

# Image Fusion

计 64 翁家翌 2016011446

2018.6

## Contents

1	运行说明	2
2	实现方式	2
3	算法细节	2
3.1	MVC . . . . .	2
3.2	Poisson . . . . .	3
3.3	实验结果 . . . . .	3
3.3.1	Test1 . . . . .	3
3.3.2	Test2 . . . . .	4
3.3.3	Test3 . . . . .	5

# 1 运行说明

首先安装依赖的第三方库：

```
1 sudo apt install cmake libcgall-dev libcgall-qt5-dev
2 sudo pip2 install opencv-python
```

然后编译 C++ 文件：

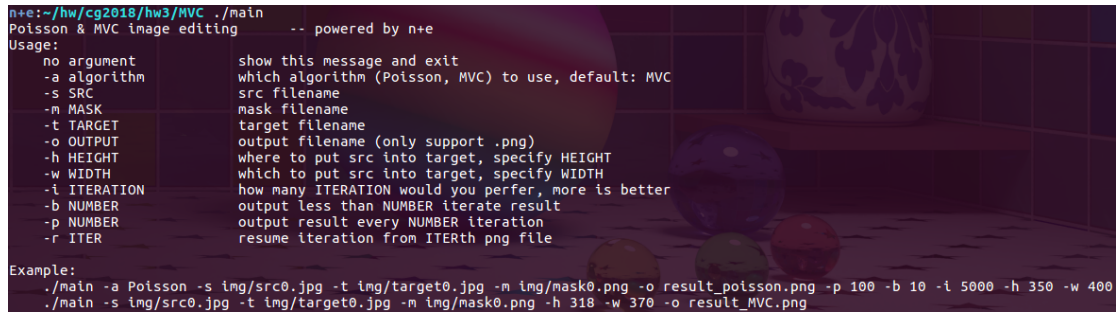
```
1 cmake .
2 make
```

在当前目录下生成可执行文件 main。

使用命令

```
1 ./main <args>
```

即可运行或查看帮助，如图1所示：



```
n+e:~/hw/cg2018/hw3/MVC ./main
Poisson & MVC image editing -- powered by n+e
Usage:
no argument          show this message and exit
-a algorithm          which algorithm (Poisson, MVC) to use, default: MVC
-s SRC                src filename
-m MASK               mask filename
-t TARGET              target filename
-o OUTPUT              output filename (only support .png)
-h HEIGHT              where to put src into target, specify HEIGHT
-w WIDTH              which to put src into target, specify WIDTH
-i ITERATION           how many ITERATION would you prefer, more is better
-b NUMBER              output less than NUMBER iterate result
-p NUMBER              output result every NUMBER iteration
-r ITER               resume iteration from ITERth png file

Example:
./main -a Poisson -s img/src0.jpg -t img/target0.jpg -m img/mask0.png -o result_poisson.png -p 100 -b 10 -i 5000 -h 350 -w 400
./main -s img/src0.jpg -t img/target0.jpg -m img/mask0.png -h 318 -w 370 -o result_MVC.png
```

图 1: 查看帮助

## 2 实现方式

考虑到实现效率，我采用 C++ 编写代码，其中图片的读入和输出采用 [github](https://github.com/nothings/stb) 上的开源仓库"stb"<sup>1</sup>实现，图片边缘的提取采用 [OpenCV](https://opencv.org/)<sup>2</sup>。

## 3 算法细节

### 3.1 MVC

MVC[3, 2] 算法可分为如下步骤：

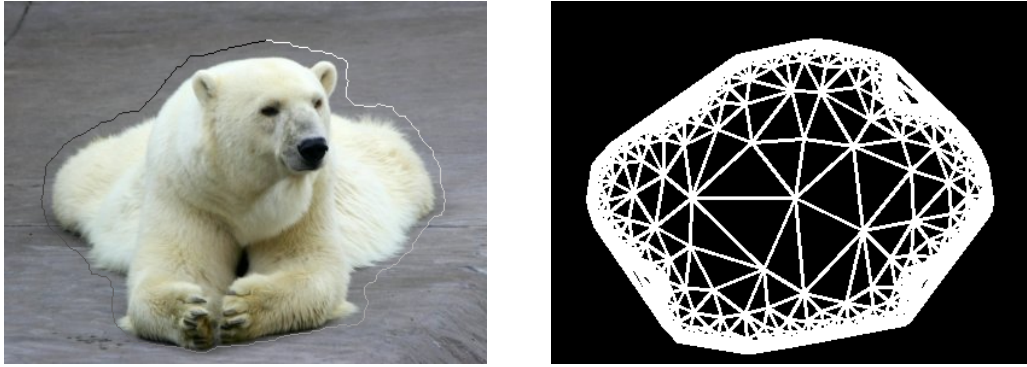
1. 找出所有的边缘像素点的位置；
2. 使用 CGAL 将这些点进行带约束的三角剖分；
3. 对于三角剖分结果中的非边缘像素点，依据论文中给出的权重计算方式算出这个点所要垫高的权值；

<sup>1</sup><https://github.com/nothings/stb>

<sup>2</sup><https://opencv.org/>

4. 对于每个三角面片，已经求出了三个顶点的垫高的权值，并且由于假设函数在该范围内平滑，因此直接用一个平面去拟合三角形内的所有像素点的权值大小即可。

图2展示了 MVC 算法的中间结果，图2(a)显示的是从 mask 中找到边缘之后的可视化效果，图2(b)显示的是从这些边界上的点生成三角剖分的网格之后的效果。请注意，原图并不是一个凸多边形，因此在三角剖分之后要手动去掉那些不在轮廓内的三角形，具体可以采用射线法判断某个点是否是多边形内的算法即可。



(a) 边界分割

(b) 三角剖分结果

图 2: MVC 可视化中间步骤

## 3.2 Poisson

此外我还实现了泊松图像融合算法 [4, 5] 进行对比，其梯度计算方式为

$$\nabla(x, y) = 4I(x, y) - I(x - 1, y) - I(x, y - 1) - I(x + 1, y) - I(x, y + 1)$$

将原图求完梯度之后，将该梯度匹配到目标图上的某一区域，本质上是一个解线性方程组的问题。形式化地，设有  $N$  个像素点需要匹配到目标图片中，则需要求解线性方程组

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

其中  $\vec{x}$  代表融合后的图片中像素点的值，矩阵  $A$  的大小  $\sim N \times N$ ，列向量  $\vec{x}$  和  $\vec{b}$  的大小  $\sim N$ ，并且  $A$  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。

$A$  是一个巨大的稀疏矩阵。考虑到矩阵求逆的复杂度为  $O(N^3)$  太高，并且某些情况下连  $A$  都无法直接以矩阵形式存储，因此无法直接从公式

$$\vec{x} = A^{-1}\vec{b}$$

求得  $\vec{x}$ 。此处采用 **Jacobi Method** 迭代求解出  $\vec{x}$  的值，详见 [1]。

## 3.3 实验结果

### 3.3.1 Test1

使用命令

```
1 time ./main -s img/src3.jpg -t img/target1.jpg -m img/mask3.jpg -o
   result_MVC.png -h -135 -w -30 -a MVC
```

```
2 time ./main -s img/src3.jpg -t img/target1.jpg -m img/mask3.jpg -o
   result_Poisson.png -i 5000 -h 50 -w 100 -a Poisson
```

可得到如下结果

```
1 Done with 262 vertices and 466 triangles.
2 real    0m0.243s
3 user    0m0.287s
4 sys     0m0.233s
5 -----
6 iter 5001  err 0.001767 0.001913 0.003486
7 real    0m0.361s
8 user    0m0.345s
9 sys     0m0.016s
```

可以看到，MVC 总共只用了 262 个三角顶点，耗时 0.243s；Poisson 由于使用迭代方法求解矩阵，迭代次数越多解的越精确，5000 轮之后误差几乎为 0，并且运行速度为 0.36s，二者均十分快。合成效果如图3所示。由于该样本太容易合成，二者在效果上看不出有什么明显差别。



图 3: 第一组数据合成效果

### 3.3.2 Test2

使用命令

```
1 time ./main -s img/src4.png -t img/target2.png -m img/mask4.png -o
   result_MVC.png -h 142 -w 132 -a MVC
2 time ./main -s img/src4.png -t img/target2.png -m img/mask4.png -o
   result_Poisson.png -i 10000 -h 150 -w 150 -a Poisson
```

可得到如下结果

```
1 Done with 836 vertices and 1580 triangles.
2 real    0m0.233s
3 user    0m0.228s
```

```

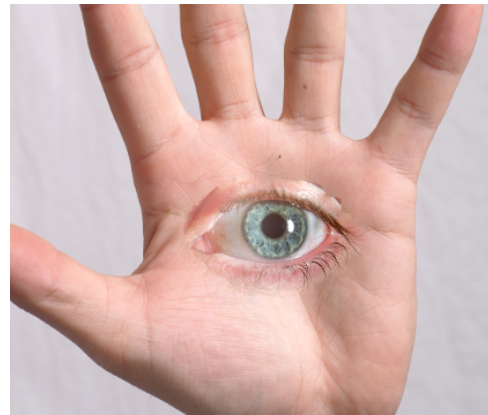
4 sys      0m0.305s
5 -----
6 iter 10001  err 82.519269 49.665685 58.341544
7 real      0m5.356s
8 user      0m5.351s
9 sys      0m0.004s

```

可以看到 MVC 总共只用了 836 个三角顶点，耗时 0.228s；Poisson 在 10000 轮之后总误差不到 100，平均每个像素点误差不到 0.01，并且运行速度为 5s 左右。合成效果如图4所示，对比可以发现 MVC 的边缘比 Poisson 更加平滑。



(a) MVC



(b) Poisson

图 4: 第二组数据合成效果

### 3.3.3 Test3

使用命令

```

1 time ./main -s img/src0.jpg -t img/target0.jpg -m img/mask0.png -h
   318 -w 370 -o result_MVC.png
2 time ./main -a Poisson -s img/src0.jpg -t img/target0.jpg -m img/
   mask0.png -o result_poisson.png -i 10000 -h 350 -w 400

```

可得到如下结果

```

1 Done with 1702 vertices and 3346 triangles.
2 real      0m0.420s
3 user      0m0.458s
4 sys      0m0.253s
5 -----
6 iter 10001  err 2238.275301 1683.477450 1885.838338
7 real     0m43.901s
8 user     0m43.816s
9 sys      0m0.040s

```

这是 MVC 论文中的图，可以看到 MVC 总共只用了 1702 个三角顶点，耗时 0.458s；Poisson 在 10000 轮之后总误差大约 2000，平均每个像素点误差不到 0.03，但是运行速度达到了 43s 左右。合成效果如图5所示。从效果上看而言还是 MVC 更胜一筹。



(a) MVC



(b) Poisson

图 5: 第三组数据合成效果

## References

- [1] Jacobi method. [https://en.wikipedia.org/wiki/Jacobi\\_method](https://en.wikipedia.org/wiki/Jacobi_method). Accessed: 2018-04-12.
- [2] Kewei Chen and Chenen Wu. Mvc. <https://www.csie.ntu.edu.tw/~r00944002/CPHW2/MVC.htm>. Accessed: 2018-06-12.
- [3] Zeev Farbman, Gil Hoffer, Yaron Lipman, Daniel Cohen-Or, and Dani Lischinski. Coordinates for instant image cloning. *ACM Transactions on Graphics (TOG)*, 28(3):67, 2009.
- [4] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Transactions on graphics (TOG)*, 22(3):313--318, 2003.
- [5] Christopher J. Tralie. Poisson image editing. <http://www.ctralie.com/Teaching/PoissonImageEditing/>. Accessed: 2018-04-12.