图像匹配技术文档

2013013297 朱子晨 软件 31

一、方案设计

无论是哪一个模式，总体的方案设计类似，都是**先筛选再细粒度匹配**的方式，这样可以在一定程度上**减少时间复杂度**。在进行匹配时候，**第一步**，先对原图或者待匹配的子图进行处理，然后**根据子图的第一行匹配**，筛选出原图中的一部分点，如果筛选后的点的总数刚好为一，那么就可以认为这个点就是我们的最终结果，程序结束；**第二步**，如果筛选后的点的总数大于一，那么对剩下的点进行全图匹配，也就是遍历这些筛选过后的点，根据这些点在原图的位置，找出和待匹配子图同样大小的区域，进行全局匹配，全局匹配完成时会产生一个误差值，也就是一个点坐标在原图的位置找到的子图和待匹配的子图之间的误差，最后在这些误差值中选取一个**误差最小**的点作为我们最后匹配的结果。

二、实现难点

本实验的实验主要难点在于由于干扰的类型不同，所以不同的模式下的图片行匹配方式（用于筛选处理）都不一样，误差计算方式（用于最终匹配）也不一样。具体的实现方式如下表所示：

	原图处理	**待匹配图处理**	**行匹配方式**	**误差计算方式**
模式 0	无	无	某个像素值（RGB 会分别匹配一次）相等即可	不相等的像素值的个数（一个点可以有 0 到 3 个不匹配的像素值）
模式 1	灰度近似	灰度近似 与相邻像素点的灰度平均数	在某个误差范围内，可以认为相等。（会除去若干个匹配失败的离群点）	处理过后的图像在对应点上的灰度差值的绝对值之和
模式 2	灰度近似 与相邻像素点的灰度方差	灰度近似 与相邻像素点的灰度方差	在某个误差范围内，可以认为相等。（会除去若干个匹配失败的离群点）	处理过后的图像在对应点上的计算过后的方差的差值的绝对值之和
模式 3	灰度近似	灰度近似 与相邻像素点的灰度中位数	在某个误差范围内，可以认为相等。（会除	处理过后的图像在对应点上的灰度差值的

			去若干个匹配失败的离群点）	绝对值之和
模式 4	无	无	待匹配点的某像素值（RGB 三个值都会各自进行判断）小于原图某点的像素值	不满足小于条件的像素值个数（一个点可以有 0 到 3 个不满足像素值）

但是，表中仍然有很多需要注意的问题需要考虑。

实现重难点一：什么样的灰度近似最合适？

一般来说，根据 RGB 计算灰度的方法如下：

Gray = `0.299 * red + 0.587 * green + 0.114 * blue`

但是，灰度也可以近似计算，近似的处理方式有很多种，有**取 RGB 平均数，Green 值代替**这两种更为简单的计算方式，那么选取哪种更为合适呢？

在选择的时候，我们需要考虑的是灰度计算的复杂性对时间复杂度的影响，和更重要的灰度近似值对最后结果的影响。很明显最正确的灰度计算会花费更多的时间，而直接用 Green 值代替的方法会花费更少的时间。相应的计算方式已经在 getDustArray 函数中被注释掉，经过实验发现直接 Green 值得到的结果正确率在合理的范围内，所以最后我们选取直接用 Green 值代替灰度值的方式来作为灰度近似的方式，而不用花费过多的时间在灰度计算上面。

实现重难点二：计算平均值，中位数和方差时多少的相邻像素点合适？

相邻像素点可以选择，**左右两个，左右上下四个，周围 8 个**，如果相邻的范围更大，对时间复杂度影响更大，那么其他的相邻方式就不在考虑范围之内。

从正确性的角度来看，不应该只关注左右的相邻，所以排除掉**左右两个**这个相邻的选项。而至于**左右上下四个**和**周围 8 个**这两种相邻方式，后者时间复杂度更高。正确性也可能更高。但是在经过我们实验，发现**左右上下四个**这种相邻方式并得到的正确率并不低，所以最后采取了**左右上下四个**这种相邻方式。

实现重难点三：行匹配是否就是全部的行匹配，可否考虑只匹配行中的部分点？

如果行匹配要求不严格，也就是说只匹配部分点，那么多筛选得到的点就可能增多，而导致全局匹配的店增多。虽然在行匹配时减少了时间复杂度，但是在全局匹配时却可能增加复杂度。所以最后考虑**行匹配是整行匹配，而非只匹配行中的部分点**。

实现重难点四：全图匹配是否就是真正的全图匹配呢，可否考虑只匹配部分点？

全局匹配的时间复杂度非常高，因为一个子图往往非常大，如果真的做到全局匹配的话会在把时间复杂度提高很多很多。所以我们很有必要在全局匹配时选取全局的部分点进行匹配，最后实现时，我们考虑**只匹配全局匹配中的某个二十分之一，来进行误差计算**。由于子图原本比较大，取十分之一（数据量仍然很大）对实验结果的正确性不产生太大的

影响。

实现重难点五：是否需要二级筛选，再进行细粒度匹配？

很明显这需要根据一级筛选后剩下的点的总数决定。在我的程序中，**模式 1，模式 2，模式 3，实现了二级筛选**，当一级筛选剩下的点的总数仍然大于 10000 的时候，我们考虑二级筛选，二级筛选之后才进行最后的细粒度匹配。

三、实现亮点

一维数组和二维数组的时间复杂度比较

可以发现，SRC 文件夹下有两个 hs 文件，Match.hs 文件是通过一维数组记录一张图的每个像素点的信息，而 Match2.hs 文件是通过二位数组记录一张图每个像素点的信息。在实现时，二者不同情况下有不同的实现效率，在保证文件和图片路径正确的情况下，我们通过 bat 文件(**为了保证 bat 文件正常执行，我把 img 的文件夹放在 SRC 文件夹中，另外在保证相应的 exe 文件和 Enter.txt 存在情况下，我们可以通过 bat 文件测试时间**)测试了两个实现方式的时间差异。下图是我在模式 0，模式 1，模式 4 下对时间统计的实验结果：

模式	图片	Match.hs（一维）	Match1.hs（二维）
模式 0	img1	1.48s	0.63s
	img2	1.96s	1.13s
	img3	1.39s	0.44s
	img4	6.08s	3.51s
	img5	1.65s	0.56s
模式 1	img1	28.09s	20.73s
	img2	167s	5min54s
	img3	16.24s	11.73s
	img4	12min 15s	17min51s
	img5	37.5s	63.94s
模式 4	img1	47.44s	47.15s
	img2	2.19s	1.4s
	img3	1.5s	0.55s
	img4	7.34s	4.63s
	img5	14.72s	14.41s

以上是时间复杂度的统计，其他两个模式没有做系统的统计，二维数组的效率比一维数组的效率稍高。**正确度方面，除了模式 2 之外，其他的模式得出的匹配点误差都是在可以接受的范围之内**，可能是在在模式 2 下对图片的预处理的方式上不应该用方差的方式来进行处理。**时间复杂度方面，除了简单的模式 0 和模式 4，其他模式下大图可能会导致运行更多的时间。**

四、程序使用方法

SRC 文件夹下有两个可编译的 hs 文件，分别是 Match.hs 和 Match1.hs，两者都可以通过 ghc 进行编译（**注意：使用 ghc 进行编译时，请加入"-package bmp"参数，此次作业中使用了 bmp 库来读取图片的信息**），另外由于我的作业是在 windows 系统下完成的，所以程序的运行时间是通过 bat 文件统计的，各个模式下的运行时间在上一个模块已经展示过。

五、感想和体会

经过这次图像匹配的实验，不仅让我对图像匹配方面的知识有了初步的了解和认识，更是加深了我对 Haskell 的理解。此外，本次作业没有加入并行也是一个遗憾的地方，因为程序编写的思路问题，并行最多只能提高在图像预处理的时候的效率，而本身这个图像预处理（如灰度近似，中位数计算）是不用花费多少时间的，而需要花时间的筛选和误差排序，这两个大程序只能是先后条件下运行的，所以不能把这两部分合在一起进行并行计算。另外很遗憾的一点是，本次实验很多地方都有代码重复，Haskell 的代码风格还需要好好提升。还需要注意的一点是，程序中有很多**注释掉的代码**，还有一些**没有用到的变量**（因为不会用到，所以不会对变量求值），那些都是在之前实现难点中提到的问题中为了**测试各个选择下的时间复杂度的影响**而留下的，对于主程序不产生其他影响。