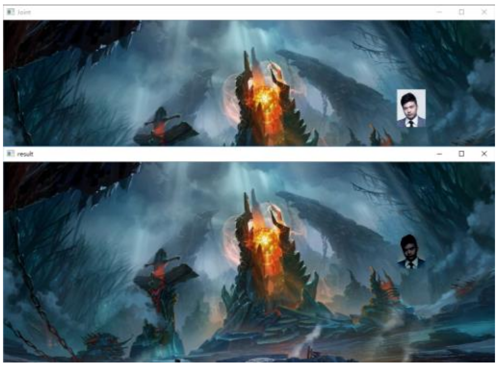
见图二

C. *对比*

从两图中可以看出，在迭代 100 次之后还可以看到比较模糊的边界，但迭代 1000 次之后几乎无法看到明显的边界。此外在求解出来的前景图片中我们可以看出，前景图片的颜色发生了变化，这是因为该算法是尽可能的保留梯度信息，这应当是一种折中。



图一，迭代 100 次



图二，迭代 1000 次

参考文献

[1] Poisson Image Editing,2003_siggraph_perez

[2] https://blog.csdn.net/hjimce/article/details/45716603

[3] https://blog.csdn.net/u011534057/article/details/68922197

[4] http://eric-yuan.me/poisson-blending/

[5] https://blog.csdn.net/majinlei121/article/details/46831769