高级图像处理与分析课程实验报告

学号：SA23225226　　姓名：郭浩天　　日期：2023.11.14

实验名称	频域滤波
实验内容	1、 灰度图像的 DFT 和 IDFT。 具体内容：利用 OpenCV 提供的 cvDFT 函数对图像进行 DFT 和 IDFT 变换 。 2、利用理想高通和低通滤波器对灰度图像进行频域滤波 具体内容：利用 cvDFT 函数实现 DFT，在频域上利用理想高通和低通滤波器进行滤波，并把滤波过后的图像显示在屏幕上（观察振铃现象），要求截止频 率可输入。 3、利用布特沃斯高通和低通滤波器对灰度图像进行频域滤波。 具体内容：利用 cvDFT 函数实现 DFT，在频域上进行利用布特沃斯高通和低通滤波器进行滤波，并把滤波过后的图像显示在屏幕上（观察振铃现象），要求截止频率和 n 可输入。
实验完成情况 (包括完成的实验内容及每个实验的完成程度。注意要贴出每个实验的核心代码)	

```
/*灰度图像的 DFT 和 IDFT。
 *
 *具体内容：利用 OpenCV 提供的 cvDFT 函数对图像进行 DFT 和 IDFT 变换.
 *
 * */
void DFTAndIDFT(string& src) {
	Mat image = imread(src, 0);
	imshow("输入图像", image);
	//获取最优傅里叶尺寸大小，是2的整数次方
	int w = getOptimalDFTSize(image.cols);
	int h = getOptimalDFTSize(image.rows);
	Mat padded;
	//填充常量并将图像保存到padded中
	copyMakeBorder(image, padded, 0, h - image.rows, 0, w - image.cols,
BORDER_CONSTANT, Scalar::all(0));
	//Mat数组，前面为扩展后的图像，后面的为空图像
	Mat plane[] =
{ Mat_<float>(padded),Mat::zeros(padded.size(),CV_32F) };
	Mat complexIm;
	//合并通道，DFT需要一个两通道的Mat
	merge(plane, 2, complexIm);
	//进行傅立叶变换，结果保存在自身
	dft(complexIm, complexIm);
	// compute log(1 + sqrt(Re(DFT(img))**2 + Im(DFT(img))**2))
```

```
        //分离通道
        split(complexIm, plane);
        //获取幅度图像，0通道为实数通道，1为虚数，因为二维傅立叶变换结果是复数
        magnitude(plane[0], plane[1], plane[0]);
        //一下的操作是移动图像，左上与右下交换位置，右上与左下交换位置
        //对傅立叶变换的图像进行重排，4个区块，从左到右，从上到下 分别为q0, q1, q2, q3
        int cx = padded.cols / 2;
        int cy = padded.rows / 2;
        //对调q0和q3, q1和q2
        Mat temp;
        Mat part1(plane[0], Rect(0, 0, cx, cy));
        Mat part2(plane[0], Rect(cx, 0, cx, cy));
        Mat part3(plane[0], Rect(0, cy, cx, cy));
        Mat part4(plane[0], Rect(cx, cy, cx, cy));

        part1.copyTo(temp);
        part4.copyTo(part1);
        temp.copyTo(part4);

        part2.copyTo(temp);
        part3.copyTo(part2);
        temp.copyTo(part3);


        Mat _complexim;
        //把变换结果复制一份，进行逆变换，也就是恢复原图
        complexIm.copyTo(_complexim);
        //创建两个通道，类型为float，大小为填充后的尺寸
        Mat iDft[] = { Mat::zeros(plane[0].size(),CV_32F),Mat::zeros(plane[0].size(),CV_32F) };
        //傅立叶逆变换
        idft(_complexim, _complexim);
        //分离通道，主要获取0通道
        split(_complexim, iDft);
        magnitude(iDft[0], iDft[1], iDft[0]);
        //归一化处理，float类型的显示范围为0-1,大于1为白色，小于0为黑色
        normalize(iDft[0], iDft[0], 1, 0, NORM_MINMAX);
        imshow("IDFT后图像", iDft[0]);//显示逆变换
        //傅立叶变换后的图片不好分析，进行对数处理，结果比较好看
        plane[0] += Scalar::all(1);
        log(plane[0], plane[0]);
        normalize(plane[0], plane[0], 1, 0, NORM_MINMAX);
        imshow("DFT后图像", plane[0]);
        waitKey(0);
        destroyAllWindows();
}

/*利用理想高通和低通滤波器对灰度图像进行频域滤波
 *
 * 具体内容：利用 cvDFT 函数实现 DFT，在频域上利用理想高通和低通滤波器进行滤波，并把滤波过后的图像显示在屏幕上（观察振铃现象），要求截止频率可输入。
 *
 * */
void idealFilterOfLowOrHighPass(string& src, double f, int model) {
        Mat image = imread(src, 0);
        imshow("输入图像", image);
```

```
        Mat img = image.clone();
        //调整图像加速傅里叶变换
        if (model == 0) cout << "低通 ";
        if (model == 1) cout << "高通 ";
        int M = getOptimalDFTSize(img.rows);
        int N = getOptimalDFTSize(img.cols);
        Mat padded;
        copyMakeBorder(img, padded, 0, M - img.rows, 0, N - img.cols,
BORDER_CONSTANT, Scalar::all(0));
        //记录傅里叶变换的实部和虚部
        Mat planes[] = { Mat_<float>(padded), Mat::zeros(padded.size(),
CV_32F) };
        Mat complexImg;
        merge(planes, 2, complexImg);
        //进行傅里叶变换
        dft(complexImg, complexImg);
        //获取图像
        Mat mag = complexImg;
        //保证偶数的边长
        mag = mag(Rect(0, 0, mag.cols & -2, mag.rows & -2));
        //这里为什么&上-2具体查看opencv文档
        //其实是为了把行和列变成偶数 -2的二进制是11111111.......10 最后一位是
0
        //获取中心点坐标
        int cx = mag.cols / 2;
        int cy = mag.rows / 2;
        //调整频域
        Mat tmp;
        Mat q0(mag, Rect(0, 0, cx, cy));
        Mat q1(mag, Rect(cx, 0, cx, cy));
        Mat q2(mag, Rect(0, cy, cx, cy));
        Mat q3(mag, Rect(cx, cy, cx, cy));

        q0.copyTo(tmp);
        q3.copyTo(q0);
        tmp.copyTo(q3);

        q1.copyTo(tmp);
        q2.copyTo(q1);
        tmp.copyTo(q2);
        //Do为自己设定的阀值具体看公式

        //处理按公式保留中心部分
        for (int y = 0; y < mag.rows; y++) {
                auto* data = mag.ptr<double>(y);
                for (int x = 0; x < mag.cols; x++) {
                        double d = sqrt(pow((y - cy), 2) + pow((x - cx), 2));
                        if (model == 0) {
                                if (d > f) data[x] = 0;//低通
                        }
                        else if (model == 1) {
                                if (d < f) data[x] = 0;//高通
                        }
                }
        }
        //再调整频域
        q0.copyTo(tmp);
        q3.copyTo(q0);
```

```
        tmp.copyTo(q3);
        q1.copyTo(tmp);
        q2.copyTo(q1);
        tmp.copyTo(q2);
        //逆变换
        Mat invDFT, invDFTcvt;
        idft(mag, invDFT, DFT_SCALE | DFT_REAL_OUTPUT); // Applying IDFT
        invDFT.convertTo(invDFTcvt, CV_8U);

        imshow("理想低通/高通滤波器", invDFTcvt);
        waitKey(0);
        destroyAllWindows();
}

/*利用布特沃斯高通和低通滤波器对灰度图像进行频域滤波。
 *
 * 具体内容：利用 cvDFT 函数实现 DFT，在频域上进行利用布特沃斯高通和低通滤波器进行滤波，并把滤波过后的图像显示在屏幕上（观察振铃现象），要求截止频率和 n 可输入。
 *
 * */
void butterworthFilterOfLowOrHighPass(string& src, double f, int n, int model) {
        Mat image = imread(src, 0); // Read the file
        imshow("原始图像", image);

        Mat img = image.clone();
        if (model == 0) cout << "低通 ";
        if (model == 1) cout << "高通 ";
        //调整图像加速傅里叶变换

        int M = getOptimalDFTSize(img.rows);
        int N = getOptimalDFTSize(img.cols);
        Mat padded;
        copyMakeBorder(img, padded, 0, M - img.rows, 0, N - img.cols, BORDER_CONSTANT, Scalar::all(0));

        Mat planes[] = { Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F) };
        Mat complexImg;
        merge(planes, 2, complexImg);

        dft(complexImg, complexImg);

        Mat mag = complexImg;
        mag = mag(Rect(0, 0, mag.cols & -2, mag.rows & -2));

        int cx = mag.cols / 2;
        int cy = mag.rows / 2;

        Mat tmp;
        Mat q0(mag, Rect(0, 0, cx, cy));
        Mat q1(mag, Rect(cx, 0, cx, cy));
        Mat q2(mag, Rect(0, cy, cx, cy));
        Mat q3(mag, Rect(cx, cy, cx, cy));

        q0.copyTo(tmp);
        q3.copyTo(q0);
```

```
        tmp.copyTo(q3);

        q1.copyTo(tmp);
        q2.copyTo(q1);
        tmp.copyTo(q2);



        for (int y = 0; y < mag.rows; y++)
        {
                auto* data = mag.ptr<double>(y);
                for (int x = 0; x < mag.cols; x++)
                {
                        //cout << data[x] << endl;
                        double d = sqrt(pow((y - cy), 2) + pow((x - cx), 2));
                        //cout << d << endl;
                        double h = 0;
                        //    阶数n=1 无振铃和负值
                        //    阶数n=2 轻微振铃和负值
                        //    阶数n=5 明显振铃和负值
                        //    阶数n=20 与ILPF相似
                        //H = 1 / (1+(D/D0)^2n)
                        if (model == 0)  h = 1.0 / (1 + pow(d / f, 2 * n));
                        if (model == 1)  h = 1.0 / (1 + pow(f / d, 2 * n));
                        if (h <= 0.5)
                        {
                                data[x] = 0;
                        }

                }
        }
        q0.copyTo(tmp);
        q3.copyTo(q0);
        tmp.copyTo(q3);
        q1.copyTo(tmp);
        q2.copyTo(q1);
        tmp.copyTo(q2);
        //逆变换
        Mat invDFT, invDFTcvt;
        idft(complexImg, invDFT, DFT_SCALE | DFT_REAL_OUTPUT); // Applying
IDFT
        invDFT.convertTo(invDFTcvt, CV_8U);
        imshow("巴特沃斯低通/高通滤波器", invDFTcvt);

        waitKey(0);
        destroyAllWindows();
}
```

实验中的问题

(包括在实验中遇到的问题，以及解决问题的方法)

OpenCV 中的 DFT 变换

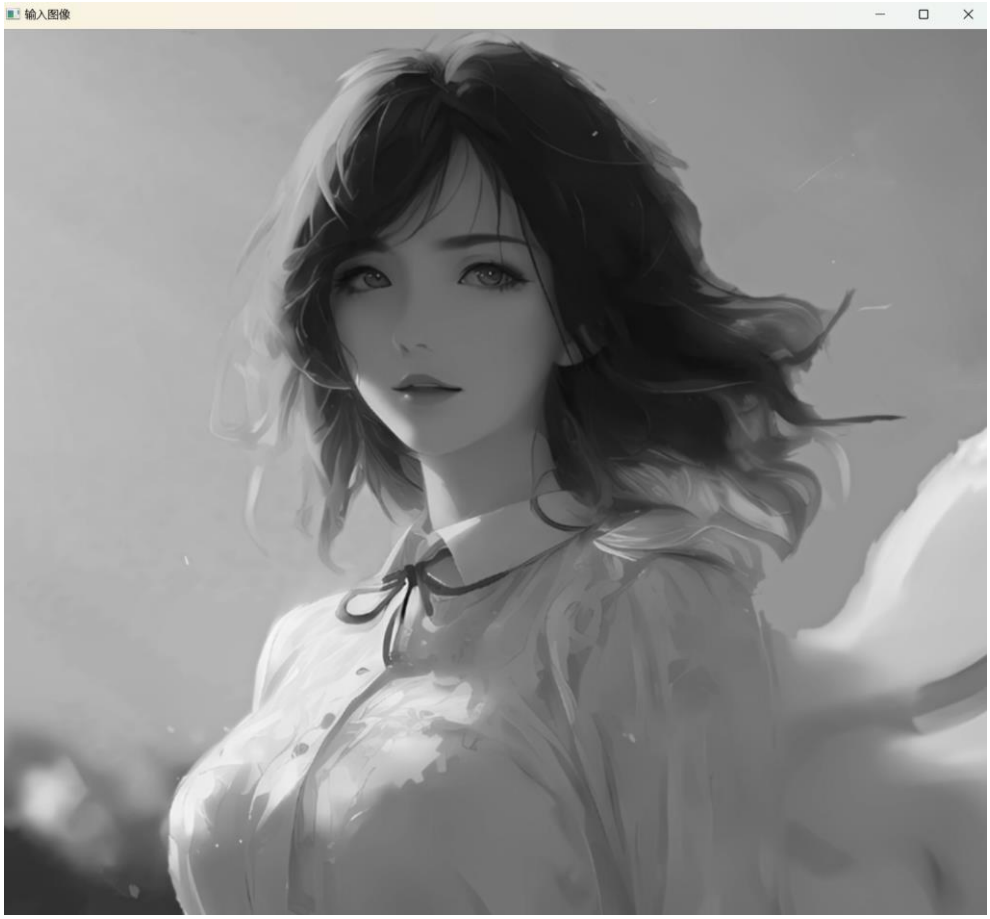
void cvDFT(const CvArr* src, CvArr* dst, int flags);

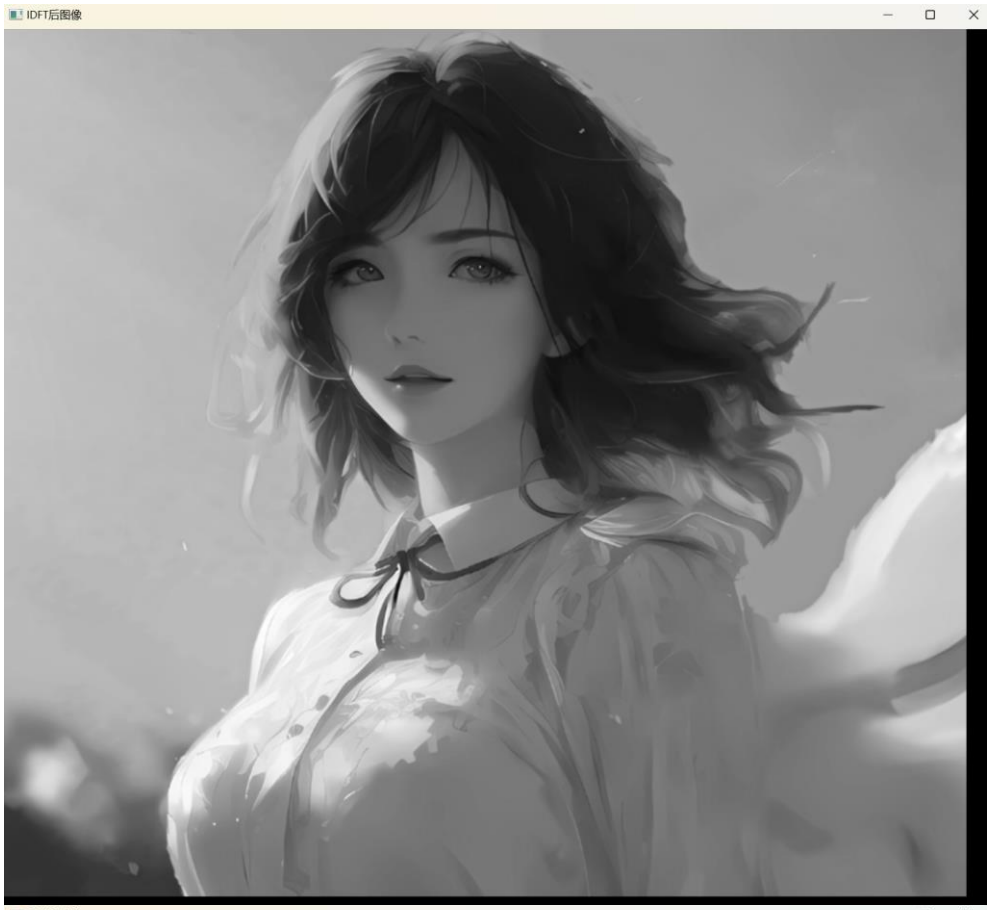
src 输入数组, 实数或者复数.

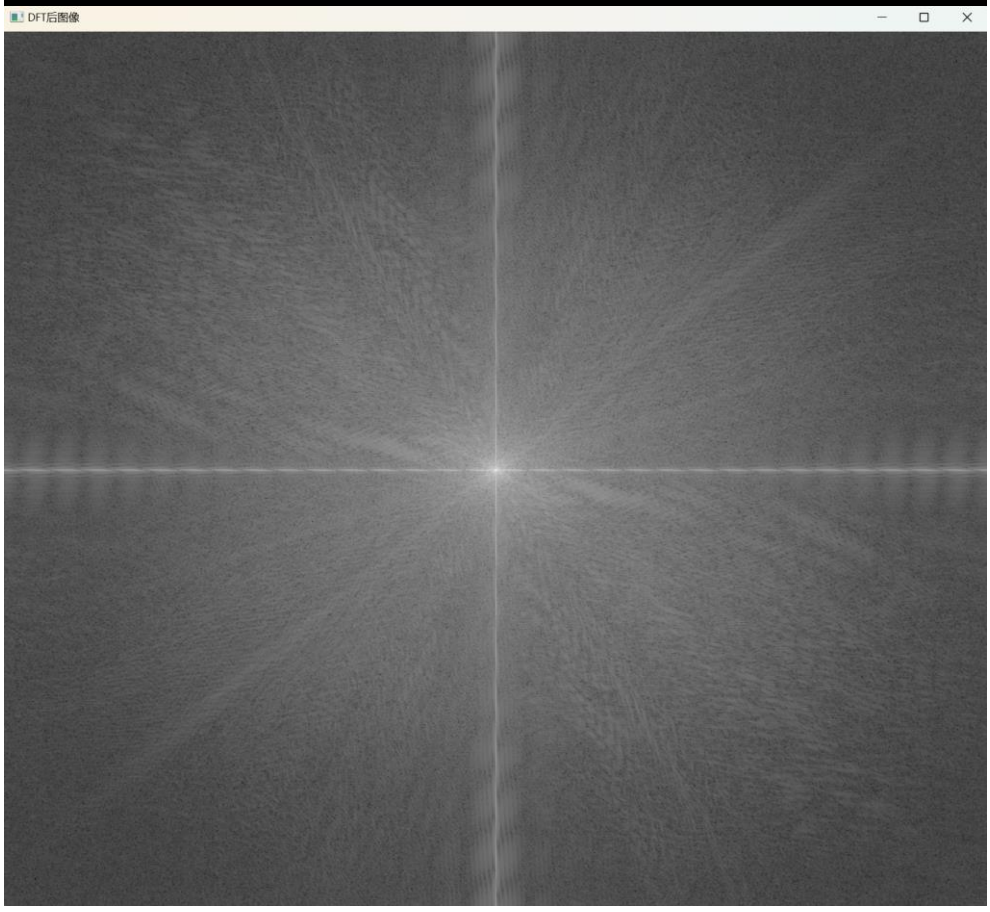
dst 输出数组，和输入数组有相同的类型和大小。

flags 变换标志, 下面的值的组合:

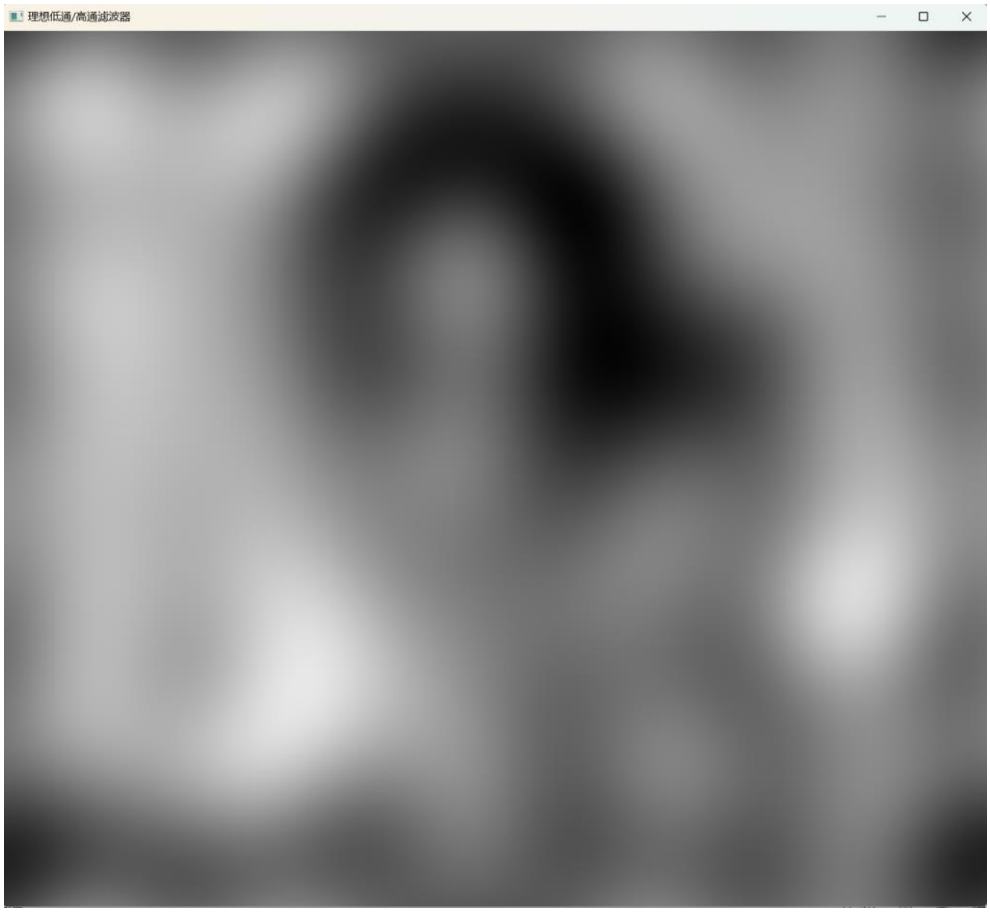
CV_DXT_FORWARD - 正向 1D 或者 2D 变换. 结果不被缩放.

	CV_DXT_INVERSE - 逆向 1D 或者 2D 变换. 结果不被缩放。 CV_DXT_FORWARD 和 CV_DXT_INVERSE 是互斥的. 利用 cvDFT 对图像进行处理需要考虑虚部，对虚步进行填 0 操作。 图像在进行 DFT 前要进行归一化处理。
实验结果 （实验完成后的源码和打包文件的说明）	

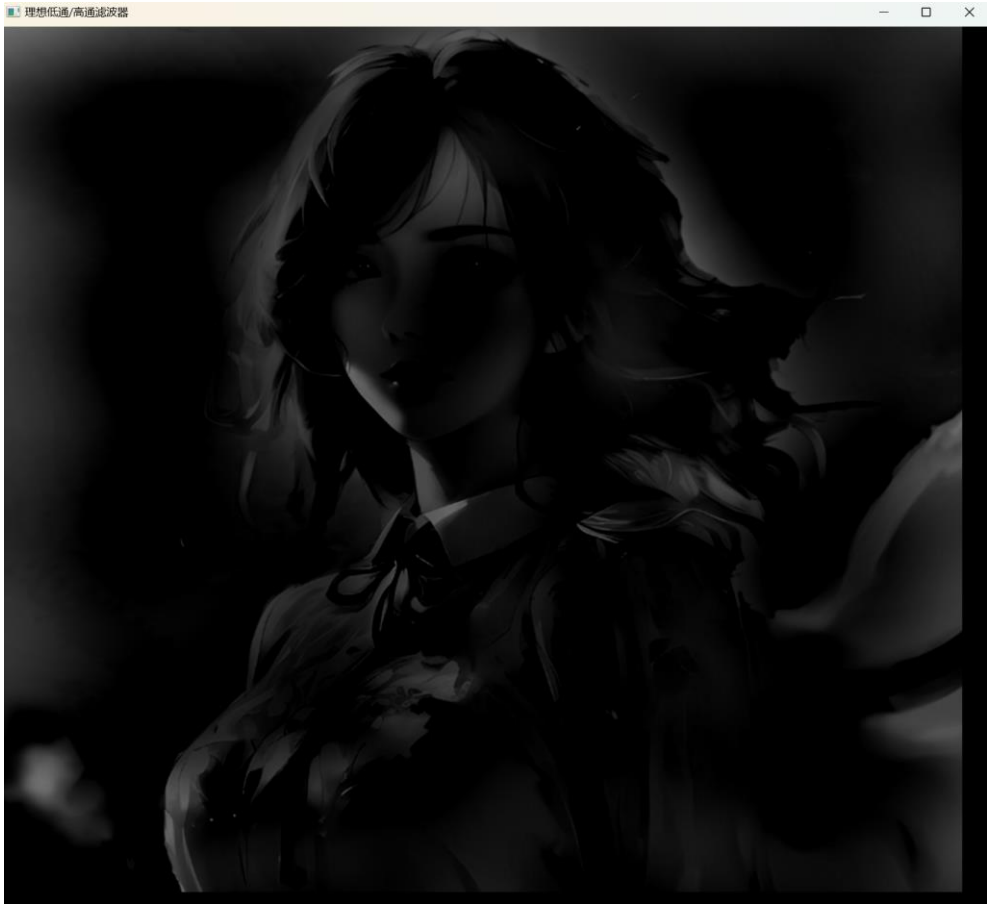




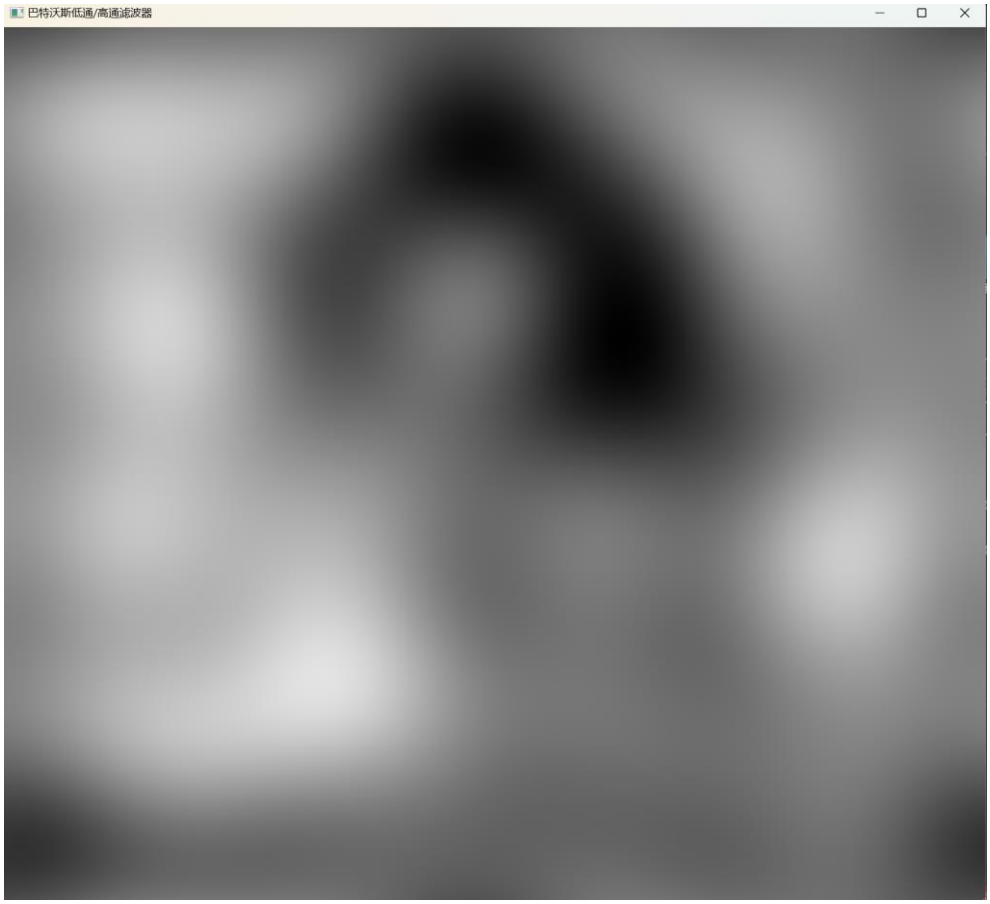
理想低通：



理想高通：



巴特沃斯低通：



巴特沃斯高通：