Digital Image Processing 1^{st} Report

生 51 陈旭鹏 2014012882

April 18 2019

目录

1 Image Fusion **2**
1.1 代码说明 2
1.2 算法原理 2
1.3 实验结果 2

2 Image Morphing **4**
2.1 代码说明 4
2.2 算法原理 4
2.3 实验结果 4
2.3.1 人脸关键点检测和 Delaunay 三角剖分 4
2.3.2 morphing and blending 4

3 参考文献 **6**

1 Image Fusion

1.1 代码说明

本部分采取 python 实现，使用 opencv 的 IO 功能以及 numpy，scipy 的一些矩阵变换，稀疏矩阵及线性方程组求解模块。实现效果可以通过运行 image fusion 文件夹下的 poisson_image_editing.ipynb 文件获得。

1.2 算法原理

原始的泊松方程定义为：

$$\Delta\varphi = f$$

为了将两个图像融合，实际需要求解的方程为：

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

即为了把图像 source 融合在图像 target 上，f 表示融合的图像 result，f* 表示 target image，v 表示 source image 的梯度，∇f 表示 f 的一阶梯度，Ω 表示要融合的区域，$\partial\Omega$ 代表融合区域的边缘部分。即在 target image 边缘不变的情况下，求融合部分的 result image，使得 result image 在融合部分的梯度与 source image 在融合部分的梯度最为接近。

对于离散的图像，源图像的梯度可表示为：

$$\left|\Delta B_{(x,y)}\right| = 4B(x,y) - B(x-1,y) - B(x+1,y) - B(x,y-1) - B(x,y+1)$$

上述公式可以转化为一个差分方程：

$$H(x,y) - \sum_{(dx,dy)+(x,y)\in\Omega} H(x+dx,y+dy) - \sum_{((dx,d))+(xy)\in\partial\Omega} A(x+dx,y+dy) = \sum_{(dx,dy)+(x,y)\in(\Omega\cup\partial\Omega)} (B(x+dx,y+dy) - B(x,y)) \tag{1}$$

与 H 有关的都是未知数，求解出这些像素值即可，把求二阶梯度的公式左边看成一个 $n\cdot n$ 矩阵 A，把未知区域的 H 看做是 $\vec{x}$，把已知的 source image B 的二阶梯度即式子右边看做是一个 $n\cdot 1$ 列矩阵 $\vec{b}$，那么问题的本质就是求解

$$A\vec{x} = \vec{b}$$

A 的每一行至多只有 5 个非零元素，并且对角线上的元素均为 4，是一个巨大的稀疏矩阵。使用 Jacobi method 可以解出 $\vec{x} = A^{-1}\vec{b}$，即将矩阵 A 分解为一个对角矩阵 D 和上、下三角矩阵 L 和 U，然后每一步迭代的步骤是

$$\mathbf{x}^{(k)} = D^{-1}(L+U)\mathbf{x}^{(k-1)} + D^{-1}\mathbf{b}$$

可以设置 $X^{(0)} = 0$，然后迭代足够多次即可求解出 $\vec{x}$

1.3 实验结果

第一组图的原图 source，mask，target 如图 1所示

融合后的图效果如图 2所示

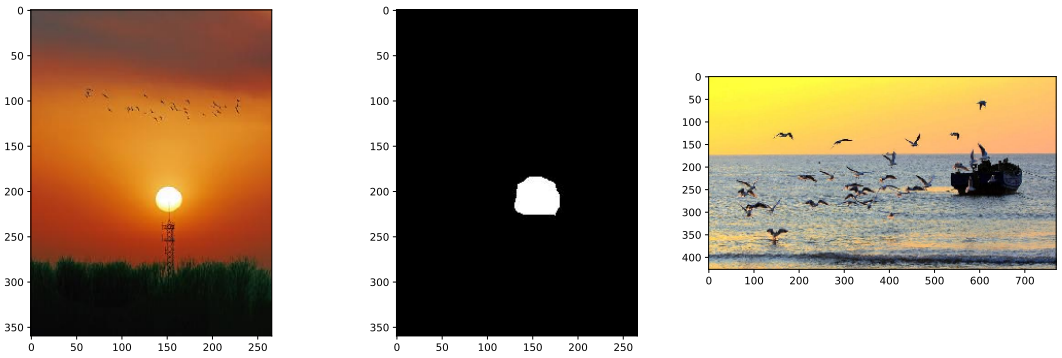

图 1: Figure 1

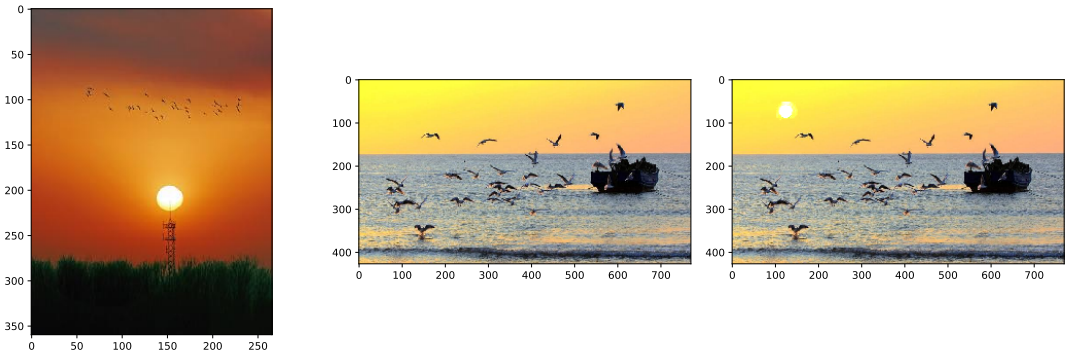

图 2: Figure 2

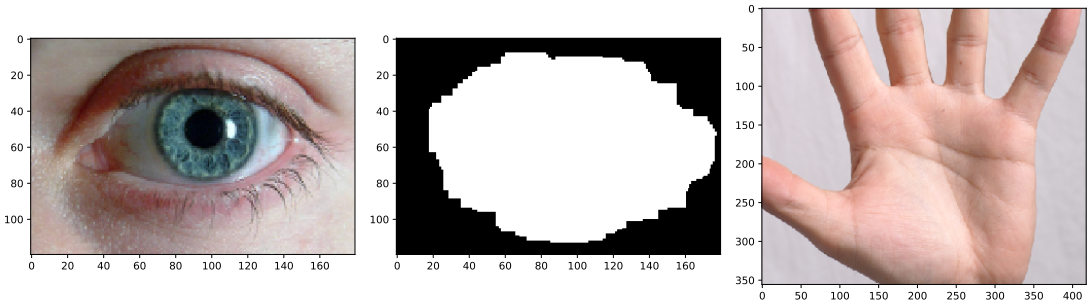

图 3: Figure 3

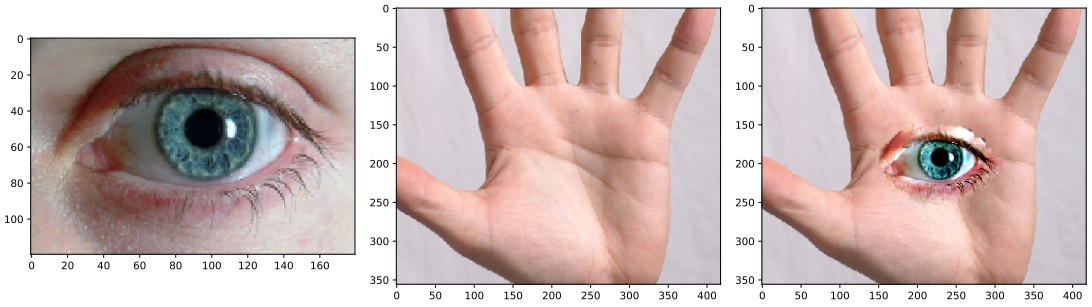

图 4: Figure 4

第二组图的原图 source，mask，target 如图 3所示

融合后的图效果如图 4所示

虽然使用 python,但是由于使用 scipy 的 sparse 和 scipy.sparse.linalg.spsolve 函数求解，运行时间并不是很慢，分别需要 10.5 和 5.26s 完成融合。

2 Image Morphing

2.1 代码说明

本部分采取 python 实现,使用 opencv 的 IO 功能以及 numpy，scipy 的一些函数。实现效果可以通过运行 image morphing 文件夹下的 image_morphing.ipynb 文件获得。对于人脸的关键点检测，我调用了 Face++ facepp v3 使用深度学习进行人脸的关键点检测。

2.2 算法原理

主要算法包括关键点检测，三角剖分，三角形的仿射变换和融合。

- 获取人脸的关键点即手工标注狮子面部的关键点。(由于 face++ 不支持狮子的面部关键点标注)
- 对于关键点进行 Delaunay 三角剖分
- 对每个三角区域分别进行仿射变换并进行融合

根据仿射变换的表达:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Source_{1x} & Source_{2x} & Source_{3x} \\ Source_{1y} & Source_{2y} & Source_{3y} \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} Target_{1x} & Target_{2x} & Target_{3x} \\ Target_{1y} & Target_{2y} & Target_{3y} \\ 1 & 1 & 1 \end{bmatrix}$$

其中仿射变换矩阵为:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

基本的矩阵求逆就可以解除 affine transformation 矩阵。

2.3 实验结果

2.3.1 人脸关键点检测和 Delaunay 三角剖分

三角剖分之后的可视化效果分别如图 5，6，7，8 所示

2.3.2 morphing and blending

使用仿射变换的方法对每组三角形进行仿射变换，再按照一定的 α 确定混合程度即可获得两张脸进行融合的效果。融合后的效果图如 9，10 所示

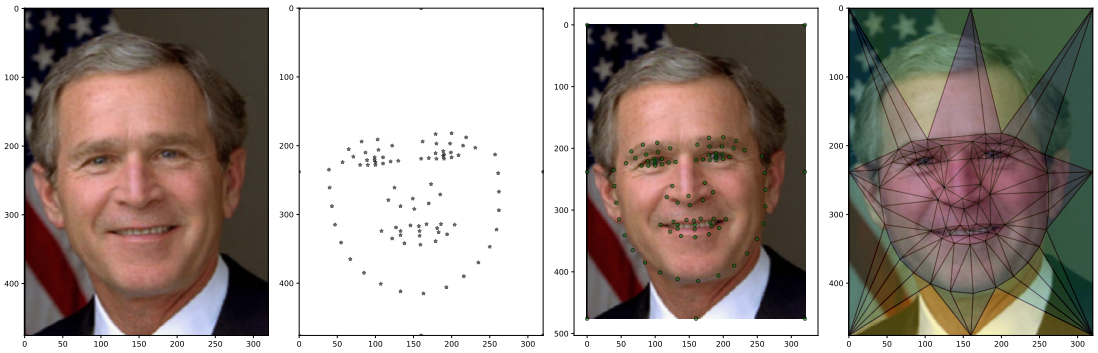

图 5: Figure 1

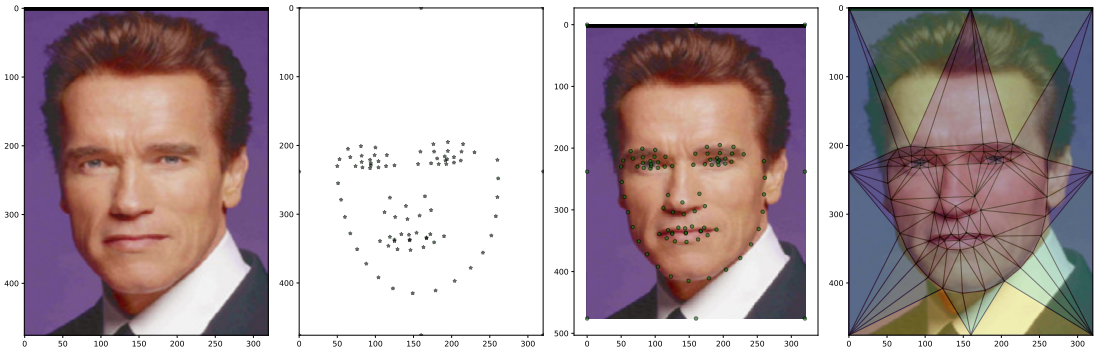

图 6: Figure 2

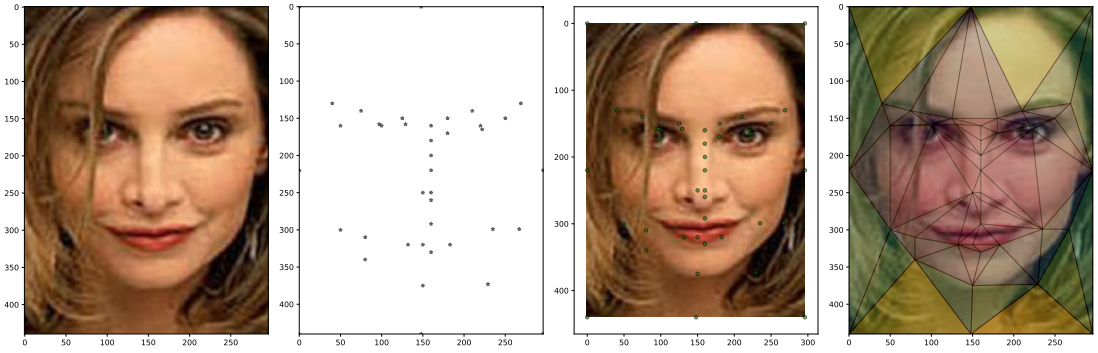

图 7: Figure 3

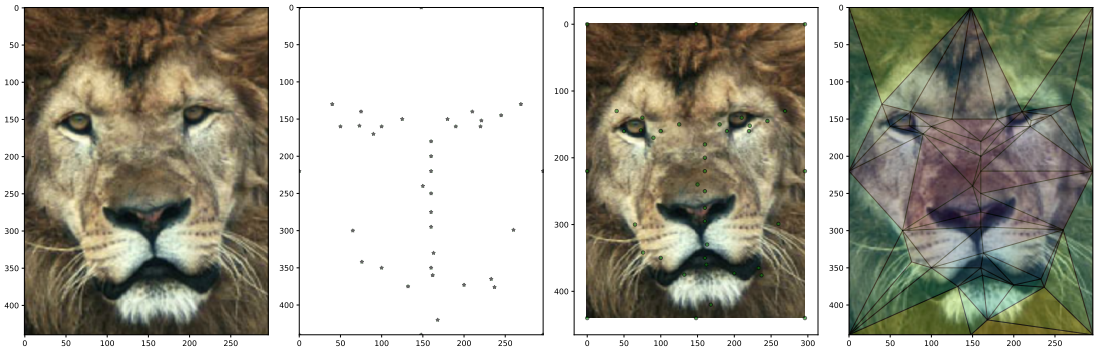

图 8: Figure 4

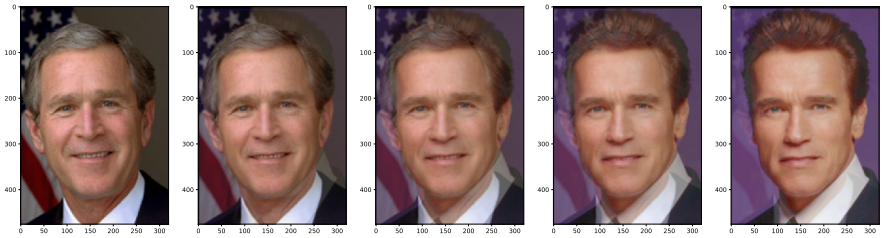

图 9: Figure 5

图 10: Figure 6

值得注意的是，由于人脸和狮子脸的 pattern 具有较大的差异，直接的融合可能导致一些问题，因此在做关键点标注的时候，我尽可能地找到两张脸位置比较近似的点作为成对的关键点进行标注。

3 参考文献

[1] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. ACM Transactions on graphics (TOG), 22(3):313--318, 2003.

[2] Jacobi method. https://en.wikipedia.org/wiki/Jacobi_method.

[3] Jacobi method. http://mathworld.wolfram.com/JacobiMethod.html

[4] Satya Mallick. https://www.learnopencv.com/face-morph-using-opencv-cpp-python/

[5] Christopher J. Tralie. Poisson image editing. http://www.ctralie.com/Teaching/PoissonImageEditing/

[6] Discrete Poisson Equation https://en.wikipedia.org/wiki/Discrete_Poisson_equation