Project5 图像处理

21307077

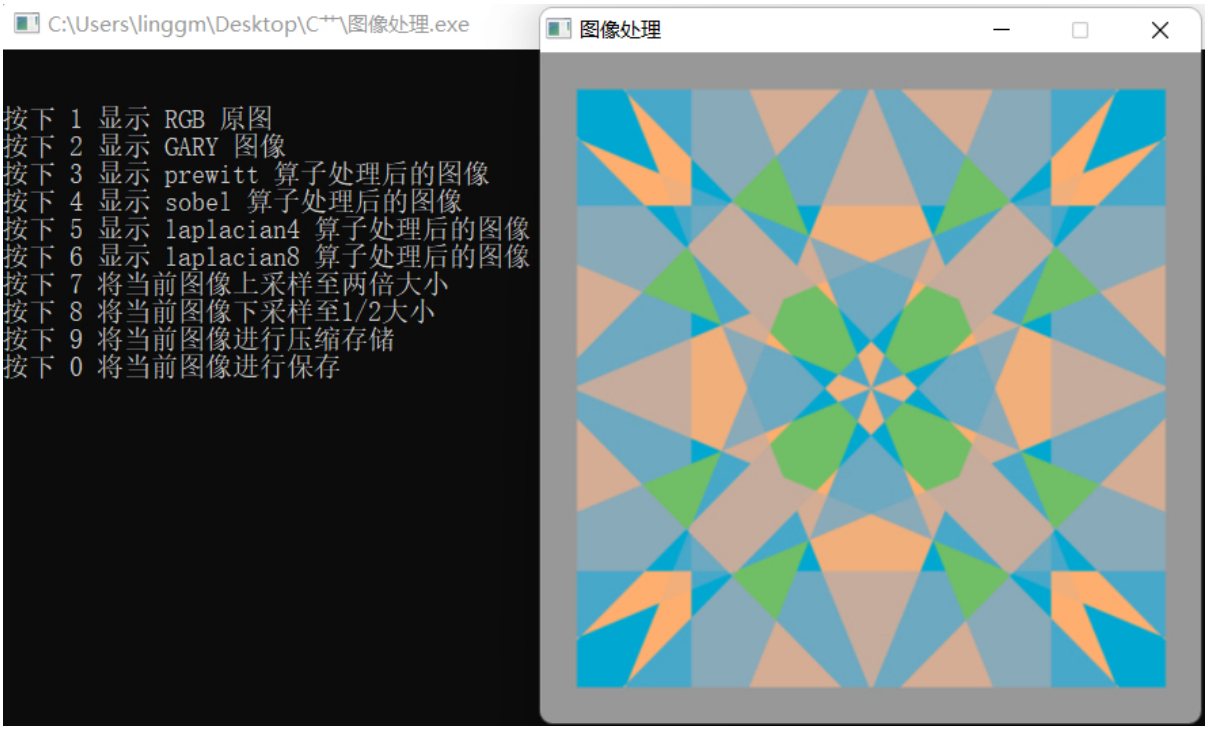
凌国明

程序功能说明

1. 读入 jpg，png 等格式的图像，并在屏幕上显示出来。
2. 对图像进行灰度化，并展示结果。
3. 对图像进行压缩存储。用的是基于概率的压缩算法——哈夫曼编码压缩。对压缩后的编码进行存储。
4. 对压缩后的编码进行译码，得出原图像的 RGB 矩阵，并进行图像的保存。
5. 对图像进行上采样和下采样，使用双线性插值的方法。
6. 将 $prewitt$，$sobel$，$laplacian$ 等算子作用在图像上，并显示结果图像。
7. 将目前屏幕上的图像进行保存。

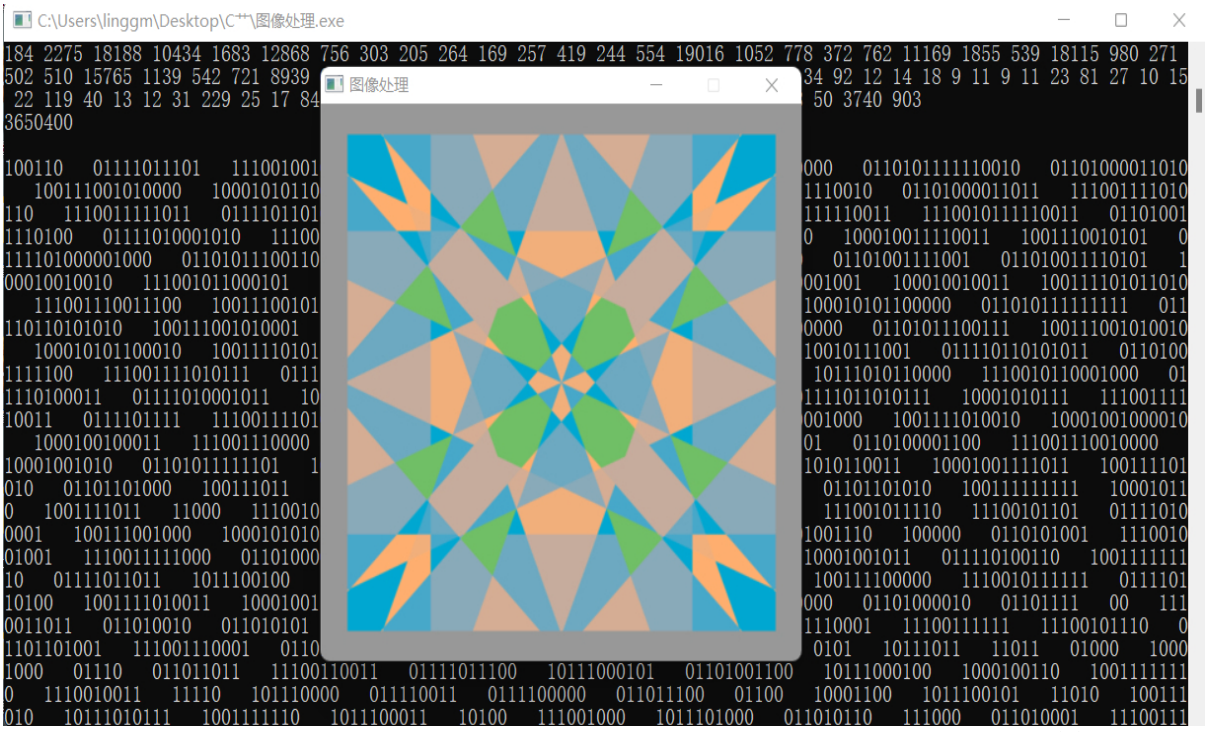
程序运行展示

输入图像



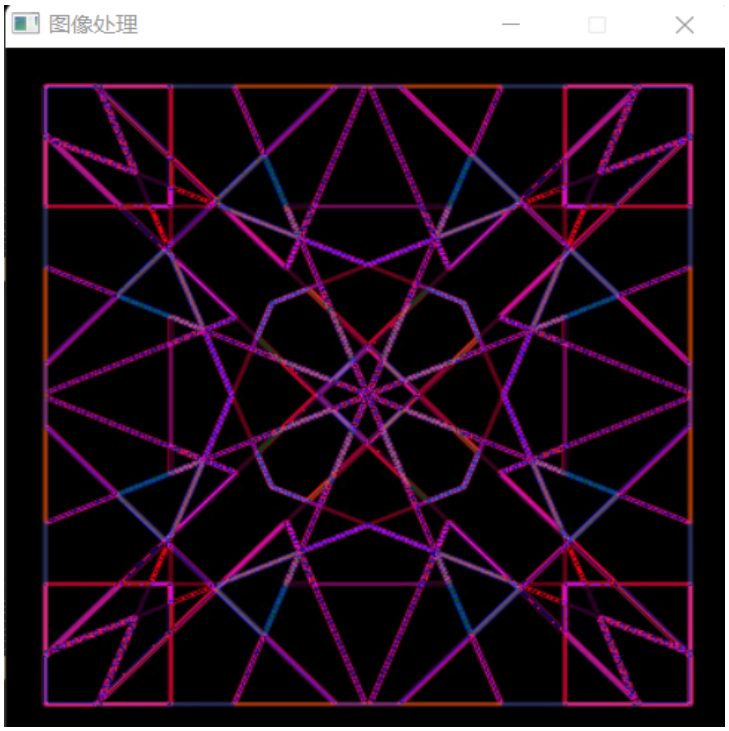
灰度化

压缩存储

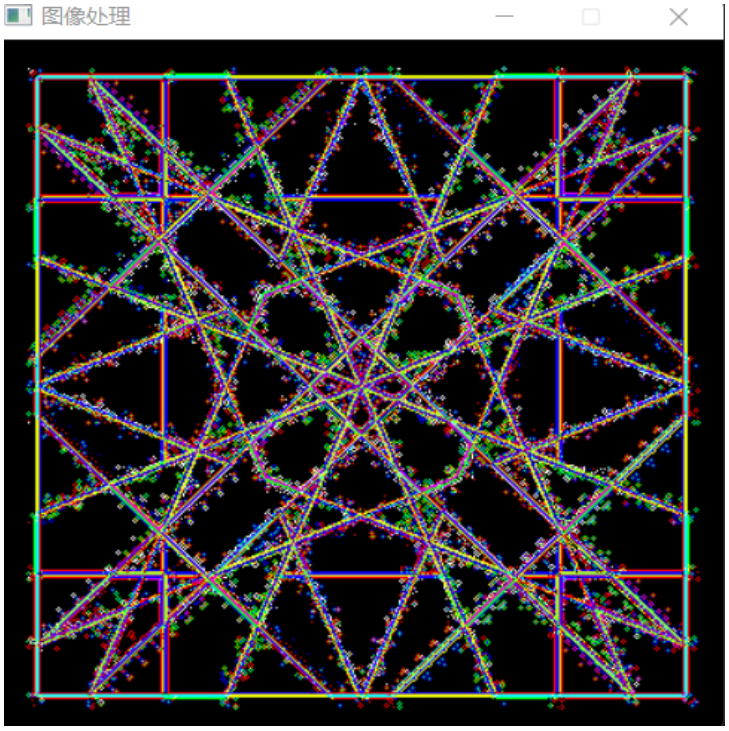


原图像需要 $3650400\ bit$ 的存储空间，压缩后仅需 $2323311\ bit$，是原来的 63.65%

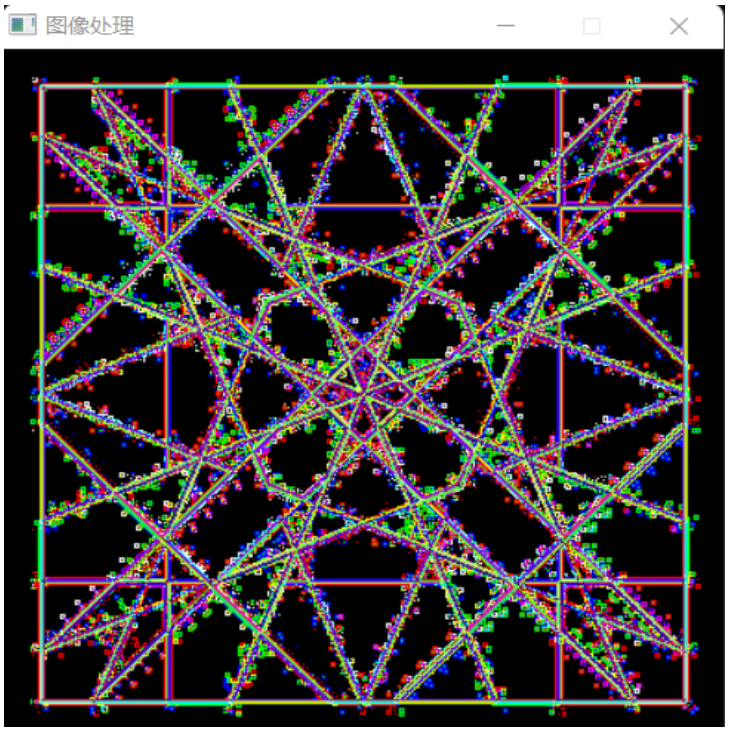
prewitt算子



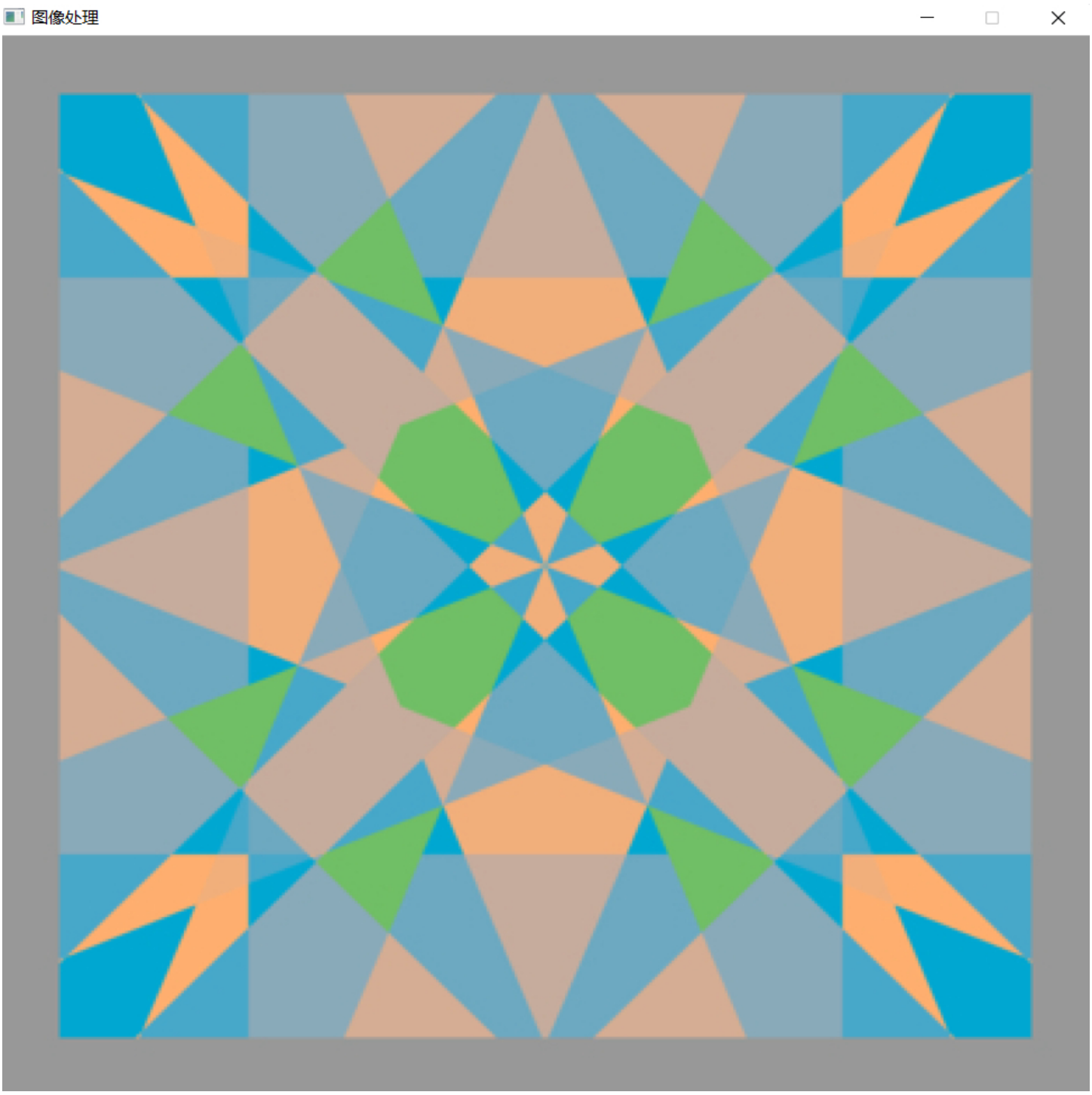
sobel算子



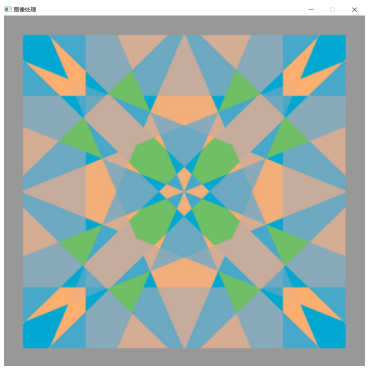
laplacian算子



上采样



下采样



部分关键代码及其说明

哈夫曼树部分代码

```
template<typename T>
class List{
public:
        Node<T> *head;
        int num;
        List(){
                head = new Node<T>(nullptr);
                num = 0;
        }
        void insert(T data){
                Node<T> *tmp = head;
                while(tmp->next != nullptr && data->data > tmp->next->data->data){
                        tmp = tmp->next;
                }
                Node<T> *tmp2 = new Node<T>(data);
                tmp2->next = tmp->next;
                tmp->next = tmp2;
                num++;
//              disp();
                return;
        }
        bool pop(){
                if(num == 0)
                        return false;
                head = head->next;
                num--;
                return true;
        }
        void merge(){

                while(num >= 2){
                        Node2<int> *tmp1 = head->next->data;
                        Node2<int> *tmp2 = head->next->next->data;
                        int res = tmp1->data + tmp2->data;
                        pop();
                        pop();
                        Node2<int> *node2 = new Node2<int>(res, tmp1, tmp2);
                        insert(node2);
                }
        }

        vector<vector<bool>> init(int n, vector<int> weight){
                for(int i = 0; i < n; i++){
                        Node2<int> *node2 = new Node2<int>(weight[i]);
                        this->insert(node2);
                }
                this->merge();
                BinaryTree<int> bt((this->head)->next->data);
                vector<vector<bool>> ans = bt.code(weight);
                for(int i = 0; i < ans.size(); i++){
```

```
                        for(int j = 0; j < ans[i].size(); j++){
                                cout << ans[i][j];
                        }
                        cout << "   ";
                }
                cout << endl << bt.SumWeight();
                cout << endl;
                return ans;
        }
};
```

哈夫曼树编码

```
vector<vector<bool>> code(vector<int> weight){
    queue<Node2<T>*> que;
    vector<vector<bool>> ans;
    vector<int> which;
    queue<vector<bool>> code_que;
    Node2<T> *tmp;
    int child = 0, mem1 = 1, mem2 = 0, ceng = 0;
    que.push(root);
    code_que.push(vector<bool>() );
    while(!que.empty()){
        vector<bool> tmp_code = code_que.front();
        child = 0;
        tmp = que.front();
        if(tmp->left != nullptr){
            que.push(tmp->left);
            tmp_code.push_back(false);
            code_que.push(tmp_code);
            tmp_code = code_que.front();
            child++;
            mem2++;
        }
        if(tmp->right != nullptr){
            que.push(tmp->right);
            tmp_code.push_back(true);
            code_que.push(tmp_code);
            child++;
            mem2++;
        }
//                      cout << tmp->data << ' ';
        if(child == 0){
            ans.push_back(tmp_code);
            for(int i = 0; i < weight.size(); i++){
                if(weight[i] == tmp->data){
                    for(int j = 0; j < which.size(); j++){
                        if(which[j] == i)
                            break;
                        else if(j == which.size()-1){
                            which.push_back(i);
                        }
                    }
                    if(which.size() == 0){
                        which.push_back(i);
                    }
                }
            }
        }
        if(--mem1 == 0){
            mem1 = mem2;
            mem2 = 0;
            ceng++;
```

```
        }
        que.pop();
        code_que.pop();
    }
    sortt(ans, which);
    return ans;
}
```

下采样

```
void down_scale(int h, int w){
    initgraph(width/2, height/2, SHOWCONSOLE);
    IMAGE img2(width/2, height/2);
    CONSOLE_height = height/2;
    CONSOLE_width = width/2;
    putimage(0, 0, &img2); //显示图片
    DWORD *pMem = GetImageBuffer();
    for(int i = 0; i < height/2; i++){
        for(int j = 0; j < width/2; j++){
            *pMem = BGR(*pMem);
            // 获取RGB的三个分量值
            int tmp1 = (Red[2*i][2*j] + Red[2*i+1][2*j] + Red[2*i][2*j+1] + Red[2*i+1][2*j+1]) /
            int tmp2 = (Green[2*i][2*j] + Green[2*i+1][2*j] + Green[2*i][2*j+1] + Green[2*i+1][2
            int tmp3 = (Blue[2*i][2*j] + Blue[2*i+1][2*j] + Blue[2*i][2*j+1] + Blue[2*i+1][2*j+1
//                                  Gray[i][j] = 0.3*R + 0.59*G + 0.11*B;
            *pMem =  RGB(tmp1, tmp2, tmp3);
            pMem++;
        }
    }
}
```

$mode$ 为 0 时匹配大小写，否则不匹配。通过二维的 dp 数组构造**有限状态自动机**，以实现查找模式串的状态转移。

程序运行方式简要说明

1. 通过 $EasyX$ 库开发图形化交互界面，图像的展示，操作的交互。
2. 按下键盘，从各个功能中切换。如点击 “2” 时，可以对保存的图像进行灰度化，点击 “0” 时，可以保存当前绘图区的图像。
3. 点击 “9” 时，对当前图像进行哈夫曼编码，进行压缩存储。
4. 点击 “3-6” 时，将各种算子作用在图像上，展示结果。
5. 点击 “7-8” 时，对图像进行上/下采样，并进行展示。