

数据结构：The Art of Compression

姓名：鲁国锐 学号：17020021031 专业：电子信息科学与技术

2019.4.18 Thursday

Contents

1 问题分析	2
1.1 题目描述	2
1.2 问题分析	2
2 解决方案	2
2.1 <i>stats</i>	2
2.1.1 构造函数	2
2.1.2 <i>getSum</i> 和 <i>getSumsq</i> 函数	2
2.1.3 <i>getScore</i> 函数	3
2.1.4 <i>getArea</i> 和 <i>getAvg</i> 函数	3
2.2 <i>twoDTree</i>	3
2.2.1 构造函数	3
2.2.2 剪枝函数	3
2.2.3 <i>render</i> 函数、 <i>copy</i> 函数和 <i>clear</i> 函数	4
3 算法设计	4
4 编程实现¹	6
5 结果分析	17
5.1 结果展示	17
5.2 分析	18
6 总结体会	18

1 问题分析

1.1 题目描述

本次任务要求对图像进行有针对性的模糊化，即颜色多样性越低的地方模糊程度越大，颜色多样性越高的地方模糊程度越低。具体而言，我们的任务就是将给定的代码补全，使其能够正常运行并输出正确的结果。

1.2 问题分析

根据题目及代码，我们需要解决的问题有：

1. 根据 `stats.h` 文件写出 `stats.cpp` 文件（见2.1节）；
2. 根据 `twoDTree.h` 文件写出 `twoDTree.cpp` 文件（见2.2节）；

2 解决方案

2.1 stats

`stats` 这个类是专门用来辅助构建二叉树的。更具体的说，是为了让我们能够在常数时间内算出给定范围的矩形内所有元素的和、平方和、平均值以及多样性分数（*variability score*）。

2.1.1 构造函数

这里需要我们初始化的成员变量是六个 `vector`，其中的第 y 行第 x 列的元素代表的是对应的 PNG 通道中以 $(0,0)$ 为左上角、以 (x,y) 为右下角的矩形中所有元素之和或平方和。为减少计算量，这里我们采取“动态规划”的思想，可以得到状态转移方程：

$$dp[i][j] = dp[i-1][j] + dp[i][j-1] - dp[i-1][j-1] + im.getPixel(j, i) \quad (1)$$

由于 $dp[i-1][j-1]$ 是 $dp[i-1][j]$ 和 $dp[i][j-1]$ 都包含的部分，相加的时候被算了两次，所以需要减掉一个。另外值得注意的一点是，整个工程中涉及到的所有有关坐标的变量或函数，其第一个值代表的是横坐标，即横向的偏移量，对应过来相当于是列下标；而第二个值是纵坐标，即纵向的偏移量，对应过来是行下标。所以这里我们在调用 `getPixel` 函数是要把 i 、 j 交换位置。

当然，方程1在第一行和第一列上是不成立的，在程序中会导致越界，所以这里我们先单独把第一行和第一列的值填上，再从 $(1,1)$ 开始执行1式。

2.1.2 `getSum` 和 `getSumsq` 函数

有了那六个 `vector` 作为辅助，这两个函数实现起来就会轻松许多。这里我们直接给出公式：

$$\text{given } ul(x1, y1), lr(x2, y2) : \quad (2)$$

$$\text{sum}(ul, lr) = dp[y2][x2] - dp[y1-1][x2] - dp[y2][x1-1] + dp[y1-1][x1-1] \quad (3)$$

同样，3式在 ul 在第一行或第一列上时是不成立的，以在第一行（不包括原点）上为例：

$$\text{sum}(ul, lr) = dp[y2][x2] - dp[y2][x1-1] \quad (4)$$

参照1式和4式我们能得出 ul 在第一列或原点上的表达式。

`getSumsq` 原理同 `getSum`。

2.1.3 *getScore* 函数

根据文档中给出的等式：

$$\sum(x - \bar{x})^2 = \sum x^2 - \frac{(\sum x)^2}{|R|} \quad (5)$$

再结合 *getSum* 和 *getSumsp* 函数，我们可以在 $O(1)$ 时间内算出其多样性分数。

2.1.4 *getArea* 和 *getAvg* 函数

这里的 *getArea* 函数其实是为 *getAvg* 服务的。我们先用 *getSum* 求出对应通道中像素值的和，再将其除以 *getArea* 返回的数量即得平均值。平均值仍然分 r 、 g 、 b 三个通道，用一个 *RGBAPixel* 存放。**注意这里的 *RGBAPixel* 类中还定义了一个透明度通道，但这里我们不会涉及到它，直接遵循默认设定将其置为 1 即可。**

2.2 *twoDTree*

这一块是整个算法最核心的部分。其中构造函数和剪枝函数是最关键的两个部分，我们重点来看一看这两个函数的设计。

2.2.1 构造函数

这里并没有采用 $K-D$ 树中“深度为偶数垂直分割，深度为奇数水平分割”的构造策略，而是对当前矩形的长宽进行比较，若长大于宽，水平分隔；反之则垂直分割。这里我还专门在 *Node* 里面加了一个成员变量 *flag* 来记录对当前矩形是做的垂直分割还是水平分割。现在想想其实这个变量完全没必要，因为 *Node* 中本身就有 *upLeft* 和 *lowRight* 这两个对组，我们只需要简单算一下当前矩形的长和宽就可以知道是做的什么分割了。

在做分割时，我们用一个对组来记录坐标，沿着一条边走，计算被过这个点的垂线所分成的两个小矩形的多样性分数之和。以垂直分割为例，设左上角的坐标为 (x_1, y_1) ，右下角的坐标为 (x_2, y_2) ，我们令一个点 i 的初始坐标为 (x_1, y_2) ，让其沿 $y = y_2$ 这条直线向右移动，同时令另一个点 j 的初始坐标为 $(x_1 + 1, y_1)$ ，沿着 $y = y_1$ 向右移动。这样在移动的过程中 i 始终是左边矩形的右下角，而 j 始终是右边矩形的左上角。在此过程中我们不断计算两个矩形的多样性分数。而考虑到两个矩形的多样性分数可能不会同时达到最小，所以这里我们取令二者之和最小的分割方式。然后再递归地进行下去，直至当前矩形的左上角坐标等于右下角，即矩形中只有一个元素时，停止递归，并返回当前节点的指针。

除了分割外我们还需要调用 *stats* 类，来帮我们计算出该矩形范围内 RGB 通道的平均值，记录在 *RGBAPixel* 类型的成员变量 *avg* 中，另外我还额外给节点类增加了一个变量 *split_value*，用来记录分割的位置。

2.2.2 剪枝函数

这里我们再明确一下剪枝的要求：给定两个值 *percent* 和 *tolerance*，我们对某一棵子树计算其根节点与每一个叶节点 *avg* 之差的平方（注意这里要把三个通道的值都加起来），如果计算出来的值大于 *tolerance* 且这样的叶节点数量占总数的比例大于 *percent*，我们就要给这个子树剪枝，即只保留根节点。

但对每一个节点都执行一次遍历子树的操作计算成本太高，我们希望能够对每个节点只遍历一次。所以这里我们用 *vector* 来作为函数的返回值。当碰到叶节点时，我们将叶节点放入一个向量中返回；而对于其它节点，先进行递归得到左右子树的叶节点，将两个向量合并之后再开始判断是否需要剪枝，最后返回合并后的向量即可。

2.2.3 render 函数、copy 函数和 clear 函数

在 *render* 函数中，我们要实例化一个 *PNG* 对象，然后根据每一个像素的坐标在构建好的二叉树中找到相应的节点，把该节点的 *avg* 赋值过来即可。

copy 函数和 *clear* 函数比较简单，在此不做赘述。

3 算法设计

见图1、图2

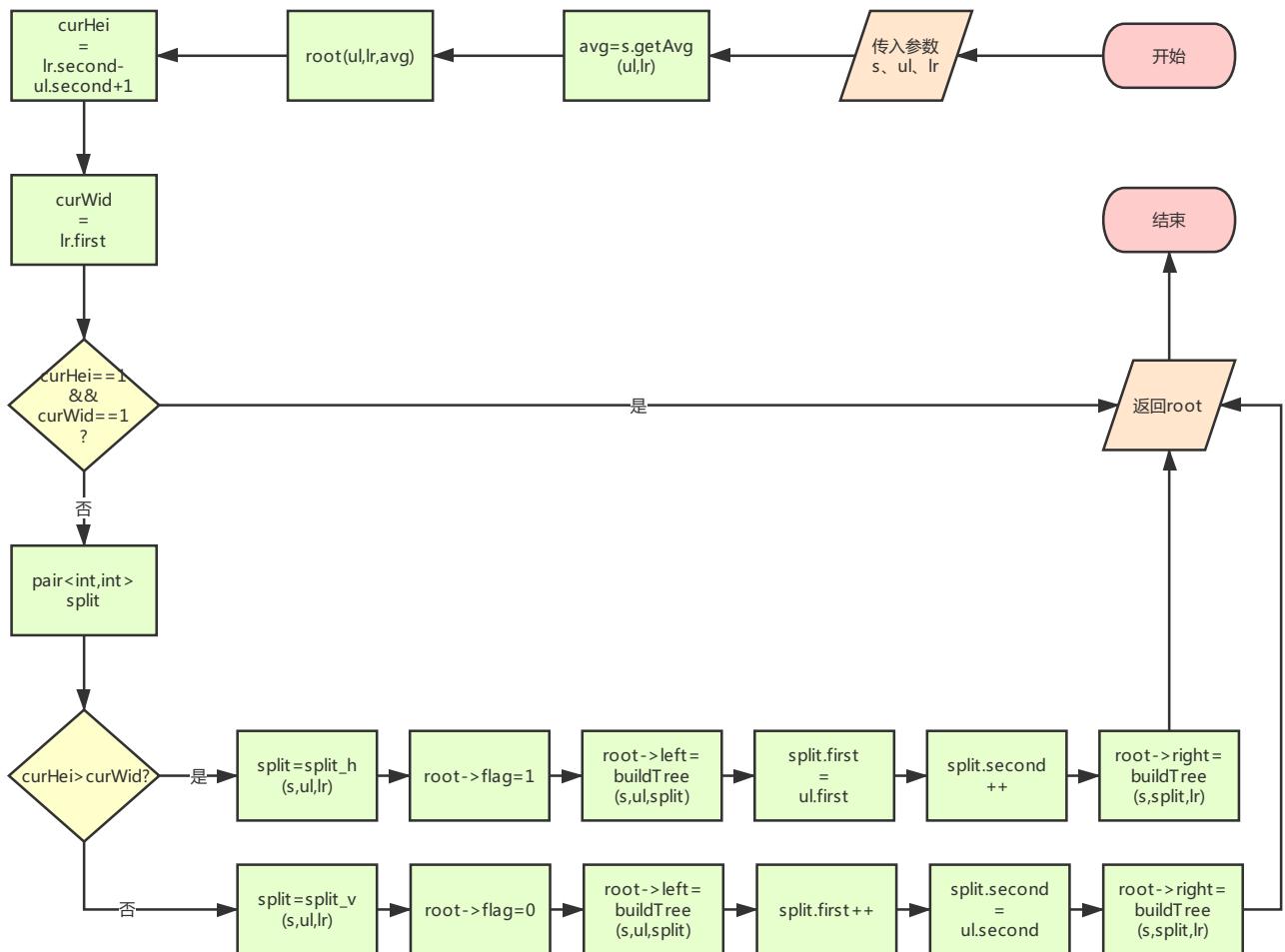


图 1: *buildTree* 函数流程图

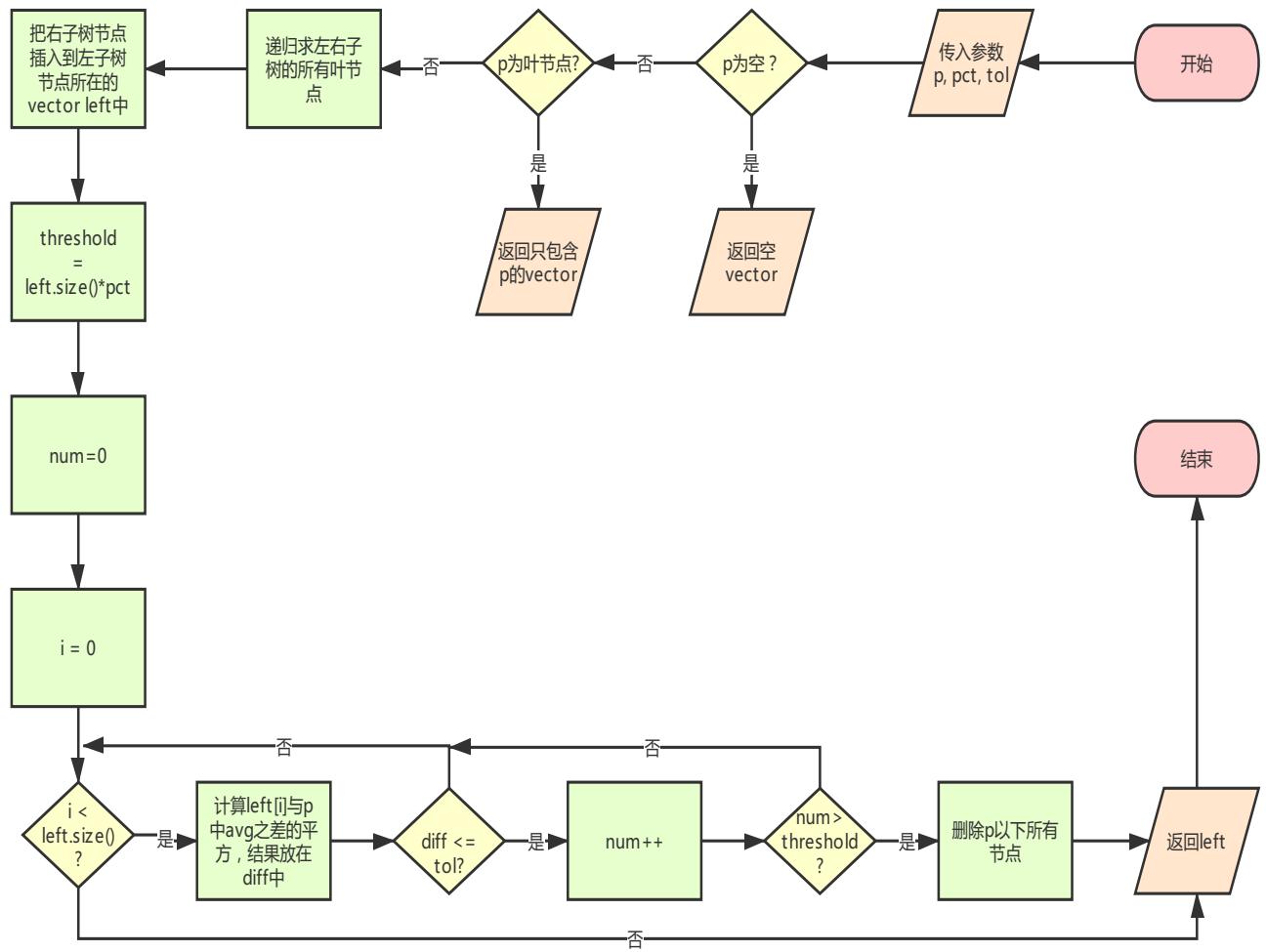


图 2: *prune* 函数流程图

4 编程实现¹

Listing 1: stats.cpp

```
1 #include "stats.h"
2 #include <iostream>
3 #include <cmath>
4
5 stats::stats(PNG &im)
6 {
7     for (int i=0; i < im.height(); i++)
8     {
9         sumRed.push_back(vector<long>(im.width()));
10        sumsqRed.push_back(vector<long>(im.width()));
11
12        sumGreen.push_back(vector<long>(im.width()));
13        sumsqGreen.push_back(vector<long>(im.width()));
14
15        sumBlue.push_back(vector<long>(im.width()));
16        sumsqBlue.push_back(vector<long>(im.width()));
17    }
18
19    sumRed[0][0] = im.getPixel(0, 0)->r;
20    sumsqRed[0][0] = pow(im.getPixel(0, 0)->r, 2);
21
22    sumGreen[0][0] = im.getPixel(0, 0)->g;
23    sumsqGreen[0][0] = pow(im.getPixel(0, 0)->g, 2);
24
25    sumBlue[0][0] = im.getPixel(0, 0)->b;
26    sumsqBlue[0][0] = pow(im.getPixel(0, 0)->b, 2);
27
28    for (int i=1; i < im.height(); i++)
29    {
30        sumRed[i][0] = im.getPixel(0, i)->r + sumRed[i-1][0];
31        sumsqRed[i][0] = pow(im.getPixel(0, i)->r, 2) + sumsqRed[i-1][0];
32
33        sumGreen[i][0] = im.getPixel(0, i)->g + sumGreen[i-1][0];
34        sumsqGreen[i][0] = pow(im.getPixel(0, i)->g, 2) + sumsqGreen[i-1][0];
35
36        sumBlue[i][0] = im.getPixel(0, i)->b + sumBlue[i-1][0];
37        sumsqBlue[i][0] = pow(im.getPixel(0, i)->b, 2) + sumsqBlue[i-1][0];
38    }
39    for (int i=1; i < im.width(); i++)
40    {
41        sumRed[0][i] = im.getPixel(i, 0)->r + sumRed[0][i-1];
42        sumsqRed[0][i] = pow(im.getPixel(i, 0)->r, 2) + sumsqRed[0][i-1];
43
44        sumGreen[0][i] = im.getPixel(i, 0)->g + sumGreen[0][i-1];
45        sumsqGreen[0][i] = pow(im.getPixel(i, 0)->g, 2) + sumsqGreen[0][i-1];
46    }
```

¹完整的代码: https://github.com/chenfeng123456/CourseInOUC/tree/master/algorithm/pa3_sourcecode

```

47     sumBlue[0][i] = im.getPixel(i, 0)->b +sumBlue[0][i-1];
48     sumsqBlue[0][i] = pow(im.getPixel(i, 0)->b, 2) + sumsqBlue[0][i-1];
49 }
50 for (int i=1; i < im.height(); i++)
51 {
52     for (int j=1; j < im.width(); j++)
53     {
54         sumRed[i][j] = sumRed[i-1][j] + sumRed[i][j-1] - sumRed[i-1][j-1] + im.getPixel(
55             j, i)->r;
56         sumsqRed[i][j] = sumsqRed[i-1][j] + sumsqRed[i][j-1] -sumsqRed[i-1][j-1] + pow(
57             im.getPixel(j, i)->r, 2);
58
58         sumGreen[i][j] = sumGreen[i-1][j] + sumGreen[i][j-1] - sumGreen[i-1][j-1] + im.
59             getPixel(j, i)->g;
60         sumsqGreen[i][j] = sumsqGreen[i-1][j] + sumsqGreen[i][j-1] - sumsqGreen[i-1][j-1]
61             + pow(im.getPixel(j, i)->g, 2);
62
62         sumBlue[i][j] = sumBlue[i-1][j] + sumBlue[i][j-1] - sumBlue[i-1][j-1] + im.
63             getPixel(j, i)->b;
64         sumsqBlue[i][j] = sumsqBlue[i-1][j] + sumsqBlue[i][j-1] - sumsqBlue[i-1][j-1] +
65             pow(im.getPixel(j, i)->b, 2);
66     }
67 }
68
69 /*
70 cout << "image.width() = " << im.width() << endl;
71 cout << "image.height() = " << im.height() << endl;
72 cout << "sumRed[0].size() = " << sumRed[0].size() << endl;
73 cout << "sumRed.size() = " << sumRed.size() << endl;
74 */
75
76 cout << "im:" << endl;
77 for (int i = 0; i < 5; i++)
78 {
79     for (int j = 0; j < 5; j++)
80         cout << (long)im.getPixel(j, i)->b << " ";
81     cout << endl;
82 }
83 cout << endl;
84 cout << "sumBlue:" << endl;
85 for (int i=0; i < 5; i++)
86 {
87     for (int j=0; j < 5; j++)
88         cout << sumBlue[i][j] << " ";
89     cout << endl;
90 }
91 cout << endl;
92 cout << "sumsqBlue:" << endl;
93 for (int i=0; i < 5; i++)
94 {

```

```

92     for (int j=0; j < 5; j++)
93         cout << sumsqBlue[i][j] << " ";
94     cout << endl;
95 }
96 cout << endl;
97
98 pair<int , int> ul(2, 2);
99 pair<int , int> lr(2, 2);
100 cout << "getSum((2,2) , (2,2)):" << getSum('b', ul, lr) << endl;
101 cout << "getSum((2,2) , (2,2)):" << getSumSq('b', ul, lr) << endl;
102 cout << "getavg((2,2) , (2,2)):" << (int)getAvg(ul, lr).b << endl << endl;
103
104 }
105
106 long stats::getSum(char channel, pair<int , int> ul, pair<int , int> lr)
107 {
108     if (channel == 'r')
109     {
110         long rec1 = (ul.second == 0 ? 0 : sumRed[ul.second-1][lr.first]);
111         long rec2 = (ul.first == 0 ? 0 : sumRed[lr.second][ul.first-1]);
112         long rec3 = ((ul.second == 0 || ul.first == 0) ? 0 : sumRed[ul.second-1][ul.first-1]);
113         return sumRed[lr.second][lr.first] - rec1 - rec2 + rec3;
114     }
115     else if (channel == 'g')
116     {
117         long rec1 = (ul.second == 0 ? 0 : sumGreen[ul.second-1][lr.first]);
118         long rec2 = (ul.first == 0 ? 0 : sumGreen[lr.second][ul.first-1]);
119         long rec3 = ((ul.second == 0 || ul.first == 0) ? 0 : sumGreen[ul.second-1][ul.first-1]);
120         return sumGreen[lr.second][lr.first] - rec1 - rec2 + rec3;
121     }
122     else if (channel == 'b')
123     {
124         long rec1 = (ul.second == 0 ? 0 : sumBlue[ul.second-1][lr.first]);
125         long rec2 = (ul.first == 0 ? 0 : sumBlue[lr.second][ul.first-1]);
126         long rec3 = ((ul.second == 0 || ul.first == 0) ? 0 : sumBlue[ul.second-1][ul.first-1]);
127         return sumBlue[lr.second][lr.first] - rec1 - rec2 + rec3;
128     }
129     else
130     {
131         cerr << "Invalid channel!" << endl;
132         return -1;
133     }
134 }
135
136 long stats::getSumSq(char channel, pair<int , int> ul, pair<int , int> lr)
137 {
138     if (channel == 'r')
139     {

```

```

140     long rec1 = (ul.second == 0 ? 0 : sumsqRed[ul.second-1][lr.first]);
141     long rec2 = (ul.first == 0 ? 0 : sumsqRed[lr.second][ul.first-1]);
142     long rec3 = ((ul.second == 0 || ul.first == 0) ? 0 : sumsqRed[ul.second-1][ul.first-
143         1]);
144     return sumsqRed[lr.second][lr.first] - rec1 - rec2 + rec3;
145 }
146 else if (channel == 'g')
147 {
148     long rec1 = (ul.second == 0 ? 0 : sumsqGreen[ul.second-1][lr.first]);
149     long rec2 = (ul.first == 0 ? 0 : sumsqGreen[lr.second][ul.first-1]);
150     long rec3 = ((ul.second == 0 || ul.first == 0) ? 0 : sumsqGreen[ul.second-1][ul.
151         first-1]);
152     return sumsqGreen[lr.second][lr.first] - rec1 - rec2 + rec3;
153 }
154 else if (channel == 'b')
155 {
156     long rec1 = (ul.second == 0 ? 0 : sumsqBlue[ul.second-1][lr.first]);
157     long rec2 = (ul.first == 0 ? 0 : sumsqBlue[lr.second][ul.first-1]);
158     long rec3 = ((ul.second == 0 || ul.first == 0) ? 0 : sumsqBlue[ul.second-1][ul.first
159         -1]);
160     return sumsqBlue[lr.second][lr.first] - rec1 - rec2 + rec3;
161 }
162 else
163 {
164     cerr << "Invalid channel!" << endl;
165     return -1;
166 }
167
168 long stats::rectArea(pair<int, int> ul, pair<int, int> lr)
169 {
170     return (lr.first - ul.first + 1) * (lr.second - ul.second + 1);
171 }
172
173 long stats::getScore(pair<int, int> ul, pair<int, int> lr)
174 {
175     long R = rectArea(ul, lr);
176     long rscore = getSumSq('r', ul, lr) - pow(getSum('r', ul, lr), 2) / R;
177     long gscore = getSumSq('g', ul, lr) - pow(getSum('g', ul, lr), 2) / R;
178     long bscore = getSumSq('b', ul, lr) - pow(getSum('b', ul, lr), 2) / R;
179     return rscore + gscore + bscore;
180 }
181
182 RGBAPixel stats::getAvg(pair<int, int> ul, pair<int, int> lr)
183 {
184     long R = rectArea(ul, lr);
185     int avgR = getSum('r', ul, lr) / R;
186     int avgG = getSum('g', ul, lr) / R;
187     int avgB = getSum('b', ul, lr) / R;
188     RGBAPixel avg(avgR, avgG, avgB);
189     return avg;

```

Listing 2: twoDTree.cpp

```

1
2 /**
3 *
4 * twoDtree (pa3)
5 * slight modification of a Kd tree of dimension 2.
6 * twoDtree.cpp
7 * This file will be used for grading.
8 *
9 */
10
11 #include <cmath>
12 #include "twoDtree.h"
13 #include "cs221util/RGBAPixel.h"
14
15 /* given */
16 twoDtree::Node::Node(pair<int, int> ul, pair<int, int> lr, RGBAPixel a)
17     : upLeft(ul), lowRight(lr), avg(a), left(NULL), right(NULL), flag(-1), split_value(-1)
18 {
19
20
21
22
23 /* given */
24 twoDtree::~twoDtree(){
25     clear();
26 }
27
28 /* given */
29 twoDtree::twoDtree(const twoDtree & other) {
30     copy(other);
31 }
32
33 /* given */
34 twoDtree & twoDtree::operator=(const twoDtree & rhs){
35     if (this != &rhs) {
36         clear();
37         copy(rhs);
38     }
39     return *this;
40 }
41
42 twoDtree::twoDtree.PNG & imIn){ /* your code here */
43     im = imIn;
44     stats s(imIn);
45     height = imIn.height();
46     width = imIn.width();
47     pair<int, int> ul(0, 0);
48     pair<int, int> lr(width-1, height-1);

```

```

49     root = buildTree(s, ul, lr);
50 }
51
52
53 pair<int, int> split_h(stats &s, pair<int, int> ul, pair<int, int> lr)
54 {
55     // split the rectangle horizontally
56     long minScore = 2147483647;
57     pair<int, int> res(lr.first, ul.second);
58     pair<int, int> i(lr.first, ul.second);
59     pair<int, int> j(ul.first, ul.second+1);
60     for(; i.second < lr.second; i.second++, j.second++)
61     {
62         long score = s.getScore(ul, i);
63         score += s.getScore(j, lr);
64         if (score < minScore)
65         {
66             minScore = score;
67             res.first = i.first;
68             res.second = i.second;
69         }
70     }
71     return res;
72 }
73
74 pair<int, int> split_v(stats &s, pair<int, int> ul, pair<int, int> lr)
75 {
76     // split the rectangle vertically
77     long minScore = 2147483647;
78     pair<int, int> res(ul.first, lr.second);
79     pair<int, int> i(ul.first, lr.second);
80     pair<int, int> j(ul.first+1, ul.second);
81     for(; i.first < lr.first; i.first++, j.first++)
82     {
83         long score = s.getScore(ul, i);
84         score += s.getScore(j, lr);
85         if (score < minScore)
86         {
87             minScore = score;
88             res.first = i.first;
89             res.second = i.second;
90         }
91     }
92     return res;
93 }
94
95
96 twoDtree::Node * twoDtree::buildTree(stats &s, pair<int,int> ul, pair<int,int> lr) {
97     /* your code here */
98     RGBAPixel avg = s.getAvg(ul, lr);
99     Node *root = new Node(ul, lr, avg);

```

```

100     int curHei = lr.second - ul.second + 1;
101     int curWid = lr.first - ul.first + 1;
102     //cout << curHei << ", " << curWid << endl;
103
104     if (curHei == 1 && curWid == 1)
105     {
106         /*
107             cout << "(" << ul.first << ", " << ul.second << ")"    " << "(" << lr.first << ", " <<
108                 lr.second << ")" << endl;
109             cout << "avg = " << avg << endl;
110             cout << "(x,y)=" << *im.getPixel(ul.first , ul.second) << endl;
111             if (avg == *im.getPixel(ul.first , ul.second))
112                 cout << "True" << endl;
113             else
114             {
115                 for( int i=0; i < 100; i++)
116                 {
117                     for ( int j=0; j < 100; j++)
118                         cout << "***";
119                     cout << endl;
120                 }
121
122
123             cout << endl;
124             */
125             return root;
126     }
127     pair<int , int> split;
128     if (curHei > curWid)
129     {
130         split = split_h(s , ul , lr);
131         root->flag = 1;
132         root->split_value = split.second;
133         //cout << "root->split_value = " << root->split_value << endl;
134         root->left = buildTree(s , ul , split);
135         split.first = ul.first ;
136         split.second++;
137         root->right = buildTree(s , split , lr);
138     }
139     else
140     {
141         split = split_v(s , ul , lr);
142         root->flag = 0;
143         root->split_value = split.first ;
144         //cout << "root->split_value = " << root->split_value << endl;
145         root->left = buildTree(s , ul , split);
146         split.first++;
147         split.second = ul.second ;
148         root->right = buildTree(s , split , lr);
149     }

```

```

150
151 //cout << "(" << ul.first << ", " << ul.second << ")"    " << "(" << lr.first << ", " <<
152     lr.second << ")" << endl;
153
154     return root;
155 }
156
157 RGBAPixel* twoDtree::render_helper(int const x, int const y, Node *r)
158 {
159     if (r->left == NULL && r->right == NULL)
160         return &(r->avg);
161     RGBAPixel *p = NULL;
162     //cout << "r->avg = " << r->avg << endl;
163     //cout << "root->flag = " << r->flag << " ";
164     //cout << "r->split_value = " << r->split_value << " ";
165     //cout << "upLeft = (" << r->upLeft.first << ", " << r->upLeft.second << ") ";
166     //cout << "lowRight=( " << r->lowRight.first << ", " << r->lowRight.second << " )";
167     if (r->flag == 1)
168     {
169         //cout << " y = " << y << endl;
170         if (y <= r->split_value)
171             p = render_helper(x, y, r->left);
172         else
173             p = render_helper(x, y, r->right);
174     }
175     else
176     {
177         //cout << " x = " << x << endl;
178         if (x <= r->split_value)
179             p = render_helper(x, y, r->left);
180         else
181             p = render_helper(x, y, r->right);
182     }
183
184     if (!p)
185     {
186         /*
187         if (r->avg == *im.getPixel(x, y))
188             cout << "True" << endl;
189         else
190             cout << "False" << endl;
191         */
192         return &(r->avg);
193     }
194     //cout << *p << endl;
195     return p;
196 }
197
198
199

```

```

200 PNG twoDtree::render(){
201     /* your code here */
202     PNG image(width, height);
203     //cout << "width = " << width << "    height = " << height << endl;
204     //cout << "image.width = " << image.width() << "    image.height = " << image.height() <<
205     //endl;
206     for (int x=0; x < width; x++)
207     {
208         for (int y=0; y < height; y++)
209         {
210             RGBAPixel* p = image.getPixel(x, y);
211             cout << "x = " << x << " , y = " << y << endl;
212             RGBAPixel* other = render_helper(x, y, root);
213             cout << "*****" << endl;
214             cout << "(x,y)=" << *im.getPixel(x, y) << endl;
215             p->r = other->r; p->g = other->g; p->b = other->b; p->a = other->a;
216             cout << "r = " << (int)p->r << "    g = " << (int)p->g << "    b = " << (int)p->b
217             << endl << endl;
218             /*
219             if ((*p) == (*im.getPixel(x, y)))
220                 cout << "True" << endl;
221             else
222             {
223                 cout << "False" << endl;
224             }
225             */
226         }
227     }
228     return image;
229 }
230
231 vector<int> twoDtree::prune_helper(Node *p, double const pct, int const tol)
232 {
233     if (!p)
234     {
235         vector<int> l;
236         return l;
237     }
238     if (p->left == NULL && p->right == NULL)
239     {
240         vector<int> l;
241         l.push_back(p->avg.r + p->avg.g + p->avg.b);
242         return l;
243     }
244
245     vector<int> left = prune_helper(p->left, pct, tol);
246     vector<int> right = prune_helper(p->right, pct, tol);
247     left.insert(left.end(), right.begin(), right.end());
248
249     int threshold = left.size();

```

```

249     threshold = threshold * pct;
250     int num = 0;
251     for (int i=0; i < left.size(); i++)
252     {
253         int avgL = left[i];
254         int avgP = p->avg.r + p->avg.g + p->avg.b;
255         int diff = pow(avgL-avgP, 2);
256         if (diff <= tol)
257         {
258             num++;
259             if (num > threshold)
260             {
261                 cout << (p->left == NULL) << " , " << (p->right == NULL) << endl;
262                 remove(p);
263                 break;
264             }
265         }
266     }
267
268     return left;
269 }
270
271
272 void twoDtree::prune(double pct, int tol){
273     /* your code here */
274     prune_helper(root, pct, tol);
275 }
276
277 void twoDtree::clear() {
278     /* your code here */
279     if (root)
280     {
281         remove(root);
282         Node *old = root;
283         root = NULL;
284         delete old;
285     }
286 }
287
288 void twoDtree::remove(Node *root)
289 {
290     if (root->left)
291     {
292         if (root->left->left || root->left->right)
293             remove(root->left);
294         Node *old = root->left;
295         root->left = NULL;
296         delete old;
297     }
298
299     if (root->right)

```

```

300 {
301     if (root->right->left || root->right->right)
302         remove(root->right);
303     Node *old = root->right;
304     root->right = NULL;
305     delete old;
306 }
307
308 //cout << root << endl;
309 }
310
311 void twoDtree::copy(const twoDtree & orig){
312     /* your code here */
313     root = copy_helper(orig.root);
314     im = orig.im;
315     height = orig.height;
316     width = orig.width;
317 }
318
319
320 twoDtree::Node* twoDtree::copy_helper(const Node *r)
321 {
322     if (!r)
323         return NULL;
324
325     Node *root = new Node(r->upLeft, r->lowRight, r->avg);
326     root->flag = r->flag;
327     root->split_value = r->split_value;
328     root->left = copy_helper(r->left);
329     root->right = copy_helper(r->right);
330     root->avg = r->avg;
331     //cout << "copy " << r << endl;
332     return root;
333 }

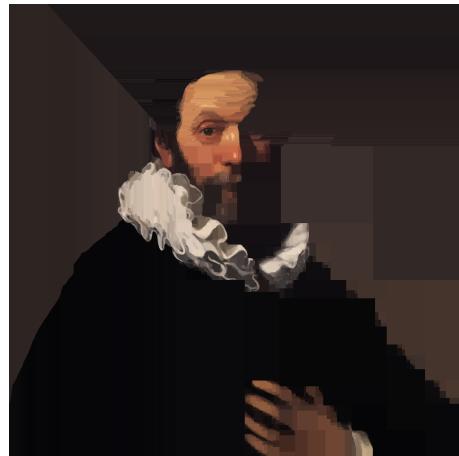
```

5 结果分析

5.1 结果展示



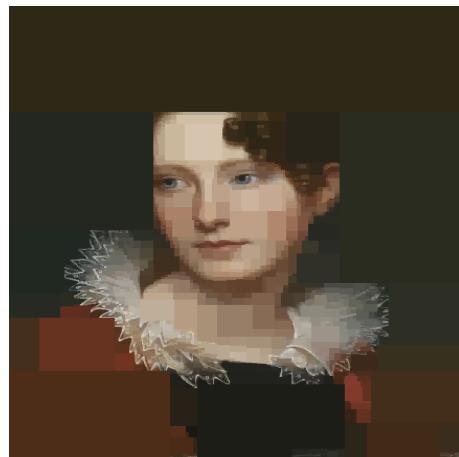
(a) 原图 1



(b) 结果 1



(c) 原图 2



(d) 结果 2



(e) 原图 3



(f) 结果 3

图 3: 测试结果



(a) 原图 4



(b) 结果 4

图 4: 测试结果

5.2 分析

可以看出，我的结果跟标准结果（报告中未给出）还是有差异的。图3(f)对背景的模糊程度比标准结果要大，而图4(b)的模糊程度比标准结果要小。

这里我所采取的分割方式能保证我们最后一定能把矩形分成一个一个像素。而采取“深度为偶数垂直分割，深度为奇数水平分割”的策略则有可能导致分到一定程度时当前矩形只剩一行（或一列）而深度却刚好是奇数（或偶数），无端增加了考虑的复杂性，而且也没有理论能表明这样分割一定能达到更好的效果，因此我觉得并没有必要局限于 $K - D$ 树的分割方式。

关于剪枝函数虽然我们用向量作为辅助使得只需要对每一个节点访问一次，但在递归返回的过程中，由于 *vector* 是在函数内部定义的，在函数结束后变量会被自动销毁，所以我们不能直接返回它的引用。这就导致了在函数返回时采用的是“值传递”，再加上需要合并向量，这就又增加了计算的成本。粗略计算仅对向量进行赋值和合并所需的时间复杂度就已经是一个与 $2^{height} \times height$ 成正比的函数了（每个节点的度数为 0 或 2），再加上删除和访问，这一块的开销其实相当大。但再仔细考虑一下， 2^{height} 代表的其实是所有叶节点的数量（其实是大于等于叶节点数量），显然 $2^{height} \times height$ 要小于 n^2 ，所以总体而言这么做还是值得的。

6 总结体会

在完成这次作业的过程中，因为代码的不规范出了很多错误。其中印象比较深的是在剪枝时我一开始是把节点作为 *vector* 的元素，但由于 *Node* 类中没有写复制构造函数（我花了好长时间才发现这一点），导致程序运行时出了很多奇怪的报错。另外就是在删除一个节点后必须把指向该节点的指针置为 *NULL*，如果不这样做的话，该指针不为空且指向一片已经被销毁的区域，会导致一些不可预计的后果。

还有一点就是有时候把变量名取得太相似会使得在打代码的过程中经常会把它们写反。比如在实现 *render_helper* 函数时，递归中我用 *r* 来指代当前节点，结果有几处把 *r* 跟整棵树的根节点 *root* 写反了，程序可以正常运行并退出（**这是最可怕的！**），但结果就是不对，导致我在这上面浪费了很多时间，*render_helper* 函数也是我用时最多的一个函数。