

# 图像抠图报告

---

计71 钟国鑫 2017010306

图像抠图实现的论文为 `A Closed Form Solution to Natural Image Matting` (CVPR 2006)

代码见 `image_matting` 目录, 或者github仓库

代码由rust语言编写, 需要nightly支持, 其他依赖库见 `Cargo.toml`, 如何运行可见github仓库的README

## 实现内容

---

- ☑ 单张图片抠图
- ☑ gif连续帧抠图
- ☑ 独立编写线性运算库, 实现预条件子优化的共轭梯度算法用于解稀疏线性方程组, 并进行多线程并行加速

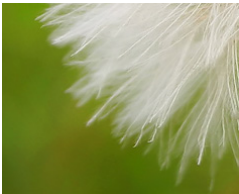
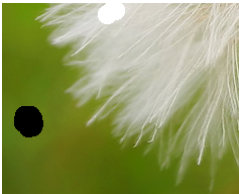

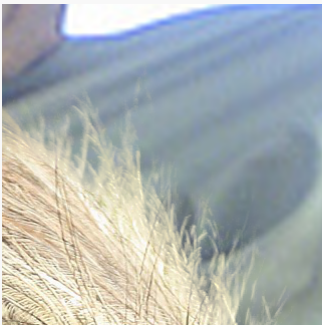

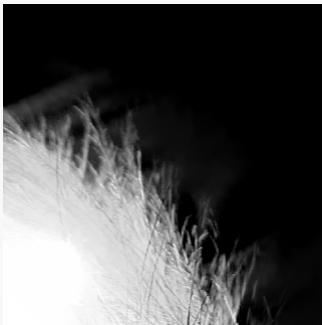
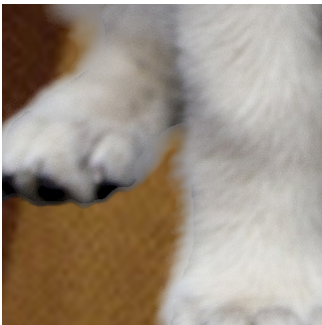

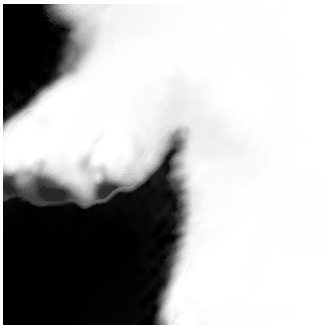
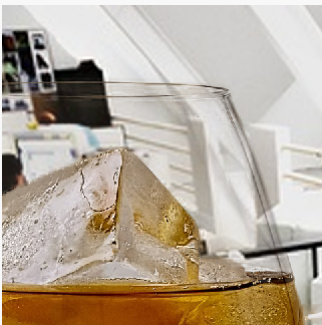

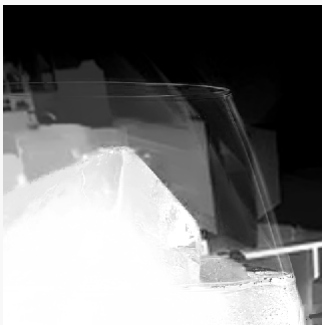
## 实现结果

---

所有结果在 `image_matting/img` 中

注: 由于pdf不支持显示gif, 因此gif的结果没有在这里给出, 具体结果见

`image_matting/img/5_gif/`, 而 `image_matting/img/5` 为此gif每帧单独抠图的结果, 由于图片太多不在文档中呈现

原图	约束图	抠图结果
		
		
		
		

- 最后一张玻璃杯的抠图可以看到结果清晰的显示出了玻璃的边界

## 论文实现

这篇论文的精妙之处在于通过一个线性模型的假设，从而能够通过解一个稀疏线性方程组来求出前景的alpha值(不透明度)，实现抠图

整篇论文最重要的公式就是

$$\sum_{k|(i,j) \in w_k} (\delta_{ij} - \frac{1}{|w_k|} (1 + (I_i - \mu_k) (\sum_k + \frac{\epsilon}{|w_k|} I_3)^{-1} (I_j - \mu_k)))$$

该公式给出了稀疏矩阵 $L$ 的第 $(i, j)$ 项的值，其中 $\sum_{k|(i,j) \in w_k}$ 含义是，遍历所有含有第 $(i, j)$ 项的 $3 \times 3$ 窗口，计算他们的右边结果之和，因此该篇论文实现的关键点主要在如何计算出这个值

实现之前，我阅读了这篇论文的公开代码([python版本](#))，在python代码版本里，作者使用了非常多的numpy库的特性来十分简单地实现了窗口处理的效果，然而这些在rust里是不存在的...

因此在实际实现中，我通过如下两个结构来实现类似的窗口处理

```

1 //见image_matting/src/method/mod.rs
2 struct WindowRef<'a> {
3     mat_ref:      &'a RgbMatrix,
4     start_idx:    (usize, usize),
5     cov_mat:      M33,
6     mean_v:       Rgb,
7     inner_mat:     Matrix,
8     global_idx:    Vec<usize>,
9 }
10 struct WinRefMatrix<'a> {
11     elem: Vec<WindowRef<'a>>,
12     nrow: usize,
13     ncol: usize,
14 }

```

运行逻辑如下

1. 输入图片，将其转化为一个 `RgbMatrix`，其中行列为图片行列，每个元素为一个 `[f64;3]`
2. 根据给定窗口大小，构造 `WinRefMatrix`，其中的每个窗口都指向之前的 `RgbMatrix`
3. 对 `WinRefMatrix` 进行批量处理，包括求解所有的小窗口分布、均值向量等
4. 由 `WinRefMatrix` 得到CSR形式的稀疏线性矩阵，最终求解方程组

当正确实现这两个结构的相关操作后，整个流程就比较顺畅了，但是我在最后一步，也就是解稀疏线性方程组的时候卡住了

## 线性库

(这一部分才是图像抠图实现最耗时间的部分)

rust目前的线性操作库很少，其中还支持稀疏线性方程组求解的就大致只有 `sprs` 和 `narray` 等，然而这两个库对于稀疏线性矩阵的求解的都是基于LU分解，其效率都非常低，对于一张  $235 \times 189$  的图像所得到的矩阵进行处理都需要10min左右

因此最终我选择自己编写线性库来支持所需要的稀疏线性方程组的求解

为了使求解速度更快，我选择的求解算法是预条件子优化的共轭梯度算法

共轭梯度算法是目前求解稀疏矩阵较好的方法之一，而经过预条件子优化可以使得迭代次数更少，更快收敛

同时我使用 `Rayon` 库(类似于c++的openmp)对算法进行了多线程的优化

具体实现可见 `image_matting/src/linear.rs`



- 目前多线程版本的预条件子优化的共轭梯度算法在处理  $235 \times 189$  的图像(即样例的第一个)所需时间大约为1min


注：`image_matting/src/linear.rs`与`linear/`实际是一样的，但由于出现了rust的库间宏调用的相关问题，导致一开始的耦合难以优雅的解决，因此`image_matting`使用的线性库是内部的`linear.rs`，而`linear/`库则由其他两个项目使用

## 效果

选择迭代类的算法，而不是LU分解，其原因之一就是收敛精度等可调，因为这种图像处理不需要太高的精度(人眼无法分辨即可)，因此这样也能为时间调整留足余地

实际收敛精度、初始值(无初值即全0，有初值即约束部分为1，其余为0)、预条件子选择确实对最终处理得到的图像有影响(迭代次数，效果)，下面给出一些例子

收敛精度	有无初值	预条件子	迭代次数	效果
1e-5	无	绝对值	1013	
1e-5	无	Jacobi	1315	
1e-5	有	无	3397	
1e-6	无	欧几里得距离	1743	
1e-6	无	Jacobi	1879	

收敛精度	有无初值	预条件子	迭代次数	效果
1e-6	无	绝对值	1900	
1e-6	无	无	2880	

- 从效果图中可以看出，不同的选择确实对图片会有影响，比如右下角的颜色和中上部高光部分的清晰度有着人眼可分辨的区别
- 除此之外，也可以看出预条件子和收敛精度的选择会很大程度上影响迭代次数，但是初值的设置却导致了收敛次数增加，这一点是比较奇怪的一点(具体原因应该是初值设置有问题)

## gif处理

视频方面我使用gif来进行替代，简化处理的同时也遇到了gif的一些问题

- gif的一帧如下



gif为了压缩大小，内部采用了一个256色的调色板，其图片上的所有颜色都是基于这个调色板合成的

当我使用ps对其每一帧画约束，然后再重组成一个新的gif后，一些没有被约束的像素颜色也会被改变(因为调色板颜色固定的缘故，现在多了一些颜色，黑白灰之类)，这就导致算法在判断约束的时候出现了问题，无法正确判断哪些像素被约束为前景，哪些被约束为了背景

除此之外，所选的gif的背景颜色过于丰富，以及与前景相邻的部分颜色又区分度不大，这些因素加在一起导致了gif的效果不是很好

- 目前能够想到的一个对于约束的解决办法就是，采用图像融合类似的做法，使用mask而不是直接在上面画图，做一个mask的gif图，然后以此来抠图处理

## 参考资料

---

1. 论文python源码<https://github.com/MarcoForte/closed-form-matting>
2. 预条件子的优化, [https://image.hanspub.org/Html/28-2620432\\_21534.htm](https://image.hanspub.org/Html/28-2620432_21534.htm)
3. 共轭梯度算法, <https://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf>