同濟大學
TONGJI UNIVERSITY

高级语言程序设计大作业

装

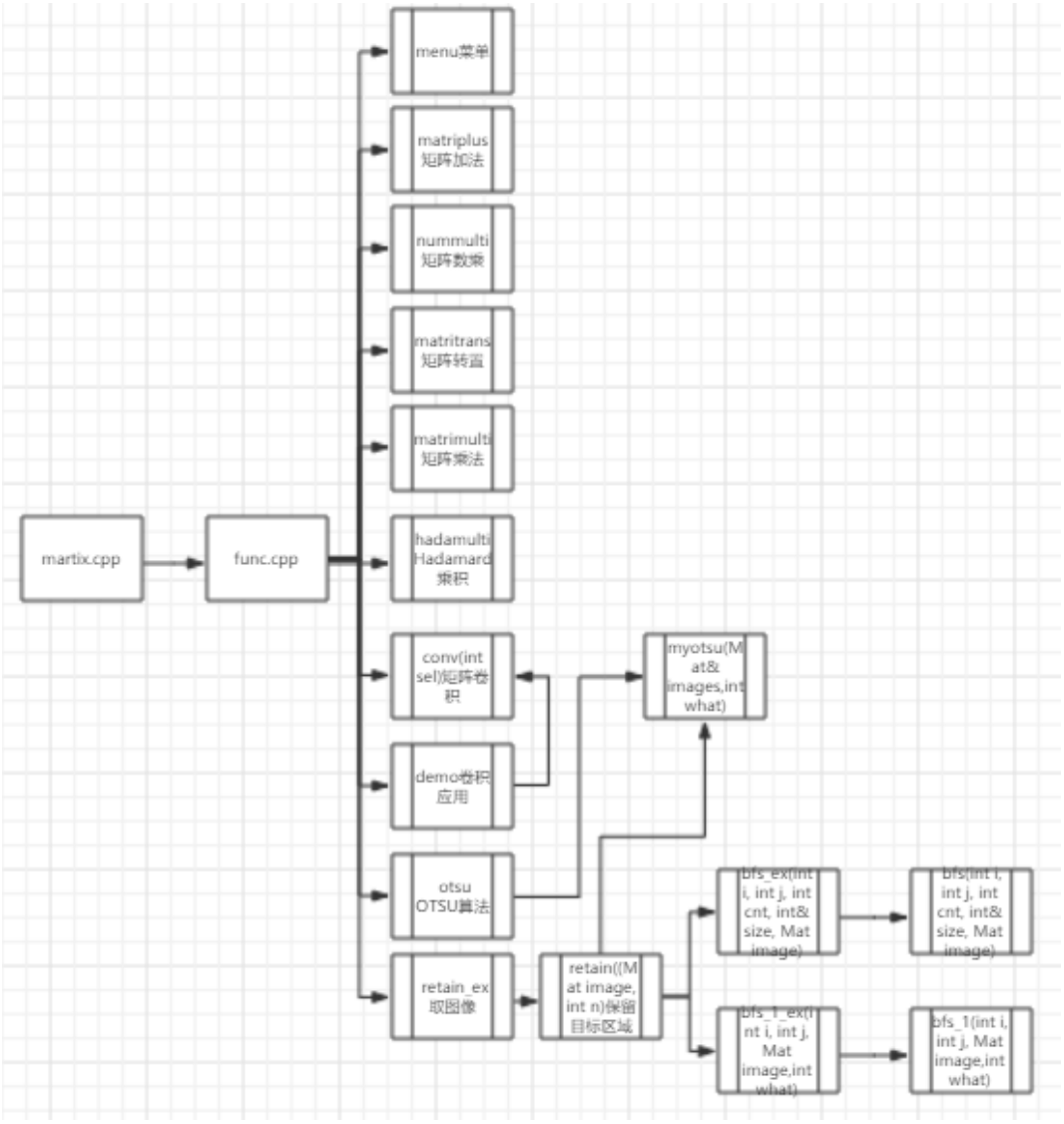
订

线

设计思路与功能描述

一、设计思路：

各函数之间的调用关系：



二、功能描述：

1. menu()：菜单

直接输出即可，注意格式。

2. matriplus()：矩阵加法

输入两矩阵的行数、列数，强制要求相同。第一个矩阵把数据放入 a[],第二个矩阵把数据放入 b[],将 a[]和 b[]的对应项相加，结果存放在 a[]，输出时注意换行。之后把数组清零。

3. nummulti()：矩阵数乘

输入矩阵的行数、列数，要乘的数 x。矩阵把数据放入 a[],将 a[]每一项和 x 相乘，结果存放在 a[]，输出时注意换行。之后把数组清零。

4. matritrans()：矩阵转置

输入矩阵的行和列数，矩阵把数据放入 a[]。int j = i / n；int k = i % n; b[j + k * m] = a[i];完成转置。 结果存放在 b[]，输出时注意换行。之后把数组清零。

5. matrimulti()：矩阵乘法

输入两矩阵的行数、列数。第一个矩阵把数据放入 a[],第二个矩阵把数据放入 b[],若矩阵一的列数与矩阵二的行数相等才可以做乘法。根据矩阵乘法，把结果存放在 c[]，输出时注意换行。之后把数组清零。

6. hadamulti()：Hadamard 乘积

输入两矩阵的行数、列数，强制要求相同。第一个矩阵把数据放入 a[],第二个矩阵把数据放入 b[],将 a[]和 b[]的对应项相乘，结果存放在 a[]，输出时注意换行。之后把数组清零。

7. conv(int sel)：矩阵卷积

输入矩阵的阶数，在输入矩阵时加上 Padding，数据放在 a[]，遍历 a[]时，用 k[9]保存卷积后的 9 个点。把 k[]的各个数加起来，得到结果点，存放在 c[]。输出时注意换行。之后把数组清零。

8. demo()：卷积应用

读取图像，图像矩阵转数组 b[],加上 Padding 存放在 a[]。输入核的代号，调用 conv(sel)，在得到 c[]后，注意将值为负数的点都转为 0。返回后，数组转图像矩阵，显示图像即可。之后把数组清零。

9. otsu()：OTSU 算法

这个算法首先通过最大类间方差原理，找到区分前景和背景的最优阈值 th，然后对图像进行遍历，进行二值化操作即可。最后显示图像。

10. retain_ex()：取图像

取出图像，调用 retain 函数。

11. retain((Mat image, int n)：保留目标区域并设置背景为黑色

首先我们把三通道图像的 blue、green、red 三层读入到不同数组中，调用 myotsu 函数，对于每一层求二值化的阈值。之后对阈值进行部分调整。然后进行二值化处理，得到值为 255，或者为 0 的数组。然后把数组看作图，把值为 255 的部分看为连通域，调用 bfs_ex 及 bfs 函数通过广度优先搜索得到每一个连通域的初始位置及大小。然后找到最大的连通域，这个就是我们要的目标。然后调用 bfs_1_ex 及 bfs_1 函数，对于这个最大的连通域再进行广度优先搜索，将其中的每个值设为 1。然后对于这个最大连通域之外的每个点都看作背景，把这些点的值设为 0,此时我们得到了一个目标位置全部为 1，其余位置全部为 0 的数组。用这个数组和原本图像对应数组进行 Hadamard 乘积，我们便成功地保留了目标区域并设置背景为黑色，当然在进行 Hadamard 乘积之前，可以把原本图像对应数组的值增大一些，实现目标高亮效果。最后把数组转化为图像即可。

12. bfs(int i, int j, int cnt, int& size, Mat image)：找所有连通域

在不越界且未访问过的情况下，连通域大小加 1，位置放入队列，标记已访问。

13. bfs_ex(int i, int j, int cnt, int& size, Mat image)：找所有连通域

连通域大小加 1，标记已访问。位置放入队列。在队列不空的情况下，位置出队，调用 bfs 函数。

14. bfs_1(int i, int j, Mat image,int what)：找最大连通域

在不越界且未访问过的情况下，位置放入队列，标记以访问，位置值改为 1。

15. bfs_1_ex(int i, int j, Mat image,int what)：找最大连通域

位置值改为 1，标记已访问，位置放入队列。在队列不空的情况下，位置出队，调用 bfs_1 函数。

16. myotsu(Mat& images,int what)：求阈值

根据类间方差最大原理，找到最优的阈值。

在实验过程中遇到的问题及解决方法

一、在每个功能实现后，数组已经被赋值，如何在使用其他功能时，依然正常使用该数组。
解决方法：在每个功能实现后，调用 memset 函数，把数组清零。

二、对图像进行卷积操作后，其值可能为负数，不符合 unsigned char 的范围。
解决方法：如果值为负数，则把该值设为 0。

三、在把数组转化为 Mat 类时，经常报错。
解决方法：发现主要原因是未进行强制转换。在 int 型数组转为 Mat 类时加上强制转换即可。

四、在进行深度优先搜索时 Stack overflow。
解决方法：发现原因是深度优先搜索递归层数过深。因此转为使用广度优先搜索。

心得体会

一、编程过程的体会：

1. 我感觉到计算机的强大能力，编程可以解决无数的问题，数学计算、图像处理等等，我们应该积极地使用编程解决问题。

2. 我感受到知识海洋的广阔。虽然自己写了很多代码，但依然有无数的知识等待我们去探索。Opencv 只是很小的一部分，学海无涯，砥砺前行。

3. 我感受到要善用搜索引擎，在未来的工作岗位上，我们也会遇到很多问题，可是那时很难找到老师、同学为我们答疑解惑，所以要善用搜索引擎，自己找到问题的答案。

4. 四是感觉到库函数的强大。计算机界有一句话叫：不要重复造轮子。前人已经为我们搭好了无数的库函数，他们非常强大，很多时候我们完全可以在理解的基础上直接使用库函数，这样可以提高开发效率。例如在这次大作业里，opencv 提供了 OTSU 算法的库函数，以及求最大连通域的库函数，我们其实可以直接拿过来用。不过我还是决定自己写，因为自己写能更好地锻炼自己的代码能力。

二、对卷积操作的理解：

卷积在人工智能领域有非常重要的作用，卷积神经网络是深度学习的代表算法。卷积在我看来是对一个图像各个小区域的操作，其目的在于提取图像每个小区域的特征。根据卷积应用的演示结果可知，卷积操作并没有明显改变图像的轮廓，而主要是改变了图像的色彩。通过这样的区域处理，这样我们便可以提取到图像的高维特征，有利于让机器更好地学习到图像的特征，并理解图像。

源代码：

一、martix.cpp:

```
#include <iostream>
#include <conio.h>
#include <opencv2/opencv.hpp>
using namespace cv;
using namespace std;
void wait_for_enter()//等待输入
{
	cout << endl << "按回车键继续";
	while (_getch() != '\r');
	cout << endl << endl;
}
void menu();
void matriplus();
void nummulti();
void matritrans();
void matrimulti();
void hadamulti();
void conv(int sel);
void demo();
void retain_ex();
void otsu();
int main()
{
	// 定义相关变量
	char	choice;
	char ch;
	while (true) //注意该循环退出的条件
	{
		wait_for_enter();
		system("cls"); //清屏函数
		menu(); //调用菜单显示函数，自行补充完成
		cout << "按要求输入菜单选择项" << endl;
		//cin >> choice;//  按要求输入菜单选择项choice
		choice = _getch();
		if (choice == '0') //选择退出
		{
			cout << "\n  确定退出吗?" << endl;
			cin >> ch;
			if (ch == 'y' || ch == 'Y')
				break;
			else
				continue;
		}
		switch (choice)
		{
		case '1':matriplus(); break;
		case '2':nummulti(); break;
		case '3':matritrans(); break;
		case '4':matrimulti(); break;
```

```
            case '5':hadamulti(); break;
            case '6':conv(0); break;
            case '7':demo(); break;
            case '8':otsu(); break;
            case '9':retain_ex(); break;
            default:
                cout << "\n  输入错误，请重新输入" << endl;
                wait_for_enter();
            }
            cout << endl;
        }
        return 0;
}
```

二、func.cpp:

```
#include <iostream>
#include <conio.h>
#include <opencv2/opencv.hpp>
#include<queue>
using namespace cv;
using namespace std;
//使用到的数组,idx 为找最大连通子图，进行广度优先搜索使用的数组
int a[258 * 258] = { 0 }, b[256 * 256] = { 0 }, c[256 * 256] = { 0 }, idx[30010] = { 0 };
struct three //对三通道读取的结构体
{
        int blu;
        int gree;
        int re;
}d[30010],e[30010];
struct place //找最大连通子图，进行广度优先搜索使用的结构体
{
        int x;
        int y;
        int num;
}f[256 * 256];
queue<int> q, p;//找最大连通子图，进行广度优先搜索使用的队列
void menu()//菜单
{
        cout << "************************************************************" << endl;
        cout << " *          1  矩阵加法          2  矩阵数乘          3  矩阵转置          *" << endl;
        cout << " *          4  矩阵乘法          5 Hadamard 乘积    6  矩阵卷积          *" << endl;
        cout << " *          7  卷积应用          8 OTSU 算法          9  保留区域          *" << endl;
        cout << " *          0  退出系统                                                    *" << endl;
        cout << "************************************************************" << endl;
        cout << "选择菜单项<0~9>:" << endl;
        return;
}
void matriplus()//矩阵加法
{
        int m, n;
        cout << "请输入两个矩阵的行和列数：" << endl;
```

```
	cin >> m >> n;
	cout << "请输入矩阵 a：" << endl;
	for (int i = 0; i < n * m; i++)
		cin >> a[i];
	cout << "请输入矩阵 b：" << endl;
	for (int i = 0; i < n * m; i++)
	{
		cin >> b[i];
		a[i] += b[i];
	}
	cout << "结果为：";
	for (int i = 0; i < n * m; i++)
	{
		if (i % n == 0)
			cout << endl;
		cout << a[i] << " ";
	}
	memset(a, 0, sizeof(a));
	memset(b, 0, sizeof(b));
	return;
}
void nummulti()//矩阵数乘
{
	int x, m, n;
	cout << "请输入矩阵的行和列数以及矩阵要乘的数：" << endl;
	cin >> m >> n>> x;
	cout << "请输入矩阵：" << endl;
	for (int i = 0; i < n * m; i++)
	{
		cin >> a[i];
		a[i] *= x;
	}
	cout << "结果为：";
	for (int i = 0; i < n * m; i++)
	{
		if (i % n == 0)
			cout << endl;
		cout << a[i] << " ";
	}
	memset(a, 0, sizeof(a));
	return;
}
void matritrans()//矩阵转置
{
	int m, n;
	cout << "请输入矩阵的行和列数：" << endl;
	cin >> m >> n;
	cout << "请输入矩阵：" << endl;
	for (int i = 0; i < n * m; i++)
	{
		cin >> a[i];
		int j = i / n;
		int k = i % n;
```

装 订 线

```
            b[j + k * m] = a[i];
        }
        cout << "结果为：";
        for (int i = 0; i < n * m; i++)
        {
            if (i % m == 0)
                cout << endl;
            cout << b[i] << " ";
        }
        memset(a, 0, sizeof(a));
        memset(b, 0, sizeof(b));
        return;
}
void matrimulti()//矩阵乘法
{
        int m, n, p, q;
        cout << "请输入第一个和第二个矩阵的行和列数：" << endl;
        cin >> m >> n >> p >> q;
        if (n != p)
        {
            cout << "两矩阵不能进行乘法" << endl;
            return;
        }
        cout << "请输入矩阵 a：" << endl;
        for (int i = 0; i < n * m; i++)
            cin >> a[i];
        cout << "请输入矩阵 b：" << endl;
        for (int i = 0; i < p * q; i++)
            cin >> b[i];
        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < q; j++)
            {
                for (int k = 0; k < n; k++)
                    c[i * q + j] += a[i * n + k] * b[k * q + j];
            }
        }
        cout << "结果为：";
        for (int i = 0; i < m * q; i++)
        {
            if (i % q == 0)
                cout << endl;
            cout << c[i] << " ";
        }
        memset(a, 0, sizeof(a));
        memset(b, 0, sizeof(b));
        memset(c, 0, sizeof(c));
        return;
}
void hadamulti()//矩阵 Hadamard 乘积
{
        int m, n;
        cout << "请输入两个矩阵的行和列数：" << endl;
        cin >> m >> n;
```

```
	cout << "请输入矩阵 a：" << endl;
	for (int i = 0; i < n * m; i++)
		cin >> a[i];
	cout << "请输入矩阵 b：" << endl;
	for (int i = 0; i < n * m; i++)
	{
		cin >> b[i];
		a[i] *= b[i];
	}
	cout << "结果为：";
	for (int i = 0; i < n * m; i++)
	{
		if (i % n == 0)
			cout << endl;
		cout << a[i] << " ";
	}
	memset(a, 0, sizeof(a));
	memset(b, 0, sizeof(b));
	return;
}
void conv(int sel)//矩阵卷积（用于第 6 和第 7 个功能）
{
	int n, m = 0, p = 0, q;
	int d[9] = { 0 };
	switch (sel)//选择卷积核
	{
	case 0:
	{
		for (int i = 0; i < 9; i++)
		{
			if (i % 3 == 0)
				d[i] = -1;
			else if (i % 3 == 2)
				d[i] = 1;
		}
		break;
	}
	case 1:
	{
		for (int i = 0; i < 9; i++)
		{
			d[i] = 1;
		}
		break;
	}
	case 2:
	{
		d[0] = 1; d[1] = 2; d[2] = 1; d[3] = 0; d[4] = 0; d[5] = 0; d[6] = -1; d[7] = -2; d[8] = -1;
		break;
	}
	case 3:
	{
		d[0] = -1; d[1] = 0; d[2] = 1; d[3] = -2; d[4] = 0; d[5] = 2; d[6] = -1; d[7] = 0; d[8] = 1;
		break;
```

```
}
case 4:
{
	for (int i = 0; i < 9; i++)
	{
		if (i == 4)
			d[i] = 9;
		else
			d[i] = -1;
	}
	break;
}
case 5:
{
	d[0] = -1; d[1] = -1; d[2] = 0; d[3] = -1; d[4] = 0; d[5] = 1; d[6] = 0; d[7] = 1; d[8] = 1;
	break;
}
case 6:
{
	d[0] = 1; d[1] = 2; d[2] = 1; d[3] = 2; d[4] = 4; d[5] = 2; d[6] = 1; d[7] = 2; d[8] = 1;
	break;
}
default:
{
	cout << "输入错误" << endl;
	return;
}
}
if (sel == 0)//这是第 6 个功能
{
	cout << "请输入矩阵的阶数：" << endl;
	cin >> n;
	cout << "请输入矩阵：" << endl;
	for (int i = n + 3; i < n * n + 3 * n + 1; i++)//加 Padding
	{
		if (i % (n + 2) == 0 || i % (n + 2) == n + 1)
			continue;
		else
			cin >> a[i];
	}
}
else
{
	n = 256;//这是第 7 个功能
}
for (int i = 0; i < n * (n + 2); i++)//进行卷积
{
	if (i % (n + 2) == n || i % (n + 2) == n + 1)
		continue;
	else
	{
		int k[9] = { 0 };//存放 9 个点的结果
		m = 0;
		q = 0;
```

```
                for (int j = i; j < i + 2 * n + 7; j++)
                {
                    if (j % (n + 2) == i % (n + 2))
                    {
                        k[m] = a[j] * d[q % 9];
                        m++;
                        q++;
                    }
                    else if (j % (n + 2) == (i + 1) % (n + 2))
                    {
                        k[m] = a[j] * d[q % 9];
                        m++;
                        q++;
                    }
                    else if (j % (n + 2) == (i + 2) % (n + 2))
                    {
                        k[m] = a[j] * d[q % 9];
                        m++;
                        q++;
                    }
                }
                for (int l = 0; l < 9; l++)//把 9 个点加起来
                    c[p] += k[l];
                if(sel!=0)//区分第 6 个功能和第 7 个功能
                    c[p] /= 9;
                if (c[p] < 0&&sel!=0)//判断 unisigned char 类型是否越界
                    c[p] = 0;
                p++;
            }
        }
        if (sel == 0)
        {
            cout << "结果为：" << endl;
            for (int i = 0; i < n * n; i++)
            {
                if (i % n == 0)
                    cout << endl;
                cout << c[i] << " ";
            }
            memset(a, 0, sizeof(a));
            memset(c, 0, sizeof(c));
            return;
        }
        return;
}
void demo()//卷积应用
{
        int sel;
        Mat image = imread("D:\\1.jpg", 0);//单通道读取
        for (int i = 0; i < 256; i++)
            for (int j = 0; j < 256; j++)
            {
                b[i * 256 + j] = image.at<uchar>(j, i);//矩阵转数组
            }
```

```
	int l = 0;
	for (int i = 256 + 3; i < 256 * 256 + 3 * 256 + 1; i++)//加 Padding
	{
		if (i % (256 + 2) == 0 || i % (256 + 2) == 256 + 1)
			continue;
		else
		{
			a[i] = b[l];
			++l;
		}
	}
	cout << "请输入核的代号" << endl;
	cin >> sel;
	conv(sel);
	Mat result = Mat(256, 256, CV_8U, Scalar::all(0));
	for (int i = 0; i < 256; i++)
		for (int j = 0; j < 256; j++)
		{
			result.at<uchar>(j, i) = (unsigned char)c[i * 256 + j];
		}
	memset(a, 0, sizeof(a));
	memset(b, 0, sizeof(b));
	memset(c, 0, sizeof(c));
	imshow("卷积结果", result);
	waitKey(0);
	return;
}
int myotsu(Mat& images,int what)//寻找阈值
{
	int th=0;
	const int Scale = 256;	//单通道总灰度 256 级
	int count[Scale] = { 0 };
	int sum = images.cols * images.rows;
	float Pro[Scale] = { 0 };//每个灰度值所占总像素比例
	float w0, w1, v0, v1, u0, u1, deltaTmp, deltaMax = 0;
	if (what == 0)
	{
		for (int i = 0; i < images.cols; i++)
		{
			for (int j = 0; j < images.rows; j++)
			{
				count[images.at<uchar>(j, i)]++;//统计每个灰度级中像素的个数
			}
		}
	}
	else
	{
		for (int i = 0; i < images.cols; i++)
		{
			for (int j = 0; j < images.rows; j++)
			{
				count[d[j * images.cols + i].blu]++;//统计每个灰度级中像素的个数
			}
		}
```

```
	}
	for (int i = 0; i < Scale; i++)
	{
		Pro[i] = count[i] * 1.0 / sum;//计算每个灰度级的像素数目占整幅图像的比例
	}

	for (int i = 0; i < Scale; i++)//测试哪一个的类间方差最大
	{
		w0 = w1 = v0 = v1 = u0 = u1 = deltaTmp = 0;
		for (int j = 0; j < Scale; j++)
		{
			if (j <= i)
			{
				w0 += Pro[j];
				v0 += j * Pro[j];
			}
			else
			{
				w1 += Pro[j];
				v1 += j * Pro[j];
			}
		}
		u0 = v0 / w0;
		u1 = v1 / w1;
		deltaTmp = (float)(w0 * w1 * pow((u0 - u1), 2)); //方差公式
		if (deltaTmp > deltaMax)
		{
			deltaMax = deltaTmp;
			th = i;
		}
	}
	return th;
}
void otsu()//OTSU 算法
{
	Mat image = imread("D:\\1.jpg", 0);
	int th=myotsu(image,0);//得阈值
	for (int i = 0; i < image.rows; i++)
		for (int j = 0; j < image.cols; j++)
		{
			if (image.at<uchar>(i, j) > th)
			{
				image.at<uchar>(i, j) = 255;
			}
			else
			{
				image.at<uchar>(i, j) = 0;
			}
		}
	imshow("OTSU", image);
	waitKey(0);
	return;
}
void bfs(int i, int j, int cnt, int& size, Mat image)//广度优先搜索找所有白色连通域
```

```
{
    if (i < 0 || i >= image.rows || j < 0 || j >= image.cols)
        return;   //出界
    else if (idx[i*image.cols + j] > 0 || d[i * image.cols + j].blu ==0)
        return;   //为 0 或者已经访问过
    else
    {
        size++;
        q.push(i);
        q.push(j);
        idx[i * image.cols + j] = cnt;
        return;
    }
}
void bfs_ex(int i, int j, int cnt, int& size, Mat image)//广度优先搜索找所有白色连通域
{
    size++;//连通域大小
    q.push(i);
    q.push(j);
    idx[i * image.cols + j] = cnt;//已经访问过
    while (!q.empty())
    {
        int k = q.front();
        q.pop();
        int l = q.front();
        q.pop();
        bfs(k - 1, l, cnt, size, image);
        bfs(k + 1, l, cnt, size, image);
        bfs(k, l - 1, cnt, size, image);
        bfs(k, l + 1, cnt, size, image);
    }
}
void bfs_1(int i, int j, Mat image,int what)//广度优先搜索调整最大白色连通域值为 1
{
    if (i < 0 || i >= image.rows || j < 0 || j >= image.cols)
        return;   //出界
    else if (idx[i * image.cols + j] > 0 || (what == 0 && d[i * image.cols + j].blu == 0) || (what == 1
&& d[i * image.cols + j].gree == 0) || (what == 2 && d[i * image.cols + j].re == 0))
        return;   //不是 255 或者已经访问过
    else
    {
        idx[i * image.cols + j] = 1;
        p.push(i);
        p.push(j);
        if(what==0)
            d[i * image.cols + j].blu = 1;
        else if (what == 1)
            d[i * image.cols + j].gree = 1;
        else if (what == 2)
            d[i * image.cols + j].re = 1;
        return;
    }
}
void bfs_1_ex(int i, int j, Mat image,int what)//广度优先搜索调整最大白色连通域值为 1
```

```
{
    if (what == 0)
        d[i * image.cols + j].blu = 1;
    else if(what==1)
        d[i * image.cols + j].gree = 1;
    else if (what == 2)
        d[i * image.cols + j].re = 1;
    idx[i * image.cols + j] = 1;
    p.push(i);
    p.push(j);
    while (!p.empty())
    {
        int k = p.front();
        p.pop();
        int l = p.front();
        p.pop();
        bfs_1(k - 1, l, image,what);
        bfs_1(k + 1, l, image,what);
        bfs_1(k, l - 1, image,what);
        bfs_1(k, l + 1, image,what);
    }
}
void retain(Mat image,int n)//保留目标区域
{
    int cnt = 0;//记录白色连通域多少
    int size = 0;//记录白色连通域大小
    Mat ima = image;
    for (int i = 0; i < image.rows; i++)
        for (int j = 0; j < image.cols; j++)
        {
            Vec3b pixel = image.at<Vec3b>(i, j);//将三通道三层取出
            d[i * image.cols + j].blu = pixel[0];
            d[i * image.cols + j].gree = pixel[1];
            d[i * image.cols + j].re = pixel[2];
            e[i * image.cols + j].blu = pixel[0];
            e[i * image.cols + j].gree = pixel[1];
            e[i * image.cols + j].re = pixel[2];
        }
    int th = myotsu(image, 1);//找到阈值
    if (n == 2)//处理阈值
        th -= 37;
    for (int i = 0; i < image.rows; i++)//二值化处理
        for (int j = 0; j < image.cols; j++)
        {
            if (d[i * image.cols + j].blu > th)
            {
                d[i * image.cols + j].blu = 255;
            }
            else
            {
                d[i * image.cols + j].blu = 0;
            }
            if (d[i * image.cols + j].gree > th)
            {
```

```
                d[i * image.cols + j].gree = 255;
            }
            else
            {
                d[i * image.cols + j].gree = 0;
            }
            if (d[i * image.cols + j].re > th)
            {
                d[i * image.cols + j].re = 255;
            }
            else
            {
                d[i * image.cols + j].re = 0;
            }
        }
    for (int i = 0; i < image.rows; i++)//进行广度优先搜索找所有白色（255）连通域
        for (int j = 0; j < image.cols; j++)
        {
            if (d[i * image.cols + j].blu == 255 && idx[i * image.cols + j] == 0)
            {
                size = 0;
                bfs_ex(i, j, ++cnt, size, image);
                f[cnt].num = size;
                f[cnt].x = i;
                f[cnt].y = j;
            }
        }
    for (int i = 1; i <= cnt; i++)//对找到的连通域按大小排序
    {
        place t;
        for (int j = i; j <= cnt; j++)
        {
            if (f[i].num > f[j].num)
            {
                t = f[i];
                f[i] = f[j];
                f[j] = t;
            }
        }
    }
    int p = f[cnt].x;//最大连通域的位置
    int q = f[cnt].y;
    memset(idx, 0, sizeof(idx));
    bfs_1_ex(p, q, image, 0);//处理 blue 层最大连通域，使值为 1
    for(int i=0;i<image.rows;i++)//其余小白色连通域变为背景
        for (int j = 0; j < image.cols; j++)
        {
            if (idx[i * image.cols + j] == 0&& d[i * image.cols + j].blu!=0)
            {
                d[i * image.cols + j].blu = 0;
            }
        }
    memset(idx, 0, sizeof(idx));
    bfs_1_ex(p, q, image, 1);//处理 green 层
```

```
	for (int i = 0; i < image.rows; i++)
		for (int j = 0; j < image.cols; j++)
		{
			if (idx[i * image.cols + j] == 0&& d[i * image.cols + j].gree != 0)
				d[i * image.cols + j].gree = 0;
		}
	memset(idx, 0, sizeof(idx));
	bfs_1_ex(p, q, image, 2);
	for (int i = 0; i < image.rows; i++)//处理 red 层
		for (int j = 0; j < image.cols; j++)
		{
			if (idx[i * image.cols + j] == 0&& d[i * image.cols + j].re!= 0)
				d[i * image.cols + j].re = 0;
		}
	if (n == 3)//对于图多角星形进行高亮处理
	{
		for (int i = 0; i < image.rows * image.cols; i++)
		{
			e[i].blu += 100;
			e[i].gree += 100;
			e[i].re += 100;
		}
	}
	for (int i = 0; i < image.rows * image.cols; i++)//利用 Hadamard 乘积得到目标区域，并使背景变
黑
	{
		e[i].blu = e[i].blu*d[i].blu;
		e[i].gree *= d[i].gree;
		e[i].re *= d[i].re;
	}
	for (int i = 0; i < image.rows; i++)
		for (int j = 0; j < image.cols; j++)
		{
			Vec3b pixel;
			pixel[0] = (unsigned char)e[i * image.cols + j].blu;//Blue
			pixel[1] = (unsigned char)e[i * image.cols + j].gree;//Green
			pixel[2] = (unsigned char)e[i * image.cols + j].re;//Red
			ima.at<Vec3b>(i, j) = pixel;
		}
	memset(d, 0, sizeof(d));
	memset(e, 0, sizeof(e));
	memset(idx, 0, sizeof(idx));
	memset(f, 0, sizeof(f));
	imshow("结果", ima);
	waitKey(0);
	ima = Mat(256, 256, CV_8U, Scalar::all(0));
	return;
}
void retain_ex()//保留目标区域
{
	Mat im_2 = imread("D:\\2.jpg", 1);
	Mat im_3 = imread("D:\\3.jpg", 1);
	Mat im_4 = imread("D:\\4.jpg", 1);
	Mat im_5 = imread("D:\\5.jpg", 1);
```

```
    retain(im_2, 2);
    retain(im_3, 3);
    retain(im_4, 4);
    retain(im_5, 5);
    return;
}
```

装

订

线