Meanshift算法结构设计

1. Meanshift滤波：
   1. 对每一个像素点Pixel(x, y), 进行固定次数的循环:
      * 设置cur\_color为像素点的LUV值

设置color\_sum = [0,0,0], index\_sum = [0, 0], summed\_pixels = 0

* + - 循环：对以带宽dh为半径的圆或正方形邻域内的像素点
      1. 计算ΔLUV
      2. 如果|ΔLUV|^2 <= color\_difference\_range,

-> color\_sum += LUV

-> index\_sum += pixel所在位置

-> summed\_pixels++

* + - 求color\_sum, index\_sum平均值(除summed\_pixels), 记为cs\_avg, is\_avg
    - cur\_color = cs\_avg
    - 令shift = |cs\_avg|^2 + |is\_avg|^2 (如果你想修改两者影响权重也是可以的)

如果shift < shift\_range, 则break

* 1. 将像素点的值更新为cur\_color

以上是原始的Meanshift 滤波算法, 初步想可以不进考虑颜色和index偏移量(像素点间距离), 还可以考虑纹理, 即用卷积核卷积, 然后将ΔLUV向量修改为多个卷积核的卷积值组成的向量.

1. Flooding
   1. 对每一个像素点Pixel(x, y), 赋值一个独特的label(比如y\*width+x), 存在全局二维数组中, 并进行固定次数的循环(如20次)
      * 读取邻域像素点的值(可以半径为1, 即8个, 或者半径为2, 即 25-1=24个等等, 看图像大小, 大的图像需要更大的读取半径)
      * 如果邻域像素点的ΔLUV在范围内, 则将邻居的label修改为自己的label

同步所有线程

注意, 存在这样一种情况, 即两个像素点抢一个像素点的情况(赋给自己的label), 这会由线程间的竞争决定最后谁赋值成功, 不过无所谓, 因为最后还要进行区块合并, 所以不影响.

为了减少过于密集的冲突和重复赋值, 可以每n行,每n列(n<半径) 让一个像素点线程进行洪泛搜索, 每次循环将需要将搜索的像素点进行平移(如每次将行+1), 以扩展邻域

另外需要记录每个洪泛搜索后剩下的标签, 以及每个标签对应的区域的平均LUV值(或平均纹理向量)

1. Union Find

其实就是kruskal算法, 先把每个区域作为一个单节点树, 然后通过比较每个区域对应的特征向量(LUV, 纹理), 如果Δ模长在一定范围内, 则合并两棵子树(给对应区域的像素点重新打标签). 用cuda并行化可能不太方便, 一般区域数量在几百个左右, 可以考虑pthread并行. 最好是两两比较, 这样能尽可能找到相近的区域, 并且要lockstep, 即:

两两比较->合并->更新均值向量->两两比较->合并->更新均值向量->两两比较->合并->更新均值向量…

每个step就是比较,合并,更新

工作:

1. 完成Meanshift滤波部分, 使用LUV, 然后加入卷积运算, 只使用LUV的已经有opencl的并行化代码可以参考

**此外需要进行详尽的性能测试和优化**

1. 完成洪泛搜索的并行部分, 这个部分是用simd并行的, 效果不好,因为实际还是串行算法, 需要用cuda按上面的描述重新实现

**此外需要进行详尽的性能测试和优化**

1. 完成Union Find并行化

已经有了串行代码, 希望能实现更快的并行化

1. 使用cuda并行, 可能要改进算法
2. 使用pthread并行

**此外需要进行详尽的性能测试和优化**

1, 2, 3-1, 3-2为四个任务, 第五个任务是撰写报告和相关论文.

参考链接:

<http://rsbweb.nih.gov/ij/plugins/download/Mean_Shift.java>

github下的opencv.meanshift项目