实验报告

实验题目:　基于八叉树颜色删减实验

姓名:　蒲相彤　学号:　211220123

【实验目的及要求】

实现真彩色到 256 色的颜色转换算法

（1）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。

（2）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

可以自行添加其他测试图片文件

（3）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

（4）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）山间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；

（5）实验提交：

提交本实验报告和源文件，**均以学号为文件名。**

【实验原理】（介绍自己采用的方法）

- 首先将获得的颜色存储在八叉树中。存储原理就是如下图，将每个颜色的 rgb 转换为二进制，然后取得相应的索引。存在每一级的树，这样就刚好是八叉树。

节点层数	0	1	2	3	4	5	6	7	RGB分量值
R-Bit2	0	1	1	0	1	1	0	1	109 (6DH)
G-Bit1	1	1	0	0	1	1	0	0	204 (CCH)
B-Bit0	1	0	1	0	1	0	1	0	170 (AAH)
索引值	3	6	5	0	7	6	1	4	

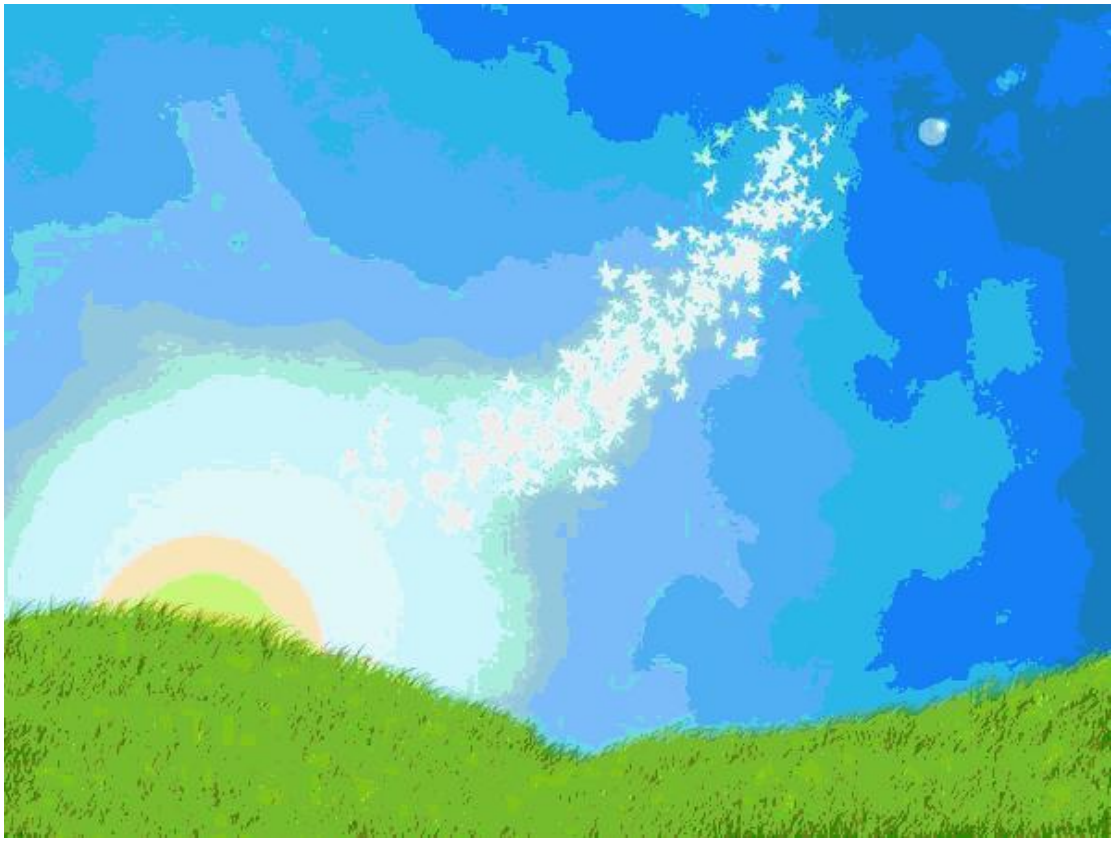
- 在插入颜色的过程中，如果超出了需要的颜色数，进行删除颜色。首先删除深度最大的的节点的叶子节点。在这个过程中，要维护一个叶子数组，这样方便访问每一个叶子来判断是否要被合并。维护 parent 指针和 childNum 来判断叶子节点的数目，depth 来维护深度。如果深度相同，则合并像素数多的那个颜色。同时合并以后要给父节点加上颜色分量。将父节点设置为叶子。
- 然后直接从维护的叶子数组获取颜色即可，其中要获得每个颜色分量的占比。
- 对于析构函数，释放根节点，如果有孩子释放八个孩子。
- 对于选色函数，直接采用三个颜色差值的平方，获得最小值得到最相近的颜色。

【实验结果】（比较转换结果）

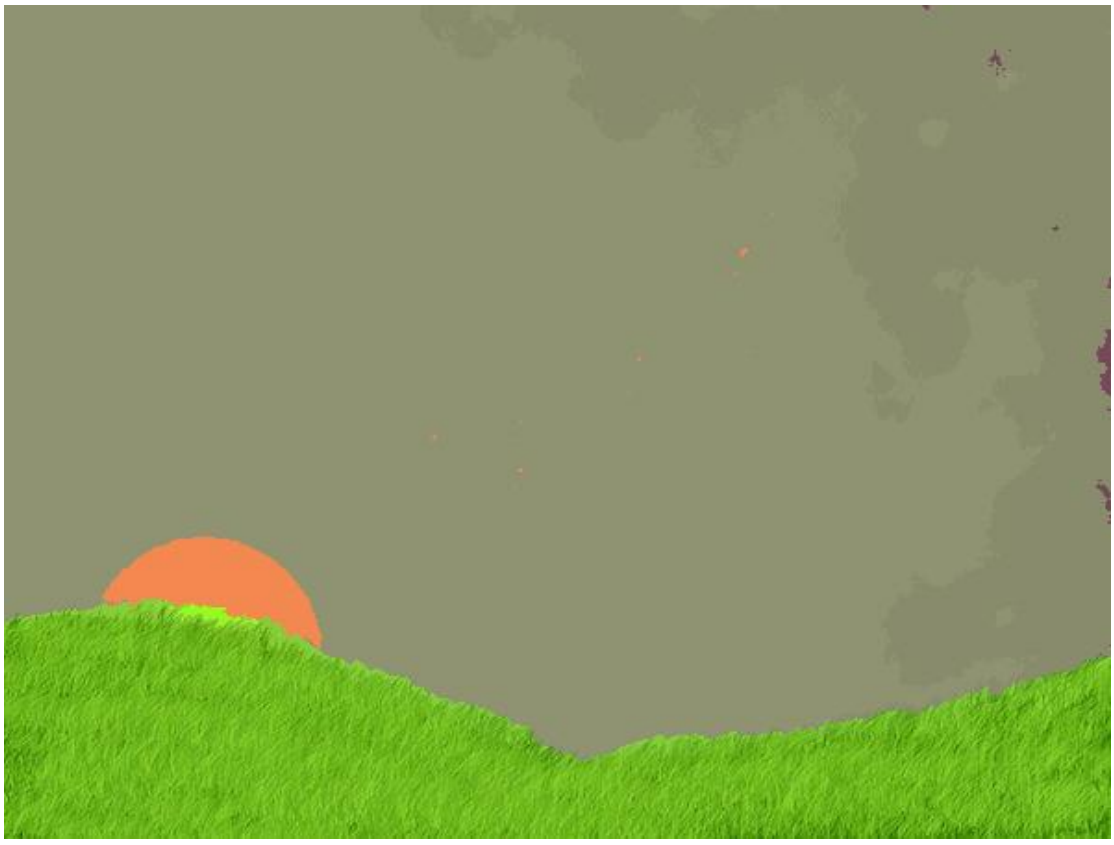
分别将删除函数放在每次插入颜色和所有颜色插入完成两种情况。得到分别如下图：

每次插入删除：

插入完删除：



另外附上两张失败作品：



【实验小结】（对本次实验的心得体会、思考和建议）

实验心得：

在本次实验中印象深刻的点有：

1. 对于位运算的处理，虽然很简单的题目但是还是需要思考来判断

```
//获取每个颜色对应的脚标。
for (int i = 0; i < 8; i++) {

    Colors[i] = (((r >> 7 - i) & 1) << 2) + (((g >> 7 - i) & 1) << 1) + ((b >> 7 - i) & 1);
}
```

2. 对于删除颜色时的维护

(1) 首先是遍历叶子数组，选取最深，像素最多的节点，如果遇到孩子节点小于 2，则向上遍历到有两个以上孩子的祖先：

```
for(int i=0;i<Leaves.size();i++)
{
    //printf("here  %d/%d---%d\n",i,len,Leaves.size());
    if(Leaves[i]!=NULL)
    {
        octNode*par=Leaves[i];
        while (par->childNum<2)
        {
            par=par->parent;
        }
        if(par->depth>maxdepth)
        {
            cur=par;
            maxdepth=par->depth;
        }
        else if(par->depth==maxdepth)
        {
            if(par->cnt < cur->cnt)
                cur=par;//change
        }
    }
}
```

(2) 对于选好的节点，将每个孩子节点设为 NULL，delete。并且将其颜色分量加到父亲节点。如果当前节点不是叶子，则一直遍历到叶子节点，删除路径的元素。最后还要维护叶子数组，将叶子从数组中删除：

```
{
    if(cur->child[i]==NULL)
        continue;

    tmp=cur->child[i];
    while (!tmp->isLeaf)
    {
        int t=0;
        while(tmp->child[t]==NULL)
            t++;
        tmp=tmp->child[t];
        delete tmp->parent;
        tmp->parent=NULL;
        cnt++;
    }
    cur->bSum+=tmp->bSum;
    cur->rSum+=tmp->rSum;
    cur->gSum+=tmp->gSum;

    for(auto it=Leaves.begin();it!=Leaves.end();it++)
    {
        if(*it==tmp)
        {
            Leaves.erase(it);
            break;
        }
    }
    delete tmp;
    tmp=NULL;
```

(3) 最后将该节点设置为叶子即可。

```
cur->isLeaf=true;
this->colors=this->colors-(cur->childNum)+1;
cur->childNum=0;
Leaves.push_back(cur);
```