# 数据结构大作业实验报告

## 图片网络爬虫设计与图像处理

## 欧阳鸿荣

## 计算机科学与技术系

## 161220096

**一、问题分析**

本次大实验可以分为两个部分：

1.设计、实现图片网络爬虫，从网站上批量下载图片；

2.对图片进行图像模糊和边缘化检测的处理。

**（1）网络爬虫方面**

在查阅资料比较各种语言后，考虑到python有比较丰富的库，且打算新学z门语言，决定使用python进行处理。实验要求爬取不重复网页的数量不少于 1000 个，图片数量不少于 10000 张，对于待访问的网址，使用队列存储，按深度优先访问。

因此，实验的基本思路是，以一个网页为起始网页，先获得其超链接指向的网页，存储于队列中，然后依次访问网页和其子网页，下载网页上的图片。

实验中可能出现环路问题，因此建立一个BF布隆过滤器类，通过选取恰当额哈希函数，对已经访问过的网址进行过滤。

由于所爬取的网址和图片数目较多，因此查阅资料后采用了多线程。

**（2）图像处理方面**

由于图片种类繁多，则选取基于C++的开源库Opencv进行图片的解析。而图像处理方面，也选择使用C++，基于均值滤波器和索贝尔边缘检测的原理来处理。

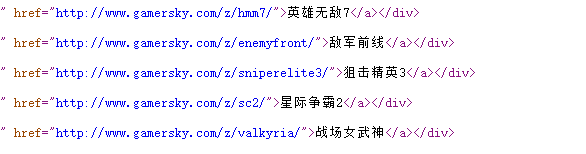
**二、总体设计框架**

**（1）网络爬虫方面**

**主要将借助re，requests，urllib三个库实现网络网页的爬取。**

**（a）库re主要用于实现正则表达式**

利用浏览器查看网页源代码，可以发现几乎所有的跳转链接的格式都是 href = ”…”，然后图片几乎都是 src=”… +.后缀名”：



故将爬取跳转链接的正则表达式定义为：

href="(http.[^ ^\t^\r^\n^]\*?)"



将爬取图片的正则表达式定义为：

src="(http.[^ ^\t^\r^\n^]\*?\.)(jpg|png|jpeg|gif|bmp)"

然后通过re.findall(模式串,待匹配串)方法即可进行模式串的匹配。

**（b）库requests主要用来查看网页内容**

这样一来便可利用正则表达式匹配到其中的跳转链接和图片下载地址。

在翻阅网上的资料是时，发现，我们需要设置一个浏览器的标识以伪装成浏览器，防止被服务器查水表。请求头部定义如下：

hd = {'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.63 Safari/537.36 Qiyu/2.1.0.0'}

**（c）库urllib用来将图片下载到指定的位置**

通过urllib.request.urlretrieve(下载地址,存储路径)即可。

**（d）对于网页去重问题，实现布隆过滤器**。

将采取选取一个哈希函数，利用不同Hash种子来模拟多Hash的方式。设需要爬取的不同网页数为n，布隆过滤器的bit数为m，误判率为p，哈希函数的个数为k，根据公式推导有如下结论：

误判率最小时有：

因此在Bloom filter的初始化中，需要给定最大爬取的网页数以及最大误判率，通过以上公式计算最优的m和k，以实现一个布隆过滤器。对于该k个哈希函数，编号为0,1,2,…,k-1，以各自的编号作为哈希函数的随机种子。

对于待访问网址的存储，选用队列作为存储结构，实现深度优先遍历。同时为了提高效率，需要使用多线程，故将先爬够足够的网址，并分别创建相应的待执行线程，最后再一起执行。

在多线程的设计中考虑到有可能出现某个网址获取很慢（有可能是爬取到一些不方便访问的网站，比如以.exe，.pdf，.apk结尾的下载链接的网站，或者是已经404 not found（不存在）的网站，），甚至是一直不返回数据而导致线程假死的情况，因此我们在代码中将为每次获取信息报设置一个timeout参数，当请求相应超过timeout秒时将放弃请求，同时每一个线程将有一个生命周期lifetime的参数，若访问失败，则lifetime--，当lifetime减到0的时候放弃该线程的执行，也即放弃该线程负责的网址中图片的爬取。

**（2）图像处理方面**

利用opencv的cvLoadImage读取图片，cvSaveImage保存图片，同时使用cvGet2D方法可获取图片某个位置处的RGB三个像素值。

**（a）图像模糊方法的实现**

可以在读取图像之后，用一个RGB\_Array的全局数组记录图像的所有位置处的RGB像素，然后遍历图像上的每个点，分别求出其周围n\*n个位置的RGB三个值的平均值，并将其赋给RGB\_Array的对应位置的记录值，全部处理完成后通过cvSet2D设置修改后的像素，最后再用cvSavaImage进行保存即可。

**（b）图像边缘检测方法的实现**

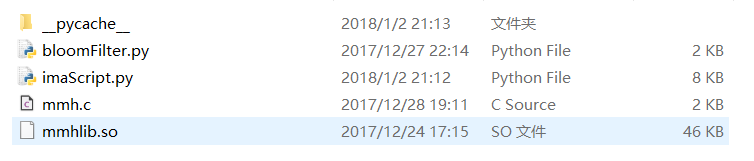
首先利用cvLoadImage方法读取图片，同时在传入参数的时候设置iscolor=0读取图片，同样可以通过cvGet2D读取到灰度值，并将之存储到一个Gray\_Array的全局数组中，通过遍历所有的像素点，利用sobel算子分别估算每一个位置处的梯度值，同时算出整个图像的灰度平均值，根据公式sqrt(scale\*mean)计算阈值，然后将所有梯度大于等于该阈值的点的灰度值置为255，小于该阈值的点的灰度值置为0。同样通过cvSet2D设置修改后的像素，最后利用cvSavaImage保存图片即可。

**三、各模块思路和代码详解**

**（1）网络爬虫方面**

爬虫部分的文件组织架构如下图：





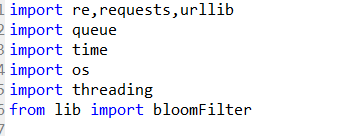
main.py是主函数，imaScript.py用于实现图片的爬取，而bloomFilter.py用来实现布隆过滤器，而mmh.c是在网上查找的一个哈希随机性非常强大的哈希函数，（由于实验初始阶段布隆过滤器写错导致网页爬取速度非常慢，但没当即检查出来，因此曾怀疑过哈希函数的速度太慢以及随机性不好导致形成环路，便先引入了现成的哈希函数，也尝试了如多线程等方法，后来才发现其实是布隆过滤器写错了）

1.imaScript.py

**（a）准备工作与初始化**

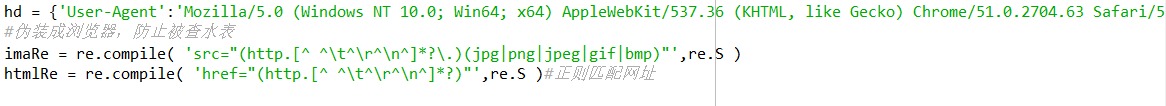
该文件引用了许多外部库以及自己所实现的bloomFilter

其中，re用于正则表达式的匹配，request用于网页内容的获取，urllib用于图片的下载，queue用于队列的实现，time用于获取系统时间，os用于文件目录的操作，threading用于多线程的实现，



hd是浏览器的标识

imaRe和htmlRe是匹配图片链接和网页链接的正则表达式



把imaScript封装为一个类，便于代码的逻辑清晰与实现，其中参数的意义在注释中已经注明。

htmlStart:网络图片爬取的起始网页

rootPath:图片存储的根目录

htmlMaxCount:需要爬取的最大网址数

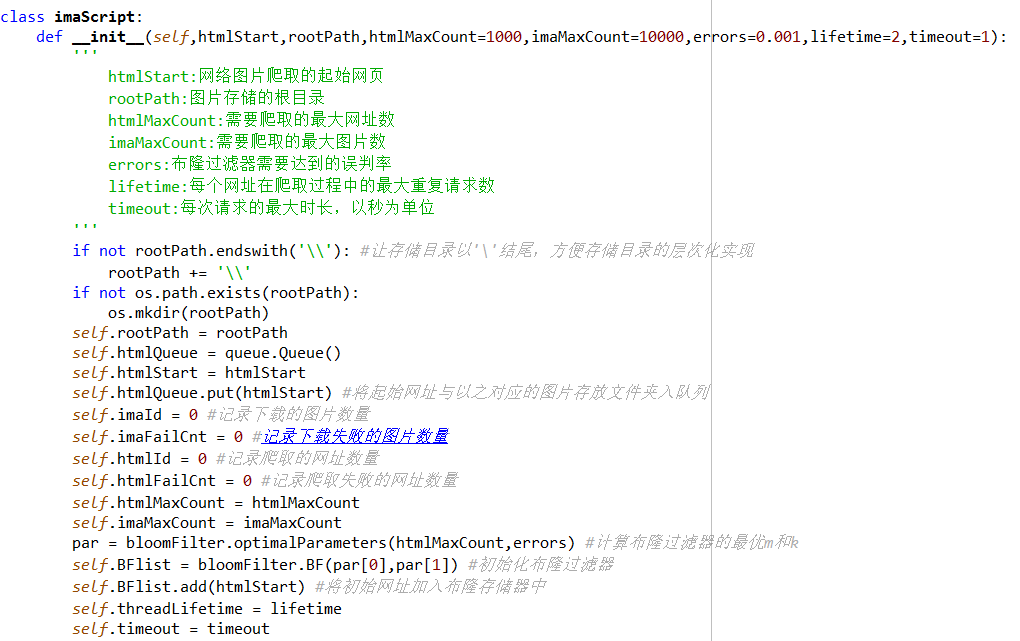
imaMaxCount:需要爬取的最大图片数

errors:布隆过滤器需要达到的误判率

lifetime:每个网址在爬取过程中的最大重复请求数

timeout:每次请求的最大时长，以秒为单位

\_init\_是imaScript类的构造函数，同时初始化布隆过滤器



**（b）run函数**



该函数主要用来执行爬虫的主程序，其中thread\_num是允许开启的最大线程数，通过\_\_mulScript执行多线程爬虫，并且记录时间等信息，待爬虫结束后输出信息。

**（c）\_\_mulScript函数**



\_\_mulScript函数是一个私有的函数，通过布隆过滤器的检验，将非重复的网址存入htmlQueue队列，同时在threading中添加相应的待执行线程，最后再执行。内置有htmlGet函数，通过该函数，获取html中的所有子链接，在布隆过滤器检验不形成环路后，存入htmlQueue和threading两个队列中。具体代码中有注释。



\_\_mulScript函数的剩余部分用于多线程的开启，先将爬取根网址图片的线程入线程队列，再深度优先遍历搜寻跳转链接，并且设置守护线程，以及开启阻塞主线程的执行



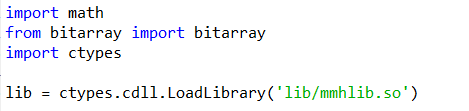
**（d）\_\_script函数**

该函数负责将html中的所有图片下载到rootPath中，并且在该函数中，也负责一些异常情况以及终止的响应和处理。若多次请求访问失败，则直接返回。若数量达到上限或者响应码异常，亦返回。若下载失败，也返回。



2.bloomFilter.py

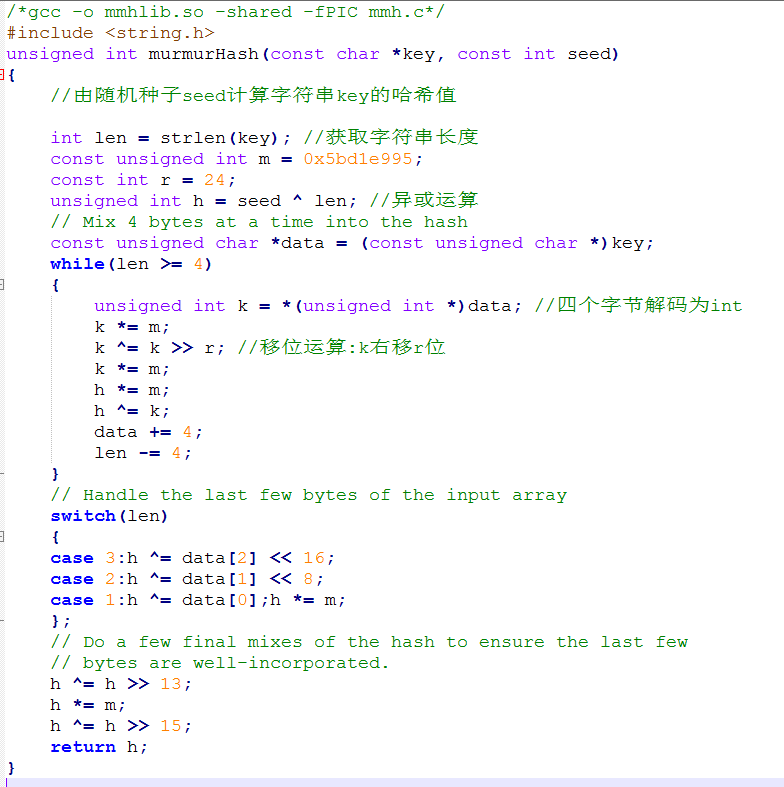
**（a）准备工作与初始化**



bloomFilter.py用来实现布隆过滤器，而mmh.c是在网上查找的一个哈希随机性非常强大的哈希函数

（由于实验初始阶段布隆过滤器写错导致网页爬取速度非常慢，但没当即检查出来，因此曾怀疑过哈希函数的速度太慢以及随机性不好导致形成环路，便先引入了现成的哈希函数，也尝试了如多线程等方法，后来才发现其实是布隆过滤器写错了）

在这里引用了基于c语言的哈希函数mmh

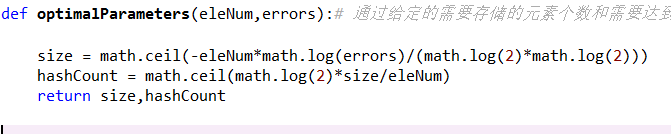


这次也学习了一下用python调用c的一些方法和技巧，通过命令行窗口在运行的目录下键入gcc -o 生成文件名.so -shared -fPIC mmh.c。然后会在目录生成一个 .so文件 这个文件就可以直接被python调用了。

python调用的方式就是通过 ctypes库。然后就可以调用C的函数。这里有个要注意的地方是在python里调用这个传入字符串的时候，要传入字符串的二进制流，而不能直接传unicode编码的str，在传入的时候，加个.encode()就可以了，encode将字符串重新编码成二进制的形式。

**（b）optimalParameters**

通过给定的需要存储的元素个数和需要达到的假阳性率计算布隆过滤器中的最优二进制数组大小和哈希函数个数，其中计算公式在前文有。

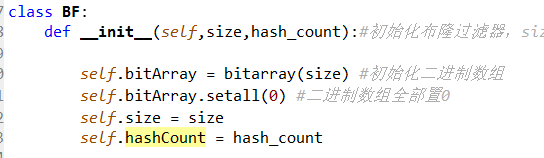


**（c）BF类**

实现一个BF类，BF类中包含三个函数，分别是\_init\_，add，isContains

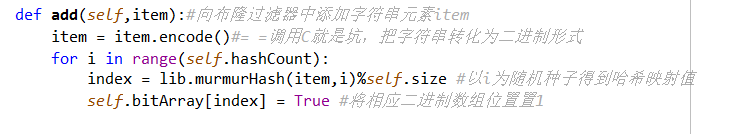
**·\_init\_：构造函数**

初始化布隆过滤器，其中size为布隆过滤器中二进制数组的大小，即m。hash\_count为哈希函数的个数,即k。在这里用了bitarray类型来方便操作，其中m和k为上面的optimalParameter计算出来的使得假阳性概率低于0.01%的合适的m和k。

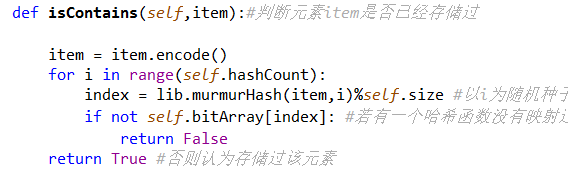


**·add：添加元素**

对于传入的item，通过调用哈希函数，向布隆过滤器中添加网址item

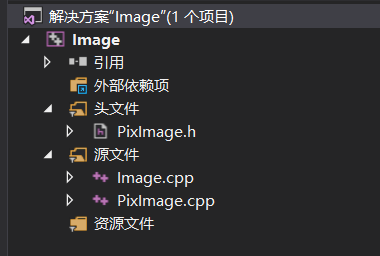
**·isContains：添加元素**

对于网址item，通过计算其哈希值，在bitArray中检索，看是否访问过



**（2）图片处理方面**

图片处理部分的文件组织架构如下图：



PixImage.h是对于PixImage类以及一些全局变量的定义

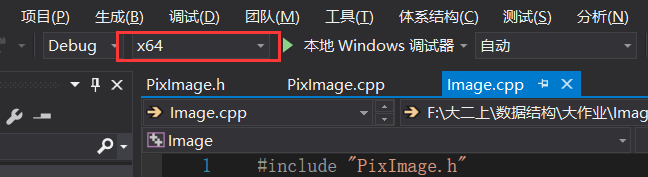
PixImage.cpp是对PixImage类内部函数的具体事项

Image.cpp是main函数所在的文件，用于与用户交互

本次实验用了opencv库对图片进行解析



在配置好系统的环境变量等变量之后，在vs2015中设置

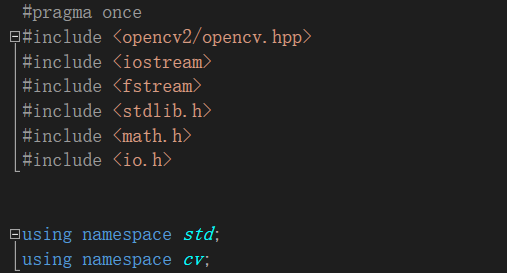


选择x64的环境以适配opencv的要求

1. PixImage.h

PixImage.h是对于PixImage类以及一些全局变量的定义

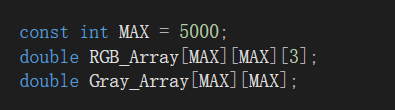
先引用一些函数库 以及声明名空间



定义MAX = 5000，这是图片像素处理的最大宽度

同时定义两个数组RGB\_Array和Gray\_Array来存储待处理图片的信息

（其实这是在Image.cpp中定义的，但是都是属于定义，就放这一起说明）



最后便是整个PixImage类的定义



PixImage类中定义了一些类的公共变量和私有变量，其中

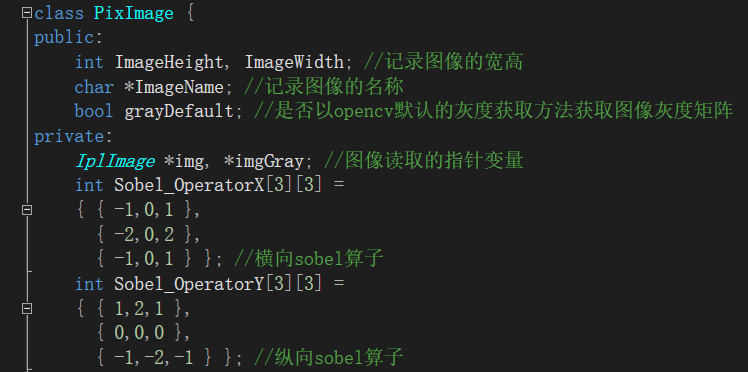
ImageHeight和 ImageWidth记录图像的宽高

ImageName记录图像的名称

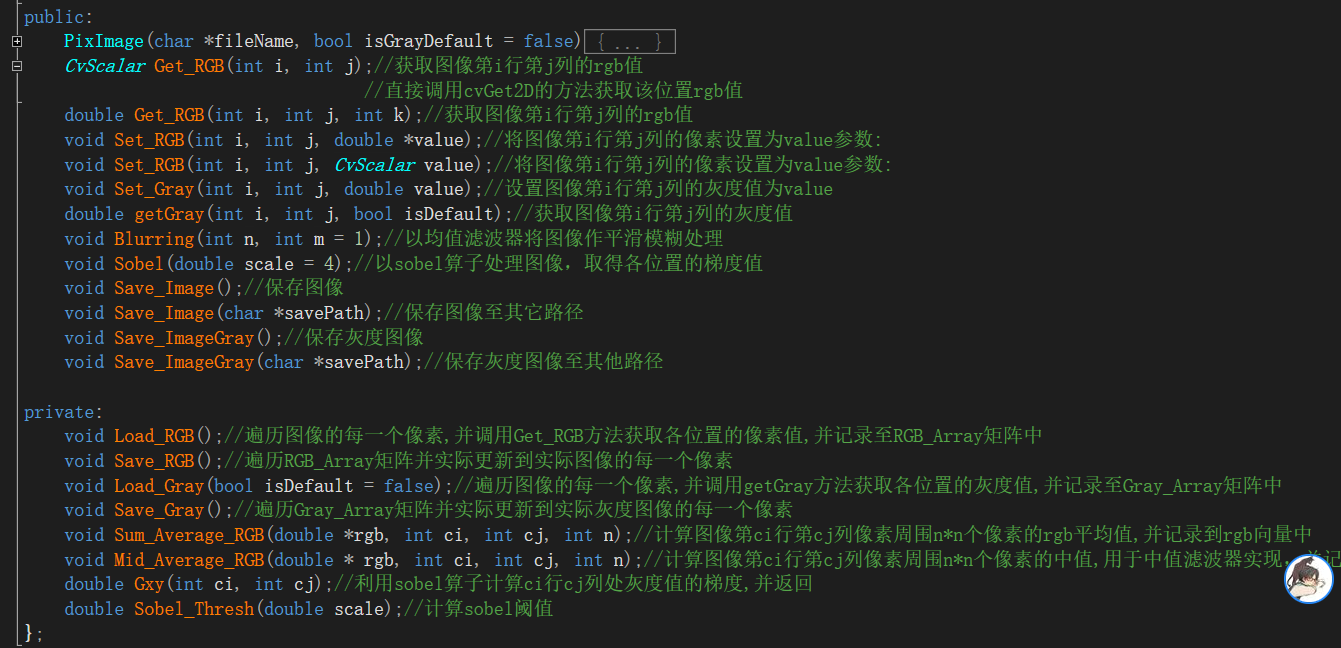
grayDefault表示是否以opencv默认的灰度获取方法获取图像灰度矩阵

\*img, \*imgGray是 图像读取的指针变量

并且用预先定义的3\*3的矩阵记录sobel算子，方便后来的边缘化检测



PixImage类也同时定义了对应图像的一些处理函数：

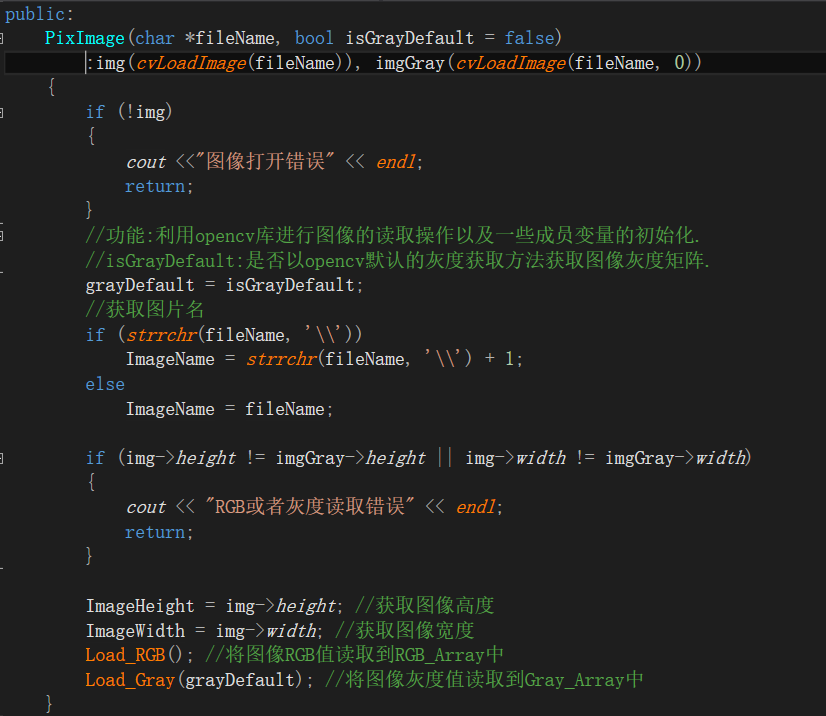


其中构造函数负责PixImage类的初始化处理

fileName 顾名思义，就是所打开图片的路径即名称

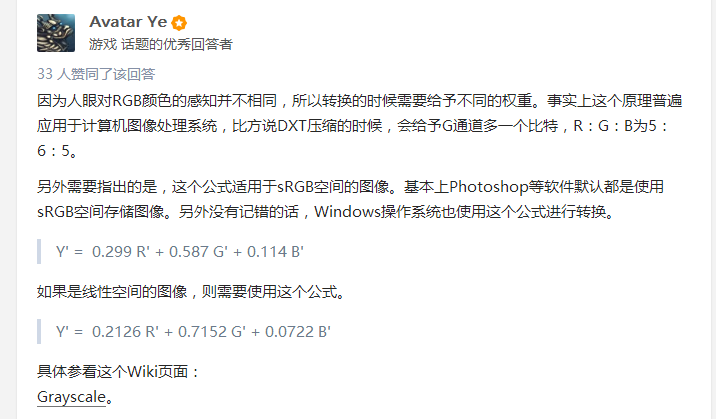
isGrayDefault表示是否以opencv默认的灰度获取方法获取图像灰度矩阵

通过构造函数，获取待处理图像的长宽，并且利用Load\_RGB和Load\_Gray这两个函数将图片的色彩和灰度信息存储在自定义RGB\_Array和Gray\_Array中



其中，灰度的处理在实现的过程中发生了一些情况

在知乎上可以查到,RGB和灰度是有一定的转换关系的：



而opencv也自带有读取灰度的函数

用一张色彩鲜艳对比明显的原图作为参考：



使用opencv自带的函数得到的图

isGrayDufault=true的图：



通过灰度计算公式得到的图

isGrayDefault=false：



通过比较可以发现，这两幅图的灰度细节有些不一致

因此为了保险起见，还是选择用opencv的基本功能

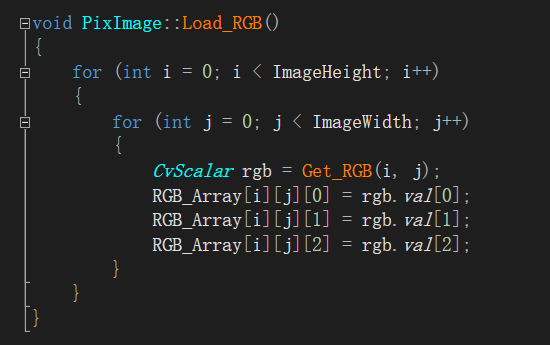
2. PixImage.cpp

下面结合代码说明图像处理的基本思路和实现方法

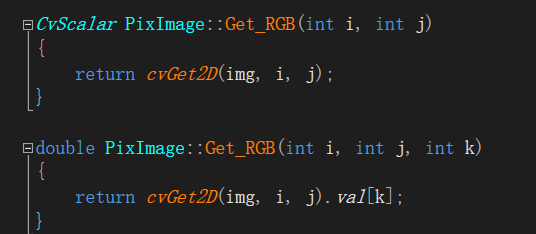
### （一）模糊处理

#### （1）读取图像信息

在构造函数中便通过Load\_RGB函数从图像中读取图片各个像素点的RGB值 信息，将图像RGB值读取到RGB\_Array中



其中用到了Get\_RGB函数，获取该位置RGB值。根据其形参的不同，有一个函数重载。其中CvScalar是opencv提供的一个结构体类型可以利用cvGet2D返回该点的RGB值，用返回类型为double的重载函数可以根据k的取值返回特定的R/G/B值。



#### （2）处理图像

图片模糊处理的主函数是Blurring函数

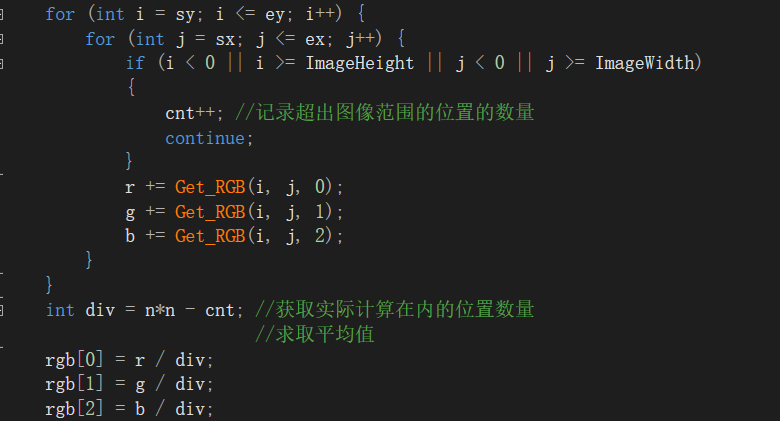
其中n是均值的范围，m为迭代的次数



均值滤波器的基本原理是：对于任意一个像素，使用其周围 n x n 像素范围内的平均值来置换该像素值。因此我们使用Sum\_Average\_RGB函数来计算其周围n x n 像素范围内的平均值，这里处理上的细节中定义了一个double \*rgb = new double[3]，用指针直接的数组直接将平均值记录在rgb中。

处理时对于超出边缘的像素点也要排除，使得平均得到的是真实值。



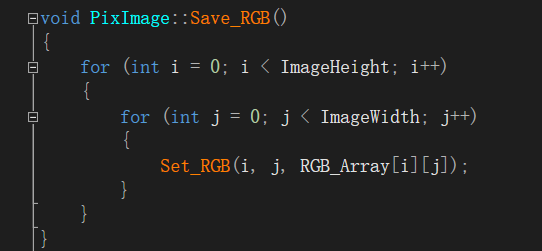


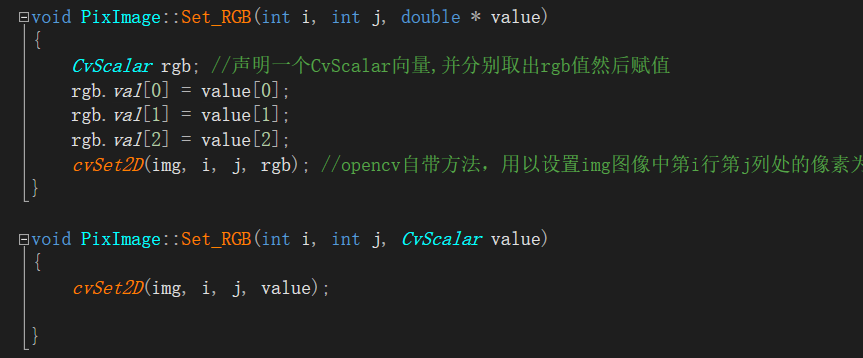
#### （3）保存图像信息

图片的保存用的是Save\_Image函数，如果未指定路径则默认保存在当前目录，否则保存在制定的路径中。



保存图像的原理是，通过Save\_RGB，调用Set\_RGB，把修改后的RGB\_Array赋值给\*img指向的图片，然后调用cvSaveImage保存图片。

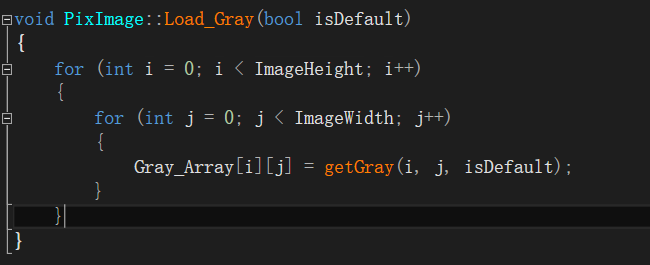




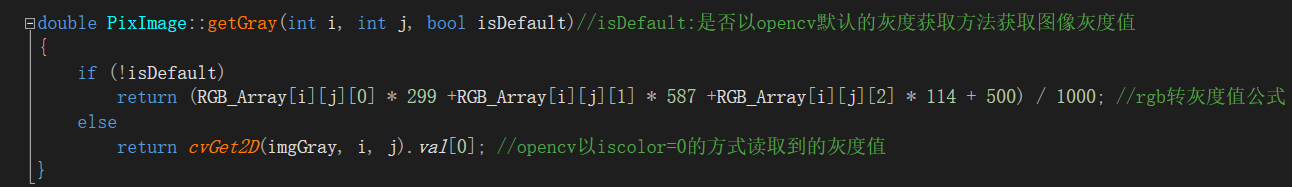
### （二）边缘化检测

#### （1）读取图像信息

在构造函数中便通过Load\_Gray函数从图像中读取图片各个像素点的灰度值 信息，将图像灰度值读取到Gray\_Array中



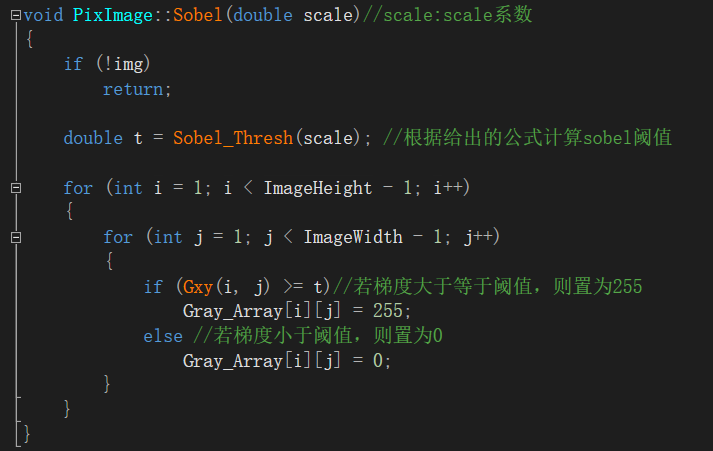
其中用到了Get\_Gray函数，获取该位置灰度值。根据其isDefault的不同，选择是采用opencv自带的灰度获取方式还是根据RGB值根据公式来计算，为了保险起见，在这次处理中都使用opencv自带的函数读取灰度值。



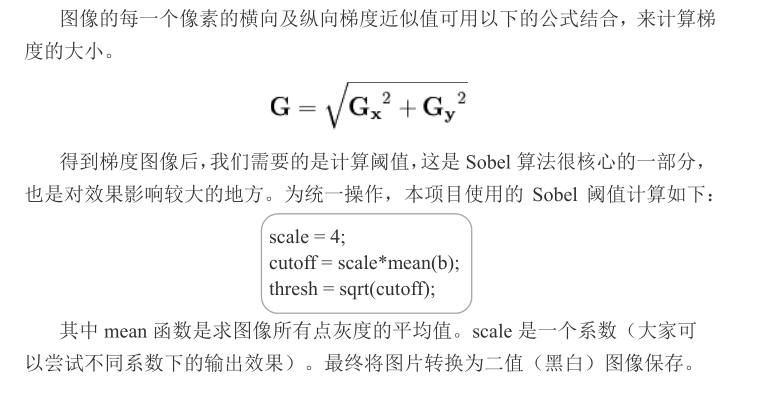
#### （2）处理图像

图片模糊处理的主函数是Sobel函数

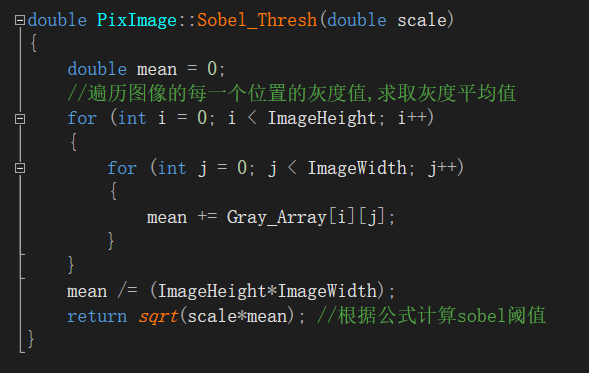
其中scale为scale系数



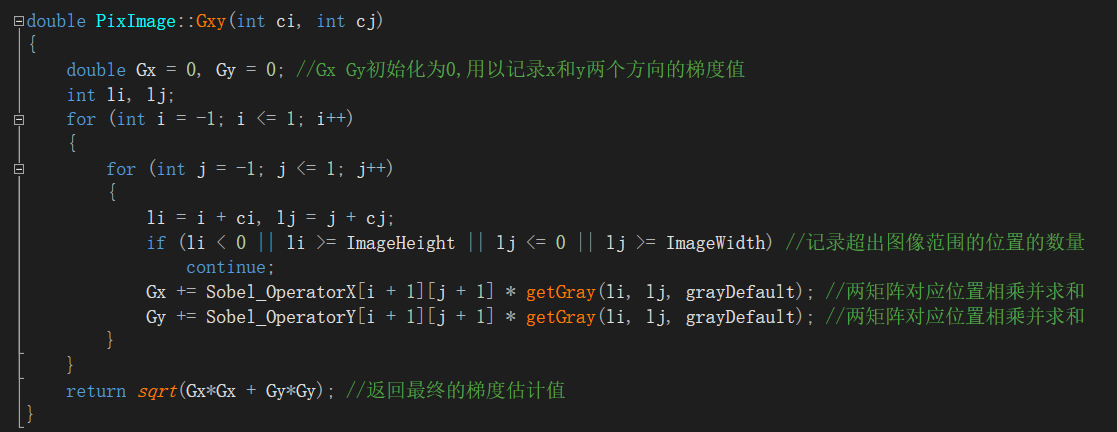
其原理是



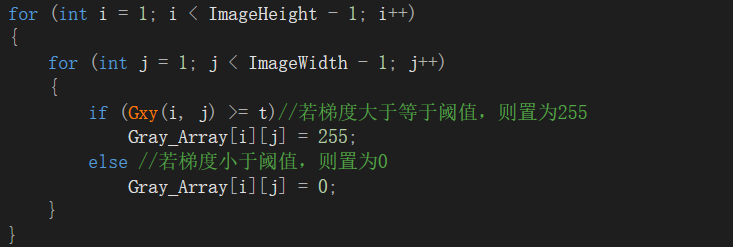
索贝尔边缘检测中，首先利用Sobel\_Thresh根据给出的公式计算sobel阈值



然后通过公式计算梯度：



最后将梯度与阈值做比较，若大于阈值则置255，否则置0。由于索贝尔算子是3\*3的矩阵，因此这里循环从1开始，对边缘的情况进行忽略。

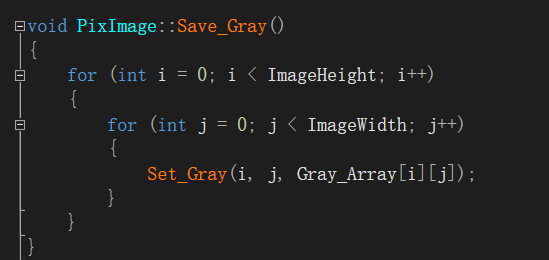


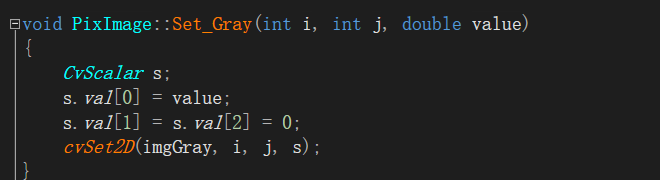
#### （3）保存图像信息

图片的保存用的是Save\_ImageGray函数，如果未指定路径则默认保存在当前目录，否则保存在制定的路径中。



保存图像的原理是，通过Save\_Gray，调用Set\_Gray，把修改后的GrayArray赋值给\*imgGray指向的图片，然后调用cvSaveImage保存图片。

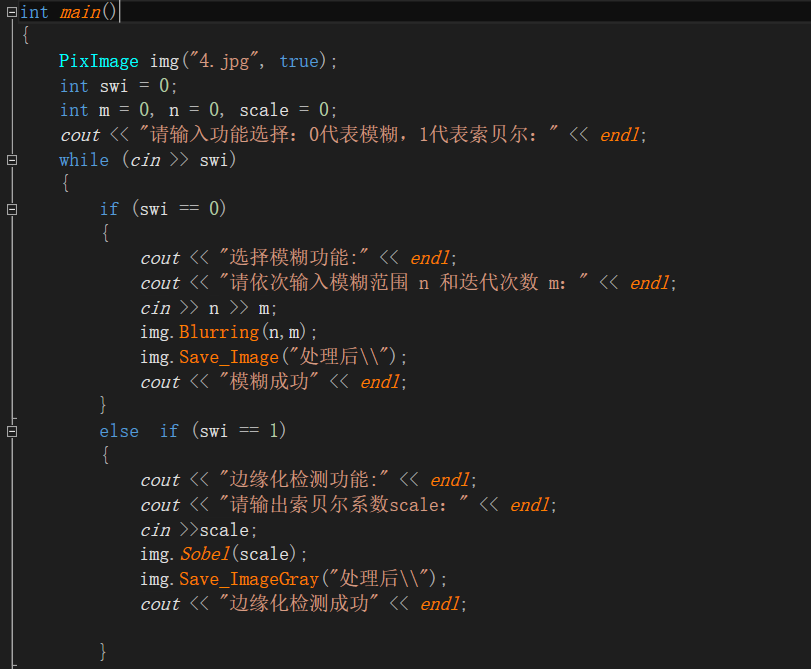




3. Image.cpp

Image.cpp是main函数所在的文件，用于与用户交互。

用户根据从键盘键入数据，选择想要实现的功能。



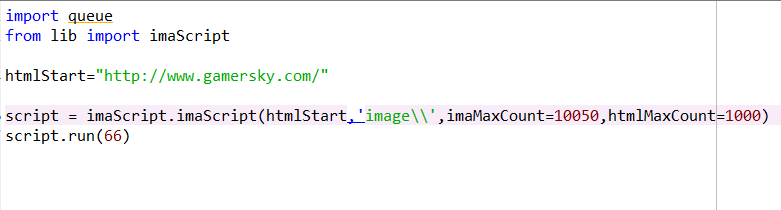
**四、最终项目达到的效果**

## （1）网络爬虫

选取<http://www.gamersky.com/>为起始网站

设置最大爬取图片数为10050（有些图片可能下载失败）

设置最大爬取网址数为1000



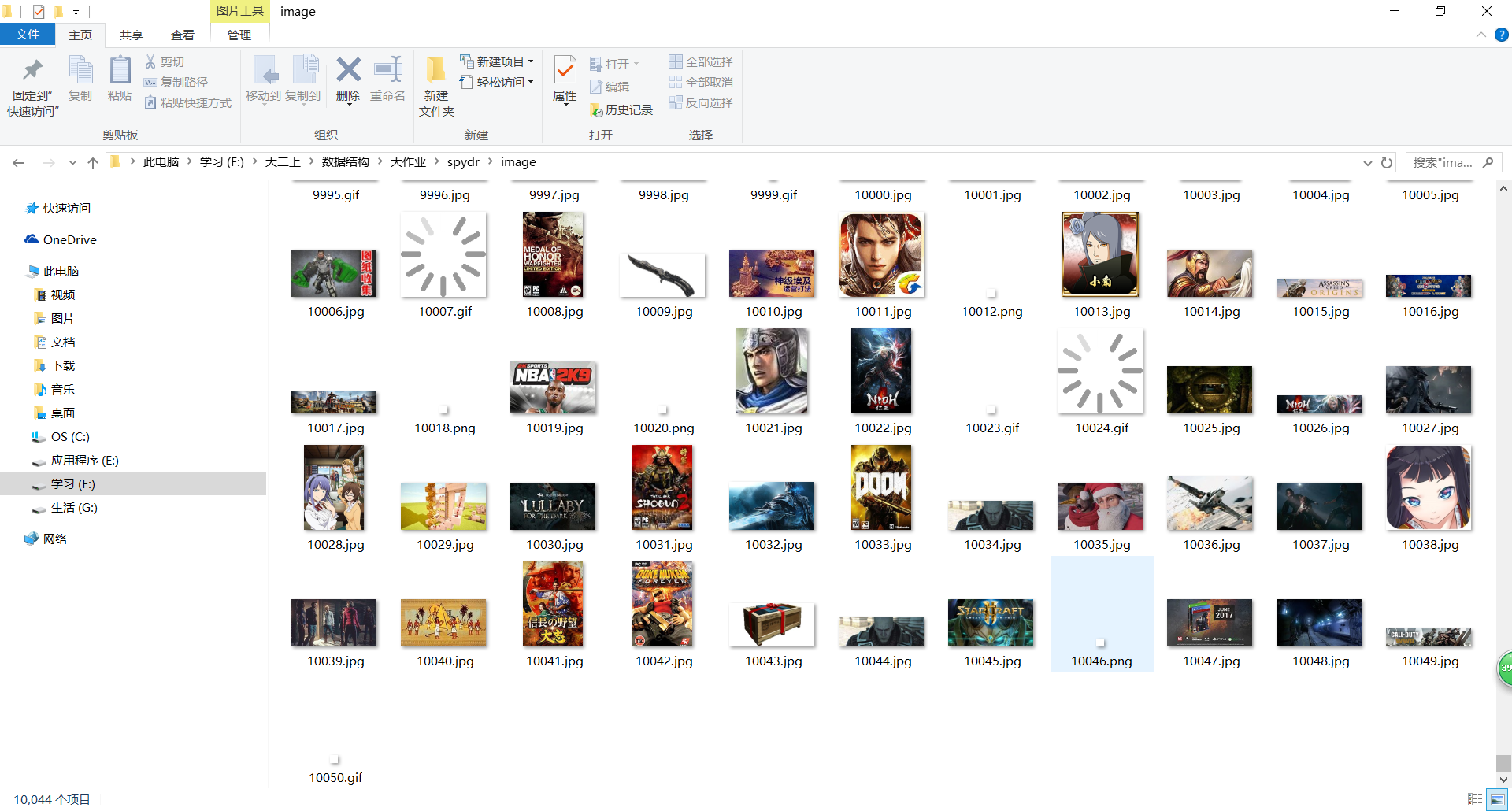
运行爬虫





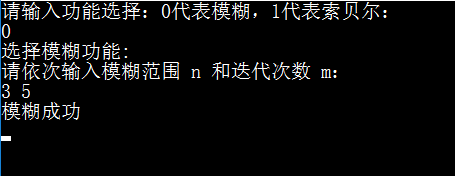
最终能够实现下载至少10000张图片的功能





## （2）图像处理

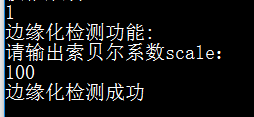
#### 1.图片模糊







#### 2.边缘检测







可以看出实验的结果符合预期和要求。

**五、附录**



在处理灰太狼图像时，发现很难达到样例的效果

查阅资料后发现，对于这类有白色噪点的图片，用中值滤波器可能好一些，于是试着把程序改了一下。其中用了vector和<algorithm>里的sort函数，但是这样一次处理图片需要很长时间，因为每次的排序的时间还是蛮长的。估计是对于这个中值的获取，有着更为简单的方法。

