实验报告

实验题目	姓名	学号
基于八叉树颜色删减实验	张桓	191220156

实验目的及要求

实现真彩色到256色的颜色转换算法

提供的代码：

`main.cpp`：提供了主控函数`main`，八叉树类`octTree`和八叉树节点结构`octNode`。

代码的编译：

由于需要使用bmp的信息头和文件头结构，该结构在`windows.h`文件中，建议使用`VisualStudio`系统编译，也可以自己定义相关变量在其它编译器里编译。

在VS里编译过程：打开VS，创建新的空项目，添加`main.cpp`，编译即可，运行命令：程序名 输入文件名 输出文件名，如“程序名 input.bmp output.bmp”

本程序只是简单的demo，未考虑太多的正确性检查，故输入文件应为24位的真彩色bmp格式文件。

测试文件可以用附带的test_in.bmp文件，同时给出了几个减色样例供对比：

`test_ACDSee.bmp:ACDSee`软件的结果

`test_ps.bmp:Photoshop`的结果

`test_PaintBrush.bmp:windows`的画图软件的结果

`test_out.bmp`:上一届同学的结果

代码简介：

1、打开输入图像，读取文件头部和信息头部并填充输出图像的文件头部和信息头部；

2、依次读取输入图像的每一像素的颜色并插入八叉树中；

3、生成256色的调色板，并写出到文件中；

4、依次读取输入图像的每一个像素并在调色板中查找出与之最相近的颜色索引值。

5、输出256色图像

实验要求：

1、完成 `octTree` 类的成员函数 `insertColor`，该函数的功能是往颜色八叉树中添加一个颜色；

（a）可以先把所有像素的颜色都加入八叉树，再进行颜色删减，也可以当八叉树中的颜色数超过256色后就进行删减；

（b）删减的准则可以是最简单的删除最深层的叶子结点，或者可以尝试其它的原则；

2、完成 `octTree` 类的成员函数 `generatePalette`，该函数的功能是从颜色八叉树生成256个调色板颜色；

3、完成 `octTree` 类的析构函数，释放分配的八叉树内存空间；

4、完成函数 `selectClosestColor`，该函数的功能是从调色板中选出与给定颜色最接近的颜色；最简单的是按颜色空间的欧式距离最近来选择，可以尝试更好的方法；

实验原理

算法原理

要实现色彩转换，关键如何从真彩图像中选出256种颜色，又要使颜色的失真比较小。

八叉树颜色量化算法。算法基本思路是：将图像中使用的RGB颜色值分布到层状的八叉树中。八叉树的深度可达9层，即根节点层加上分别表示8位的R、G、B值的每一位的8层节点。较低的节点层对应于较不重要的RGB值的位（右边的位），因此，为了提高效率和节省内存，可以去掉最低部的2 ~ 3层，这样不会对结果有太大的影响。

叶节点编码存储像素的个数和R、G、B颜色分量的值；而中间的节点组成了从最顶层到叶节点的路径。这是一种高效的存储方式，既可以存储图像中出现的颜色和其出现的次数，也不会浪费内存来存储图像中不出现的颜色。算法特点：效率高，效果好。

算法步骤

颜色量化步骤：

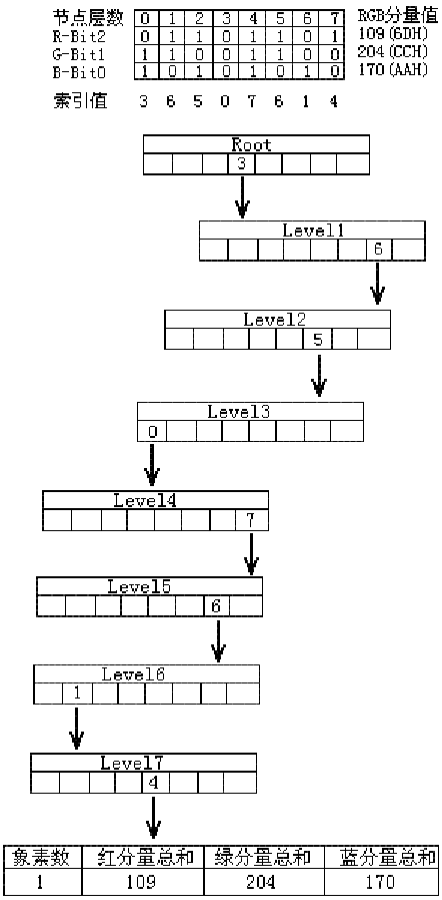
①、扫描图像的所有像素，将它们的数据累加到相应的节点中；遇到新的颜色则创建一个叶子节点，并将此像素的颜色数据存入其中。

②、如果叶子节点数目大于目标调色板所要的颜色数，就将部分叶子节点合并到父节点中，并将父节点转化为叶子节点，在其中存放颜色分量的累加值及其像素出现的次数。同时删除原来被合并的叶子节点。

③所有像素处理结束，遍历八叉树，将叶子节点的各颜色值求平均值作为节点的颜色分量值读出并存入目标调色板。

④再次遍历所有像素，通过每个像素的颜色值与调色板中选中的256色运算，求得一个最接近像素颜色值的调色板颜色，把该像素换相应的调色板颜色索引。

八叉树处理例子：节点RGB（109，204，170）如下图：



方法探索

算法的关键在于第三步的叶结点合并，超出256个颜色的叶结点很多，如何选择合并哪个？

- 基本算法

首先最简单的方式，根本不考虑顺序，就从底层向上合并，这是因为层数越深，这个结点表示的颜色越具体，代表的像素点数越少，这样被合并后受到影响的像素点数就越少。

那么就可以用**递归**来实现这个过程了，从第7层开始向上合并，合并完第7层再合并第6层，直到颜色种类不超过256个，代码如下，该函数实现合并八叉树的第`depth`层结点：

```
void reduceNode(octNode* root, int depth, int* colorCount) { //在八叉树的第depth层减色
    if (root == NULL)
        return;
    if (*colorCount <= 256)
        return;
    if (root->depth < depth) { //深度不够则递归到达
        for (int i = 0; i < 8; i++) {
            reduceNode(root->child[i], depth, colorCount);
        }
    }
    else { //深度达到了depth层 开始减色
        for (int i = 0; i < 8; i++) {
            octNode* leaf = root->child[i];
```

```
            if (leaf != NULL) {
                root->rSum += leaf->rSum;
                root->gSum += leaf->gSum;
                root->bSum += leaf->bSum;
                (*colorCount) -= 1; //颜色种类数--

                delete root->child[i];
                root->child[i] = NULL;
            }
        }
        root->isLeaf = true; //父结点成为新的叶子
        (*colorCount) += 1;
        if (*colorCount <= 256) //减色完成
            return;
    }
}
```

整体调用如下，可以看到`for`循环内部是从第7层开始向上合并的，直到颜色种数不超过256：

```
//根据现有的八叉树，选择256个颜色作为最终的调色板中的颜色
uint8 octTree::generatePalette(RGBQUAD* pal)
{
    int colorCount = getLeaftCount(root); //叶结点个数就是颜色种类数
    if (colorCount <= 256)
        return colorCount;

    for (int i = 7; i >= 0; i--) { //从深层向上减结点
        reduceNode(root, i, &colorCount);
        if (colorCount <= 256)
            break;
    }
    int index = 0;
    getPalette(root, pal, index); //下标i是传引用
    return colorCount;
}
```

- 改进

从深处开始合并是对的，但是相同深度的结点也有很多，有无先后顺序？

按照原来的思路，我们总想让**像素数量较多的颜色不被合并，而合并像素数量较少的颜色。**因为像素多的颜色被修改后在图片上的体现会非常明显，而像素少的颜色被合并则无伤大雅，我们很难分辨出来。

于是我引入了数据结构记录0~7层的结点的指针：

```
vector<vector<octNode*>> allPtr(8, vector<octNode*>()); //第0层到第7层所有结点指针
```

并且改写合并函数，同样是合并第`depth`层的结点，但是不像之前那样通过递归找到对应层，而是直接利用`allPtr`数组。

并且关键在于，合并第`depth`层结点时，首先将这一层的结点**按照代表的像素个数从小到大排序，优先合并代表像素少的结点**，代码如下：

```
void newReduceNode(int depth, int* colorCount) //在八叉树的depth层减色
{
    if (*colorCount <= 256)
        return;
    sort(allPtr[depth].begin(), allPtr[depth].end(), compare);
    int size = allPtr[depth].size();
    octNode* leaf = NULL;
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < 8; j++) {
            leaf = allPtr[depth][i]->child[j];
            if (leaf != NULL) {
                allPtr[depth][i]->rSum += leaf->rSum;
                allPtr[depth][i]->bSum += leaf->bSum;
                allPtr[depth][i]->gSum += leaf->gSum;
                --(*colorCount);
                delete leaf;
                allPtr[depth][i]->child[j] = NULL;

            }
        }
        allPtr[depth][i]->isLeaf = true;
        ++(*colorCount);
        if (*colorCount <= 256)
            return;
    }
}
```

整体调用几乎没有改变，唯一改变就是调用的删结点函数：

```
uint8 octTree::generatePalette(RGBQUAD* pal)
{
    ...
    for (int i = 7; i >= 0; i--) { //从深层向上减结点
        newReduceNode(root, i, &colorCount);
        if (colorCount <= 256)
            break;
    }
    ...
}
```

代码的详细实现见`main.cpp`

实验结果

原图

减色后

- 改进前

这时并没有考虑优先合并代表像素少的结点，效果虽然还不错但是可以看到层次感比较明显，不够自然。

- 改进后

考虑了优先合并代表像素点较少的结点，比改进前的效果好多了，与原图的差异也几乎可以忽略。

实验小结

这个算法本身非常巧妙，比如将所有真色彩都用八叉树的结点表示，并且层数越深代表的RGB位越靠右，也就是越具体，所有像素都通过树进行归类，颜色和树形结构完美契合，令人赞叹。

感觉课上讲述这部分内容不是很清楚，我在完成实验时也参考了不少资料才理解，建议可以提供一些介绍的资料。

【参考文章】：

颜色量子化（K-Means聚合算法以及八叉树的运用）

八叉树颜色压缩图解