Project5 实验报告

22336044 陈圳煌

1.程序功能简要说明

Project5 可以实现读取 ppm 文件，以图片的形式打开 ppm 文件，对 ppm 文件进行压缩（压缩成 txt 文件存储文件基本数据以及像素数据），并且可以读取 txt 文件并进行解码，恢复 ppm 文件格式并可以成功打开；可以实现彩色图像转变成灰度图像；实现图像的放大缩小（基于 OpenCV 库）。

2.程序运行以及部分代码

在开始代码实现之前，先安装了 Opencv 和 Notepad++以及 XnView MP 相关软件，用以查看图片和文件的像素点数据。

首先读取 ppm 文件（灰度图像为 P2 类型，像素点为单通道；彩色图像为 P3 类型，像素点为三通道）彩色图像采用特殊的三元组进行存储，灰度图像不进行二次存储操作，直接进行操作。

P3 类型的存储：

```
//P3类型的数据
struct ppmdata {
    int r;
    int g;
    int b;
    ppmdata(int x=0, int y=0, int z=0):r(x), g(y), b(z){}
};
//三元组
struct triple {
    int row;
    int col;
    ppmdata data;
    triple(int r=0, int c=0, ppmdata q=0):row(r), col(c), data(q){}
};
//存储P3类型
struct ppm {
    string type;
    int width;
    int height;
    int maxvalue;
    triple* sm;
};
```

P3 类型的读取：

```
ppm readppmP3(ifstream& file) {
    ppm tmp;
    file >> tmp.type;
    cout << "该图片类型为：" << tmp.type << endl;
    file >> tmp.width >> tmp.height;
    cout << "宽度为：" << tmp.width << " 高度为：" << tmp.height << endl;
    file >> tmp.maxvalue;
    cout << "最大像素值为：" << tmp.maxvalue << endl;
    int r, g, b;
    tmp.sm = new triple[tmp.width * tmp.height];
    for (int i = 0; i < tmp.width * tmp.height; i++) {
        int h = i / tmp.width;
        int c = i % tmp.width;
        file >> r >> g >> b;
        ppmdata q(r, g, b);
        triple s(h, c, q);
        tmp.sm[i] = s;
    }
    return tmp;
}
```

P3 类型图片读取后另存：

```
//坐标
struct index {
    int x;
    int y;
    index(int a, int b) :x(a), y(b){}
};
//把P3类型转化成存入类型
struct store {
    int count;
    int r;
    int g;
    int b;
    vector<index> in;
    store() {};
};
```

将数据转成 store 的结构体类，方便进行压缩。在压缩的时候，采用矩阵压缩的思路：如果有数据相同的元素，则只存储该类像素点出现次数 count，该类像素点数据:r g b 再分别存入每个像素点的矩阵坐标（x,y）。

P3 类型文件的压缩：

```
void zipP3(ppm& p) {
    ofstream ofs("zipP3.txt");
    if (!ofs.is_open()) {
        cout << "文件打开失败！" << endl;
    }
    ofs << p.type << endl;
    ofs << p.width << " " << p.height << endl;
    ofs << p.maxvalue << endl;
    int size = p.width * p.height;
    //int size = 100;
    vector<store> filestore;
```

```
    for (int i = 0; i < size; i++) {
        int red = p.sm[i].data.r;
        int green = p.sm[i].data.g;
        int blue = p.sm[i].data.b;
        int x = p.sm[i].row;
        int y = p.sm[i].col;
        if (filestore.size() == 0) {
            store tmp;
            tmp.r = red;
            tmp.g = green;
            tmp.b = blue;
            tmp.count = 1;
            index tmp1(x, y);
            tmp.in.push_back(tmp1);
            filestore.push_back(tmp);
            //cout << "0" << endl;
        }
        else {
            int j;
            for (j = 0; j < filestore.size(); j++) {
                if (filestore[j].r == red && filestore[j].g == green && filestore[j].b == blue) {
                    index tmp1(x, y);
                    filestore[j].count++;
                    filestore[j].in.push_back(tmp1);
                    break;
                }
            }
            if (j == filestore.size()) {
                store tmp;
                tmp.r = red;
                tmp.g = green;
                tmp.b = blue;
                tmp.count = 1;
                index tmp1(x, y);
                tmp.in.push_back(tmp1);
                filestore.push_back(tmp);
                //cout << "2" << endl;
            }
        }
    }
    for (int i = 0; i < filestore.size(); i++) {
        ofs << filestore[i].count << " " << filestore[i].r << " " << filestore[i].g << " " << filestore[i].b << endl;
        int j;
        for (j = 0; j < filestore[i].in.size()-1; j++) {
            ofs << filestore[i].in[j].x << " " << filestore[i].in[j].y << " ";
        }
        ofs << filestore[i].in[j].x << " " << filestore[i].in[j].y << endl;
    }
    ofs.close();
    cout << "图像文件压缩成功" << endl;
```

P2 类型的压缩和解压：采用 Huffman 编码原理，构建 Huffman 树，将出现的频率不同的字符编译成不定长编码，转换成二进制码存入压缩文件，解压缩时，通过读取二进制文件前段的数据，重建 Huffman 树进行解码。

（注：Huffman 编码部分为本人与 22336149 刘华壹同学的概率论与数理统计的小组作业内容，两人使用的代码来源相同）

P2 类型压缩和解码方法：

```
class HuffmanNode {
public:
    char c;
    long long int weight;
    int parent;//指向父节点的pos；
    int lchild;//指向左孩子的pos；
    int rchild;//指向右孩子的pos；
    string code;//存储编码后的code；
    HuffmanNode() :c(0), weight(0), parent(0), lchild(0), rchild(0), code("") {
    }
};
```

```
class HuffmanTree :private HuffmanNode {
private:
    //数据域
    int TreeSize;
    int LeafSize;
    HuffmanNode* Tree;
    map<char, string> CharToCode;
    map<string, char> CodeToChar;

    //方法域；
    void Sort(HuffmanNode* Tree, int n, int& s1, int& s2) {//搜索 [1,n-1]个元素，找出最小的两个；//注意的最小的是没有父节点的最小的两个；
        multimap<long long int, int> tem;//实现了自动排序；//weight可能相同，要multimap
        for (int i = 1; i < n; i++) {
            if (Tree[i].parent == 0) {
                tem.insert(make_pair(Tree[i].weight, i));
            }
        }
        auto p = tem.begin();
        s1 = p->second;//应该对吧。
        p++;
        s2 = p->second;
    }
```

```
    }
    void Writeinfile(string input, string filename) {
        ofstream output(filename, ifstream::binary);
        if (!output) cout << "wrong" << endl;
        //将input翻译为01字符串；
        string newstr = "";
        for (int i = 0; i < input.size(); i++) {
            for (auto j = CharToCode.begin(); j != CharToCode.end(); j++) {
                if ((*j).first == input[i]) {
                    newstr += (*j).second;
                }
            }
        }
        //将ChartoCode写入文件;要顺便存入长度等信息；
        //写入input长度；
        int countload = newstr.length() / 8 + 1;
        char lenoffinal = newstr.length() % 8;
        char tem = (int)CodeToChar.size();
        output << countload;
        output << lenoffinal;
        output << tem;
        for (auto it = CodeToChar.begin(); it != CodeToChar.end(); it++) {
            string s = (*it).first;
            char slen = s.size();
            output << slen;
            for (int i = 0; i < slen; i++) {
                output << s[i];
            }
            output << (*it).second;
        }
```

```
        //将01字符串变成八位二进制码;
        for (int i = 0; i < newstr.size(); i += 8) {
            string byte = newstr.substr(i, 8); // 按八位一组分割
            unsigned char x = static_cast<char>(bitset<8>(byte).to_ulong()); // 将八位二进制转换为字符
            output << x;
        }
        //cout << "字符编码成功" << endl;
        output.close();
    }
    }
public:
    HuffmanTree() :TreeSize(0), Tree(nullptr) {}
    void Compress(string path, string filename) {
        //打开path，把path文件中的内容化为string放在input里面;
        ifstream infile(path, ifstream::binary);
        if (!infile) {
            cout << "wrong" << endl;
        }
        string input((istreambuf_iterator<char>(infile)), istreambuf_iterator<char>());
        //统计频率并生成map;
        map<char, long long int> m;
        for (int i = 0; i < input.size(); i++) {
            if (m.find(input[i]) != m.end()) {
                m[input[i]] += 1;
            }
            else {
                m[input[i]] = 1;
            }
        }
        //将m的信息存储进Tree中，实现初始化LeafSize个Node;
        LeafSize = m.size();
        TreeSize = 2 * m.size() - 1;
        Tree = new HuffmanNode[TreeSize + 1];//0号Node不使用。最终树有2*LeafSize-1个节点
        int k = 1;//0号节点不使用。
        for (auto it = m.begin(); it != m.end(); it++, k++) {//初始化节点;
            Tree[k].weight = it->second;
            Tree[k].c = it->first;
        }

    //初始化节点后，进行建树；对于Huffman树，前面LeafSize个原始节点成为叶子节点，LeafSize-1个内部节点（用于将原始LeafSize个节点联系起来）;
    for (int i = LeafSize + 1; i < 2 * LeafSize; i++) {//进行n-1次融合；新节点放在i处；融合后，Tree【2*LeafSize-1】为根节点了。
        int s1, s2;
        Sort(Tree, i, s1, s2);//此函数选出weight最小的两个pos，s1为最小的，s2为次最小的；//注意的最小的是没有父节点的最小的两个；weight小的
        //将两个节点合为一个节点;
        Tree[i].parent = 0;//新节点无父亲节点;
        Tree[i].rchild = s1;//右节点为最小
        Tree[i].lchild = s2;//左节点为次小;
        Tree[s1].parent = i;
        Tree[s2].parent = i;//孩子节点父亲节点修改为i;
        Tree[i].weight = Tree[s1].weight + Tree[s2].weight;//新节点的weight设置为二者之和;
    }
    //遍历树进行编码；由huffman树的结构可知，编出来的码为前缀码;
    //从根节点Tree【2*LeafSize-1】处进行遍历；向左编为1，向着右编为0；实际上为层次遍历;
    queue<int> pos;//层次遍历需要的数组；一开始压入根节点;
    pos.push(2 * LeafSize - 1);
    while (pos.size() != 0) {
        int tem = pos.front();
        pos.pop();
        if (Tree[tem].lchild == 0 && Tree[tem].rchild == 0) {//到叶子节点了；不操作;
            CharToCode.insert(make_pair(Tree[tem].c, Tree[tem].code));//将叶子节点的信息存储到两个map中，方便译码;
            CodeToChar.insert(make_pair(Tree[tem].code, Tree[tem].c));
        }
        else {
            Tree[Tree[tem].lchild].code = Tree[tem].code + "1";
            Tree[Tree[tem].rchild].code = Tree[tem].code + "0";
            pos.push(Tree[tem].lchild);
            pos.push(Tree[tem].rchild);
        }
    }
    //以上操作为生成树；生成树完毕，写入filename;
    Writeinfile(input, filename);
}
```

```
}
void DeCompress(string path, string filename) {
    ifstream from(path);
    ofstream to(filename);
    if (!from) cout << "wrong" << endl;
    if (!to) cout << "wrong" << endl;
    map<string, char> m;
    int countload;
    char lenoffinal;
    char msize;
    from >> countload;
    from.read(&lenoffinal, sizeof(char));
    from.read(&msize, sizeof(char));
    for (int i = 0; i < msize; i++) {
        string s = "";
        char c;
        char slen;
        from.read(&slen, sizeof(char));
        for (int i = 0; i < slen; i++) {
            char tem;
            from.read(&tem, sizeof(char));
            s += {tem};
        }
        from.read(&c, sizeof(char));
        m.insert(make_pair(s, c));
    }
    string key = "";
    char tem = 0;
    int printlen = 0;
    int printlen = 0;
    for (int i = 0; i < countload; i++) {
        from.read((char*)&tem, sizeof(char));
        int j = 0;
        if (i == countload - 1) {
            for (int k = 0; k < 8 - lenoffinal; k++) {
                j++;
                tem = tem * 2;
            }
        }
        for (; j < 8; j++) {
            if (tem >= 0) {
                key += "0";
            }
            else {
                key += "1";
            }
            if (m.find(key) != m.end()) {
                to.write(&m[key], sizeof(char));
                printlen++;
                key = "";
            }
            tem = tem * 2;
        }
    }
}
};
```

```
class TextCompresser {
public:
    string path;
    string filename;
    TextCompresser(string path, string filename) :path(path), filename(filename) {
    }
    void HuffmanCompress() {
        HuffmanTree my;
        my.Compress(path, filename);
    }
    void HuffmanDeCompress() {
        HuffmanTree my;
        my.DeCompress(path, filename);
    }
};
//P3类型的数据
```

转换成灰度图像:

```
void changegrey(ifstream& file) {
    ppm tmp;
    tmp = readppmP3(file);
    if (tmp.type == "P2") {
        cout << "你打开的不是彩色文件！" << endl;
        return;
    }
    int size = tmp.width * tmp.height;
    for (int i = 0; i < size; i++) {
        int grey = (tmp.sm[i].data.r + tmp.sm[i].data.g + tmp.sm[i].data.b) / 3;
        tmp.sm[i].data.r = grey;
        tmp.sm[i].data.g = grey;
        tmp.sm[i].data.b = grey;
    }
    ofstream ofs("grey.ppm");
    tmp.type = "P2";
    ofs << tmp.type << endl;
    ofs << tmp.width << " " << tmp.height << endl;
    ofs << tmp.maxvalue << endl;
    for (int i = 0; i < size; i++) {
        ofs << tmp.sm[i].data.r << " ";
    }
    ofs.close();
}
```

将彩色图像三个通道的值取平均，并改成单通道存储。

图像缩放：使用 Opencv 的库函数实现

```
else if (op2 == 4) {
    cout << " a.放大图像（2倍）" << endl;
    cout << " b.缩小图像（0.5倍）" << endl;
    image = imread(filename);
    char op4;
    cin >> op4;
    if (op4 == 'a') {
        Mat largeimage;
        Size largesize(0, 0);
        double scale = 2.0;
        cv::resize(image, largeimage, largesize, scale, scale, cv::INTER_LINEAR);
        cv::imshow("Large Image", largeimage);
        cv::waitKey(0);
    }
    else if (op4 == 'b') {
        Mat smallimage;
        Size smallsize(0, 0);
        double scale = 0.5;
        cv::resize(image, smallimage, smallsize, scale, scale, cv::INTER_LINEAR);
        cv::imshow("Small Image", smallimage);
        cv::waitKey(0);
    }
```

P3 文件的解压：

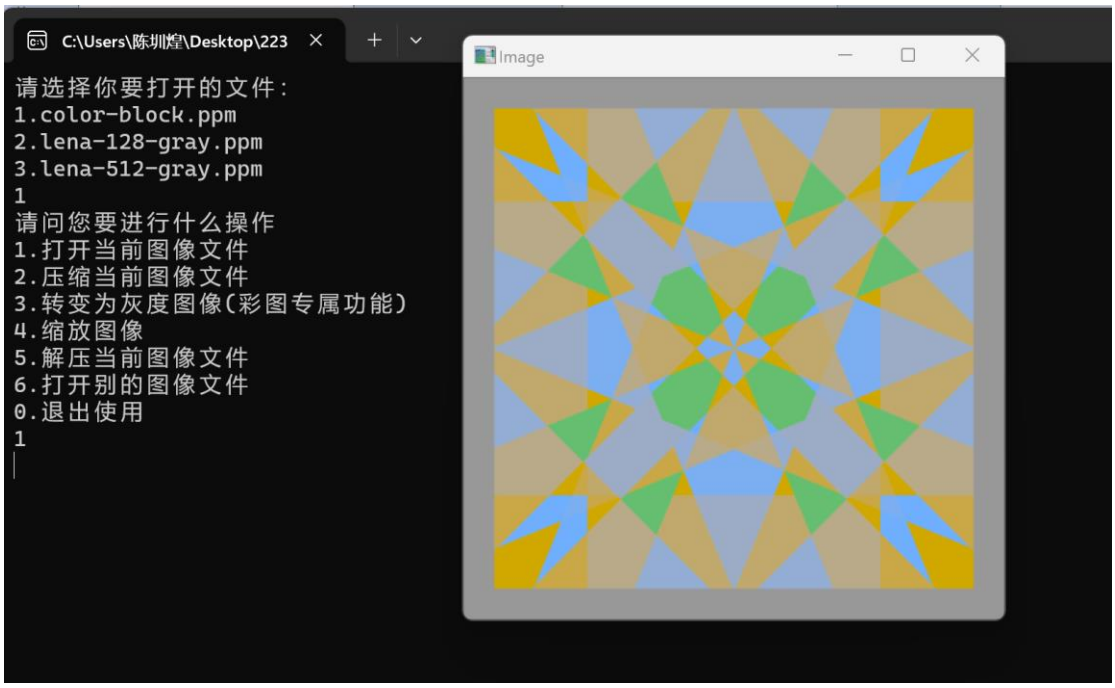
```
}
//解压P3类型的文件
void unzipP3(ifstream &ifs) {
    ppm p;
    ifs >> p.type;
    ifs >> p.width >> p.height;
    ifs >> p.maxvalue;
    vector<store> fileps;
    store ps;
    while (ifs >> ps.count >> ps.r >> ps.g >> ps.b) {
        for (int i = 0; i < ps.count; i++) {
            int a, b;
            ifs >> a >> b;
            index tmp(a, b);
            ps.in.push_back(tmp);
        }
        fileps.push_back(ps);
        ps.in.clear();
    }
    p.sm = new triple[p.width * p.height];
    int ind = 0;
    //cout << fileps.size() << endl;
    for (int i = 0; i < fileps.size(); i++) {
        int red = fileps[i].r;
        int green = fileps[i].g;
        int blue = fileps[i].b;
        ppmdata tmp2(red, green, blue);
        for (int j = 0; j < fileps[i].in.size(); j++) {
            triple s(fileps[i].in[j].x, fileps[i].in[j].y, tmp2);
            p.sm[ind] = s;
            ind++;
        }
    }
    ofstream ofs("unzipP3.ppm");
```

```
    ofstream ofs("unzipP3.ppm");
    ofs << p.type << endl;
    ofs << p.width << " " << p.height << endl;
    ofs << p.maxvalue << endl;
    int size = p.width * p.height;
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if ((p.sm[j].row * p.width) + p.sm[j].col == i) {
                ofs << p.sm[j].data.r << " " << p.sm[j].data.g << " " << p.sm[j].data.b << " ";
            }
        }
    }
    ofs.close();
}
```

将存储三通道数据按照顺序重新存回 ppm 文件。

3.测试案例

打开文件:



压缩文件

```
1
请问您要进行什么操作
1.打开当前图像文件
2.压缩当前图像文件
3.转变为灰度图像(彩图专属功能)
4.缩放图像
5.解压当前图像文件
6.打开别的图像文件
0.退出使用
2
该图片类型为：P3
宽度为：390 高度为：390
最大像素值为：255
图像文件压缩成功
```

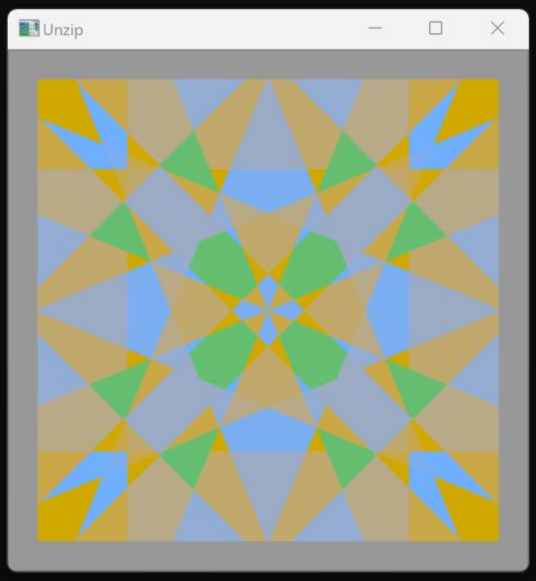
color-block.png	2012/3/8 23:36	PNG 文件	62 KB
color-block.ppm	2022/10/7 16:14	PPM 文件	1,754 KB
grey.ppm	2023/11/24 17:03	PPM 文件	595 KB
lena-128-gray.png	2022/10/7 16:18	PNG 文件	11 KB
lena-128-gray.ppm	2022/10/7 16:05	PPM 文件	60 KB
lena-512-gray.png	2022/10/7 16:18	PNG 文件	220 KB
lena-512-gray.ppm	2022/10/7 16:08	PPM 文件	945 KB
project5.cpp	2023/11/26 23:14	C++ Source File	15 KB
Project5.sln	2023/11/21 10:33	Visual Studio Soluti...	2 KB
Project5.vcxproj	2023/11/23 17:40	VCXPROJ 文件	7 KB
Project5.vcxproj.filters	2023/11/23 17:40	VC++ Project Filter...	1 KB
Project5.vcxproj.user	2023/11/21 10:33	Per-User Project O...	1 KB
unzipP2.ppm	2023/11/26 23:14	PPM 文件	60 KB
unzipP3.ppm	2023/11/23 22:35	PPM 文件	1,754 KB
zipP2.txt	2023/11/26 23:14	文本文档	24 KB
zipP3.txt	2023/11/26 23:47	文本文档	1,141 KB

项目 2.82 MB

由源文件和压缩文件的大小比值可以得出压缩率大概为 1141/1754=65%（猜想：如果改成线性压缩，将存储矩阵当成线性表可以进一步提高压缩率）

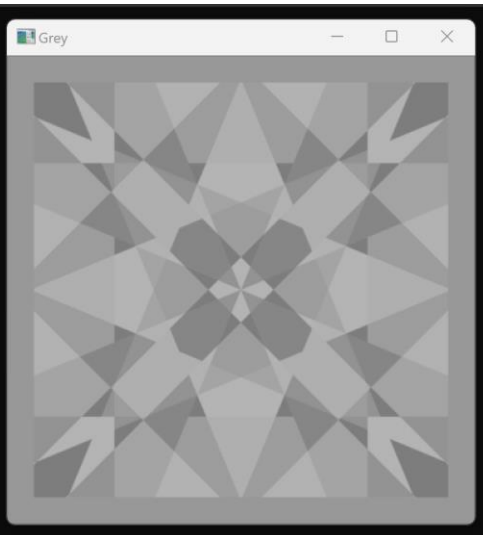
解压文件:

```
1
请问您要进行什么操作
1.打开当前图像文件
2.压缩当前图像文件
3.转变为灰度图像(彩图专属功能)
4.缩放图像
5.解压当前图像文件
6.打开别的图像文件
0.退出使用
2
该图片类型为: P3
宽度为: 390 高度为: 390
最大像素值为: 255
图像文件压缩成功
请问您要进行什么操作
1.打开当前图像文件
2.压缩当前图像文件
3.转变为灰度图像(彩图专属功能)
4.缩放图像
5.解压当前图像文件
6.打开别的图像文件
0.退出使用
5
 解压成功!
 是否打开解压后的文件
 YES or NO
YES
```

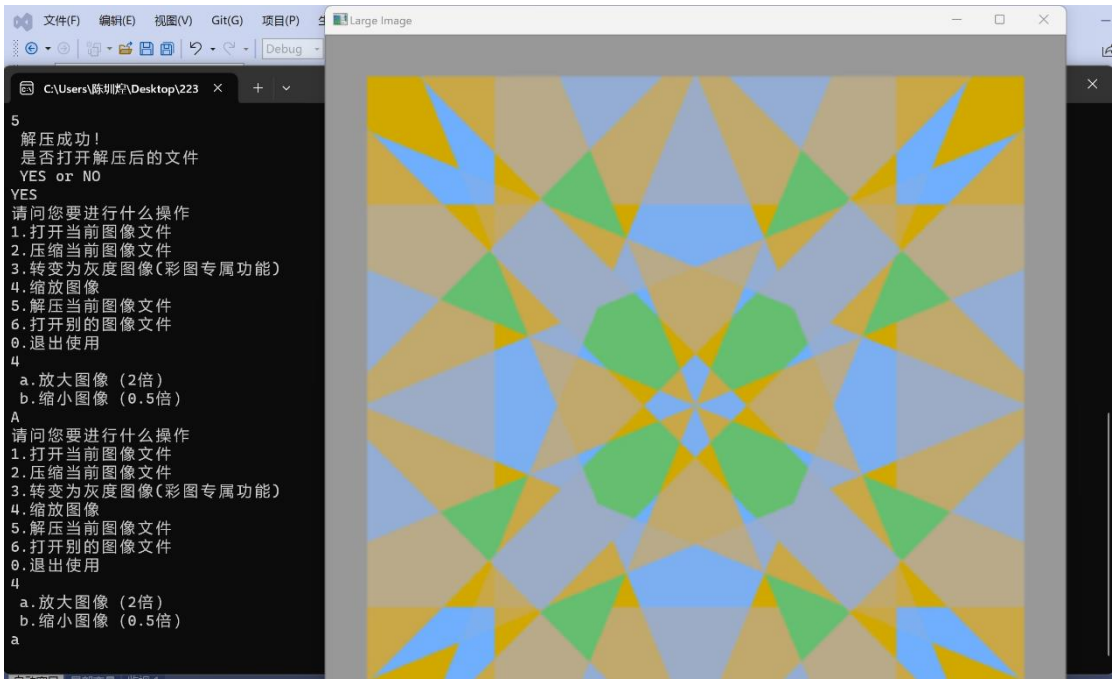


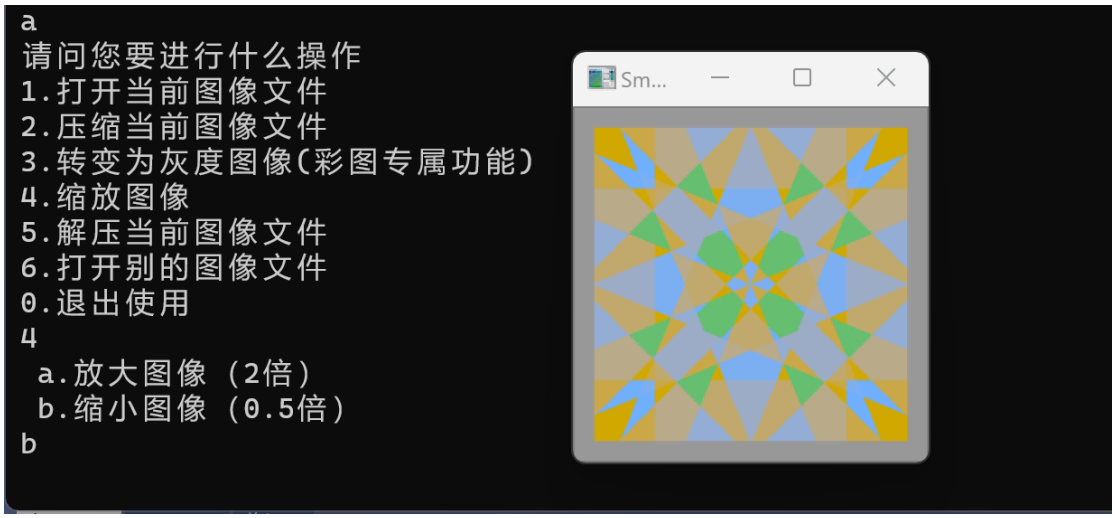
转换成灰度图像

```
 b.缩小图像 (0.5倍)
a
请问您要进行什么操作
1.打开当前图像文件
2.压缩当前图像文件
3.转变为灰度图像(彩图专属功能)
4.缩放图像
5.解压当前图像文件
6.打开别的图像文件
0.退出使用
4
 a.放大图像 (2倍)
 b.缩小图像 (0.5倍)
b
请问您要进行什么操作
1.打开当前图像文件
2.压缩当前图像文件
3.转变为灰度图像(彩图专属功能)
4.缩放图像
5.解压当前图像文件
6.打开别的图像文件
0.退出使用
3
该图片类型为: P3
宽度为: 390 高度为: 390
最大像素值为: 255
 是否打开灰度图像:
 YES or NO
YES
```



图像缩放:





4.程序运行方式

首先选择打开的图片文件（彩色或者灰度）

```
    string filename;
    cout << "请选择你要打开的文件: " << endl;
    cout << "1.color-block.ppm" << endl;
    cout << "2.lena-128-gray.ppm" << endl;
    cout << "3.lena-512-gray.ppm" << endl;
    int op;
```

之后可以选择对图片的操作:

```
    }
    cout << "请问您要进行什么操作" << endl;
    cout << "1.打开当前图像文件" << endl;
    cout << "2.压缩当前图像文件" << endl;
    cout << "3.转变为灰度图像(彩图专属功能)" << endl;
    cout << "4.缩放图像" << endl;
    cout << "5.解压当前图像文件" << endl;
    cout << "6.打开别的图像文件" << endl;
    cout << "0.退出使用" << endl;
```

选择解压，解压成功后可以选择是否打开图片：

```
        cout << " 解压成功！" << endl;
        cout << " 是否打开解压后的文件" << endl;
        cout << " YES or NO" << endl;
        string op6;
        cin >> op6;
        if (op6 == "YES") {
            image = imread("unzipP2.ppm");
            imshow("Unzip", image);
            waitKey(0);
        }
    }
```