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：2023.10.25

实验名称	空域滤波
实验内容	1、利用均值模板平滑灰度图像。 具体内容：利用 OpenCV 对图像像素进行操作，分别利用 3*3、5*5 和 9*9 尺寸的均值模板平滑灰度图像 2、利用高斯模板平滑灰度图像。 具体内容：利用 OpenCV 对图像像素进行操作，分别利用 3*3、5*5 和 9*9 尺寸的高斯模板平滑灰度图像 3、利用 Laplacian、Robert、Sobel 模板锐化灰度图像。 具体内容：利用 OpenCV 对图像像素进行操作，分别利用 Laplacian、Robert、 Sobel 模板锐化灰度图像 4、利用高提升滤波算法增强灰度图像。 具体内容：利用 OpenCV 对图像像素进行操作，设计高提升滤波算法增强图像 5、利用均值模板平滑彩色图像。 具体内容：利用 OpenCV 分别对图像像素的 RGB 三个通道进行操作，利用 3*3、5*5 和 9*9 尺寸的均值模板平滑彩色图像 6、利用高斯模板平滑彩色图像。 具体内容：利用 OpenCV 分别对图像像素的 RGB 三个通道进行操作，分别利用 3*3、5*5 和 9*9 尺寸的高斯模板平滑彩色图像 7、利用 Laplacian、Robert、Sobel 模板锐化灰度图像。 具体内容：利用 OpenCV 分别对图像像素的 RGB 三个通道进行操作，分别利用 Laplacian、Robert、Sobel 模板锐化彩色图像

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)

```
void Filter(string& src, int x, int y, int flag, string model) {
	//输入图像，flag==0表示灰度图，flag==1表示彩色图
	Mat image = imread(src, flag);


	Mat res = image.clone();

	//均值滤波
	if (model == "mean") {
		blur(image, res, Size(x, y));
	}
	//高斯滤波
	else if (model == "Gaussi") {
		GaussianBlur(image, res, Size(x, y), 1);
	}
	else if (model == "Laplacian") {
		//拉普拉斯锐化卷积核 0 -1  0 -1 4 -1 0 -1 0
		Mat kernel = (Mat_<float>(3, 3) << 0, -1, 0, -1, 4, -1, 0, -1,
0);
		//2D卷积
		filter2D(image, res, image.depth(), kernel);
	}
	else if (model == "Robert") {
		if (flag == 0) {
			//当处理灰度图像时
			for (int i = 0; i < image.rows - 1; i++) {
				for (int j = 0; j < image.cols - 1; j++) {
					//根据robert算子公式计算
					res.at<uchar>(i, j) = (uchar)sqrt(
						pow((image.at<uchar>(i, j) -
image.at<uchar>(i + 1, j + 1)), 2) +
						pow((image.at<uchar>(i + 1, j)
- image.at<uchar>(i, j + 1)), 2)
					);
				}
			}
		}
		else if (flag == 1) {
			// 当处理彩色图像时
			for (int i = 0; i < image.rows - 1; i++) {
				for (int j = 0; j < image.cols - 1; j++) {
					//根据robert算子公式计算
					for (int k = 0; k < 3; k++) {
						res.at<Vec3b>(i, j)[k] =
(uchar)sqrt(
							pow((image.at<Vec3b>(i,
j)[k] - image.at<Vec3b>(i + 1, j + 1)[k]), 2) +
							pow((image.at<Vec3b>(i +
1, j)[k] - image.at<Vec3b>(i, j + 1)[k]), 2)
						);
					}
				}
			}
		}
	}
	else if (model == "Sobel") {
		Mat grad_x, grad_y;
		Mat abs_grad_x, abs_grad_y;
```

```
                //求X方向梯度
                Sobel(image, grad_x, image.depth(), 1, 0, 3, 1, 1,
BORDER_DEFAULT);
                convertScaleAbs(grad_x, abs_grad_x);
                //imshow("X-Sobel", abs_grad_x);
                //求Y方向梯度
                Sobel(image, grad_y, image.depth(), 0, 1, 3, 1, 1,
BORDER_DEFAULT);
                convertScaleAbs(grad_y, abs_grad_y);
                //imshow("Y-Sobel", abs_grad_y);
                //合并梯度(近似)
                addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0, res);
        }
        imshow("原图", image);
        imshow(model + "Filter", res);
        waitKey(0);
        destroyAllWindows();
}
```

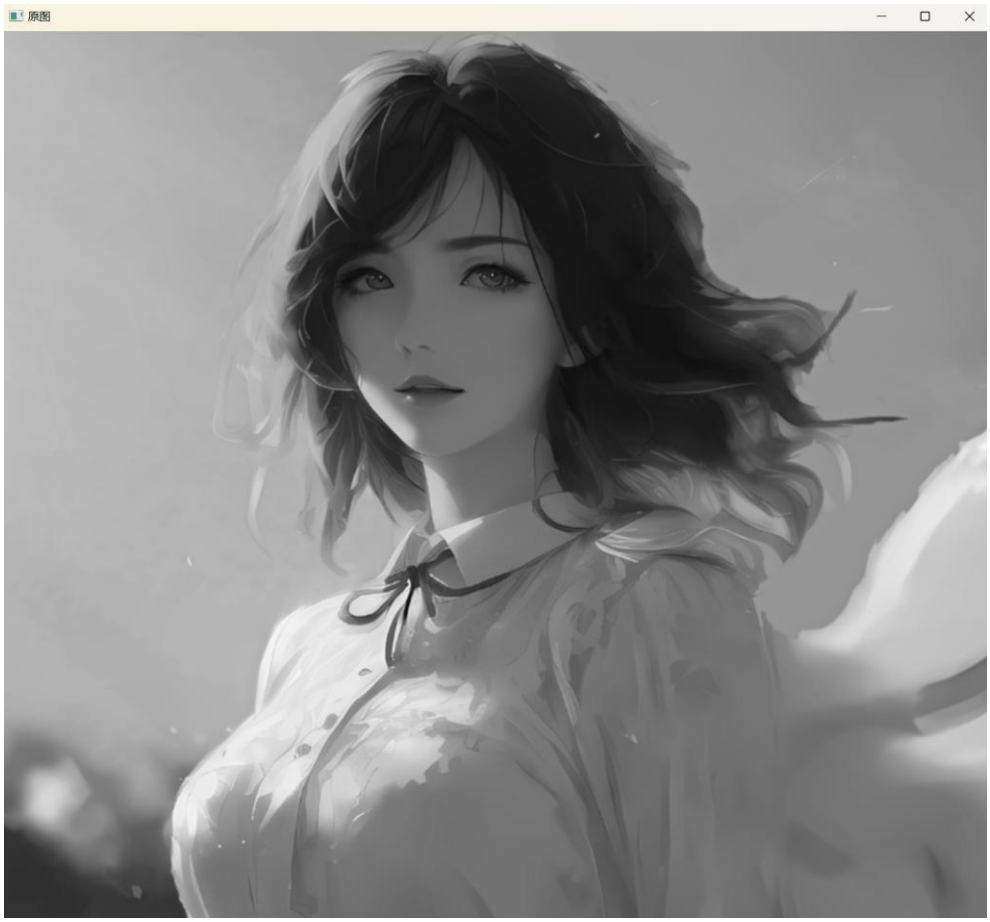
```
void EnhanceFilter(Mat img, Mat& dst, double dProportion, int nTempH, int
nTempW, int nTempMY, int nTempMX, float* pfArray, float fCoef) {


        int i, j, nHeight = img.rows, nWidth = img.cols;
        vector<vector<int>> GrayMat1, GrayMat2, GrayMat3;//暂存按比例叠加图
像，R,G,B三通道
        vector<int> vecRow1(nWidth, 0), vecRow2(nWidth, 0), vecRow3(nWidth,
0);
        for (i = 0; i < nHeight; i++)
        {
                GrayMat1.push_back(vecRow1);
                GrayMat2.push_back(vecRow2);
                GrayMat3.push_back(vecRow3);
        }

        //锐化图像，输出带符号响应，并与原图像按比例叠加
        for (i = nTempMY; i < nHeight - (nTempH - nTempMY) + 1; i++)
        {
                for (j = nTempMX; j < nWidth - (nTempW - nTempMX) + 1; j++)
                {
                        float fResult1 = 0;
                        float fResult2 = 0;
                        float fResult3 = 0;
                        for (int k = 0; k < nTempH; k++)
                        {
                                for (int l = 0; l < nTempW; l++)
                                {
                                        //分别计算三通道加权和
                                        fResult1 += img.at<Vec3b>(i, j)[0] *
pfArray[k * nTempW + l];
                                        fResult2 += img.at<Vec3b>(i, j)[1] *
pfArray[k * nTempW + l];
                                        fResult3 += img.at<Vec3b>(i, j)[2] *
pfArray[k * nTempW + l];
```

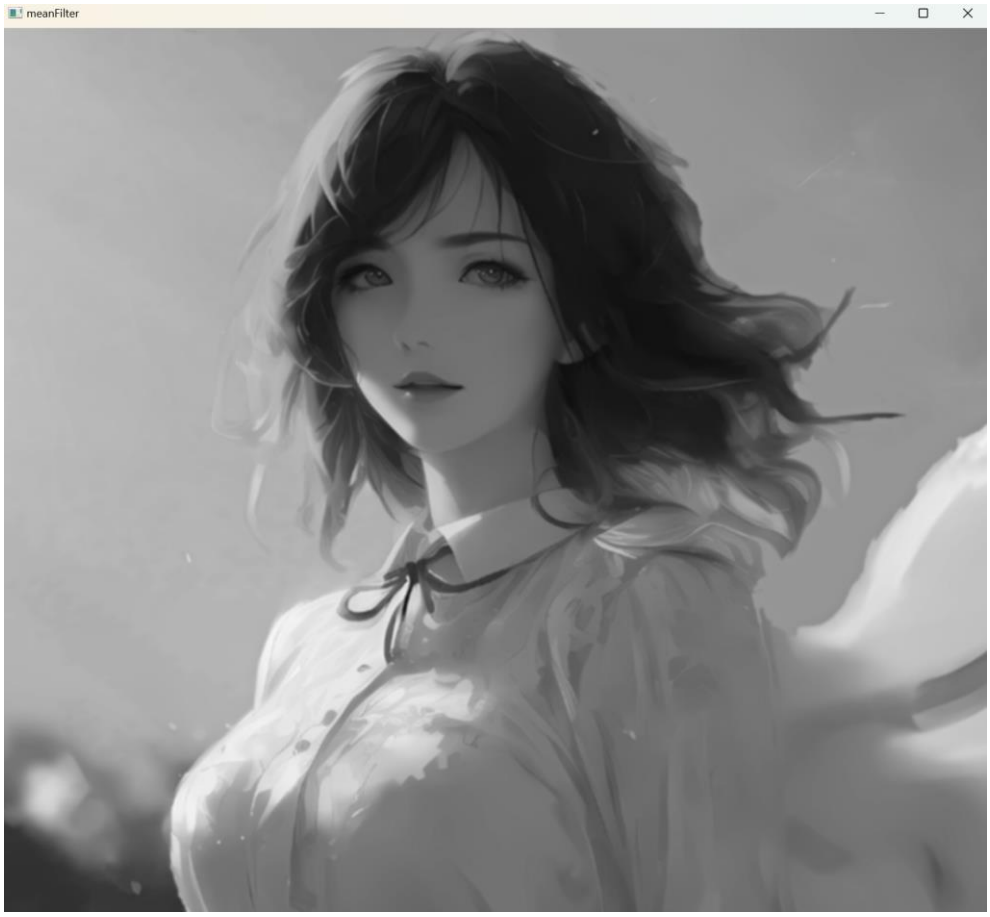
```
                        }
                    }

                    //三通道加权和分别乘以系数并限制响应范围，最后和原图像按比例混合
                    fResult1 *= fCoef;
                    if (fResult1 > 255)
                        fResult1 = 255;
                    if (fResult1 < -255)
                        fResult1 = -255;
                    GrayMat1[i][j] = dProportion * img.at<Vec3b>(i, j)[0] + fResult1 + 0.5;

                    fResult2 *= fCoef;
                    if (fResult2 > 255)
                        fResult2 = 255;
                    if (fResult2 < -255)
                        fResult2 = -255;
                    GrayMat2[i][j] = dProportion * img.at<Vec3b>(i, j)[1] + fResult2 + 0.5;

                    fResult3 *= fCoef;
                    if (fResult3 > 255)
                        fResult3 = 255;
                    if (fResult3 < -255)
                        fResult3 = -255;
                    GrayMat3[i][j] = dProportion * img.at<Vec3b>(i, j)[2] + fResult3 + 0.5;
                }
            }
            int nMax1 = 0, nMax2 = 0, nMax3 = 0;//三通道最大灰度和值
            int nMin1 = 65535, nMin2 = 65535, nMin3 = 65535;//三通道最小灰度和值
            //分别统计三通道最大值最小值
            for (i = nTempMY; i < nHeight - (nTempH - nTempMY) + 1; i++)
            {
                for (j = nTempMX; j < nWidth - (nTempW - nTempMX) + 1; j++)
                {
                    if (GrayMat1[i][j] > nMax1)
                        nMax1 = GrayMat1[i][j];
                    if (GrayMat1[i][j] < nMin1)
                        nMin1 = GrayMat1[i][j];

                    if (GrayMat2[i][j] > nMax2)
                        nMax2 = GrayMat2[i][j];
                    if (GrayMat2[i][j] < nMin2)
                        nMin2 = GrayMat2[i][j];

                    if (GrayMat3[i][j] > nMax3)
                        nMax3 = GrayMat3[i][j];
                    if (GrayMat3[i][j] < nMin3)
                        nMin3 = GrayMat3[i][j];
                }
            }
            //将按比例叠加后的三通道图像取值范围重新归一化到[0,255]
            int nSpan1 = nMax1 - nMin1, nSpan2 = nMax2 - nMin2, nSpan3 = nMax3 - nMin3;
            for (i = nTempMY; i < nHeight - (nTempH - nTempMY) + 1; i++)
            {
```

```
                for (j = nTempMX; j < nWidth - (nTempW - nTempMX) + 1; j++)
                {
                        int br, bg, bb;
                        if (nSpan1 > 0)
                                br = (GrayMat1[i][j] - nMin1) * 255 / nSpan1;
                        else if (GrayMat1[i][j] <= 255)
                                br = GrayMat1[i][j];
                        else
                                br = 255;
                        dst.at<Vec3b>(i, j)[0] = br;

                        if (nSpan2 > 0)
                                bg = (GrayMat2[i][j] - nMin2) * 255 / nSpan2;
                        else if (GrayMat2[i][j] <= 255)
                                bg = GrayMat2[i][j];
                        else
                                bg = 255;
                        dst.at<Vec3b>(i, j)[1] = bg;

                        if (nSpan3 > 0)
                                bb = (GrayMat3[i][j] - nMin3) * 255 / nSpan3;
                        else if (GrayMat3[i][j] <= 255)
                                bb = GrayMat3[i][j];
                        else
                                bb = 255;
                        dst.at<Vec3b>(i, j)[2] = bb;
                }
        }
}



//为灰度图像的高提升滤波函数
void EnhanceFilterForGray(Mat img, Mat& dst, double dProportion, int nTempH,
int nTempW, int nTempMY, int nTempMX, float* pfArray, float fCoef) {
        int i, j, nHeight = img.rows, nWidth = img.cols;
        vector<vector<int>> GrayMat;//灰度图
        vector<int> vecRow(nWidth, 0);
        for (i = 0; i < nHeight; i++)
        {
                GrayMat.push_back(vecRow);
        }

        //锐化图像，输出带符号响应，并与原图像按比例叠加
        for (i = nTempMY; i < nHeight - (nTempH - nTempMY) + 1; i++)
        {
                for (j = nTempMX; j < nWidth - (nTempW - nTempMX) + 1; j++)
                {
                        float fResult = 0;
                        for (int k = 0; k < nTempH; k++)
                        {
                                for (int l = 0; l < nTempW; l++)
                                {
                                        fResult += img.at<uchar>(i, j) *
pfArray[k * nTempW + l];
                                }
                        }
```

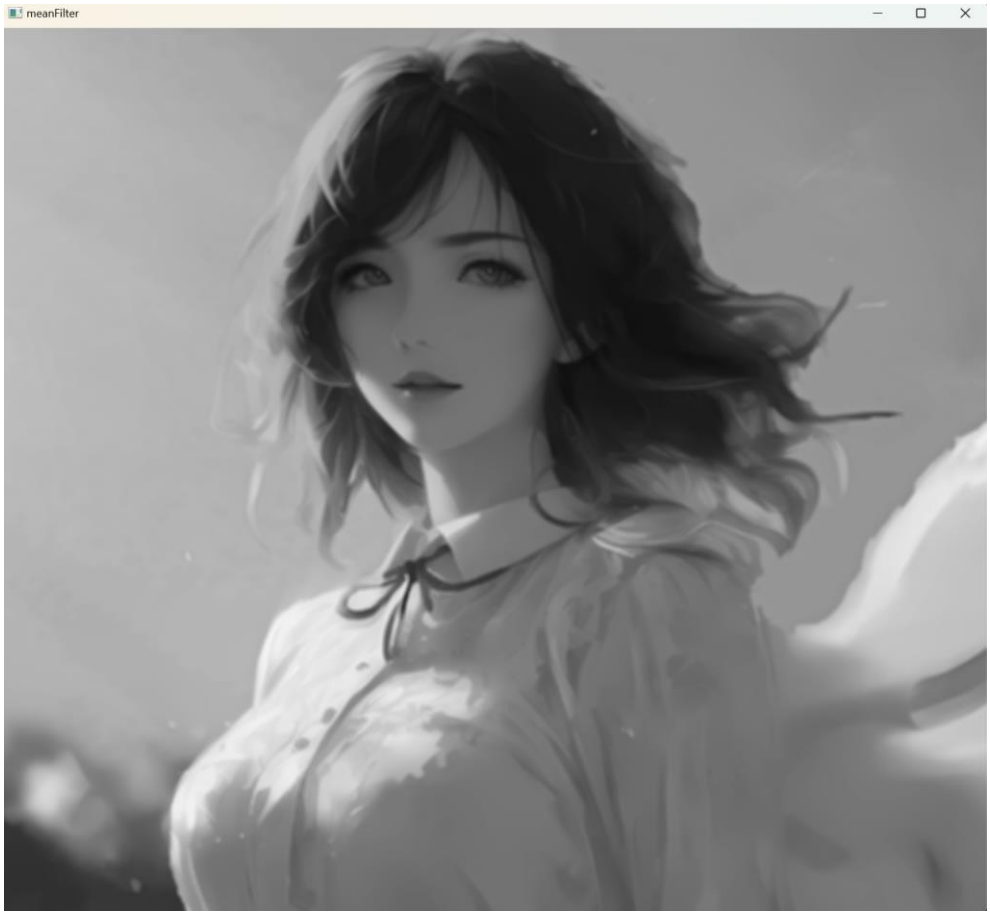
```
                        //乘以系数并限制响应范围，最后和原图像按比例混合
                        fResult *= fCoef;
                        if (fResult > 255)
                                fResult = 255;
                        if (fResult < -255)
                                fResult = -255;
                        GrayMat[i][j] = dProportion * img.at<uchar>(i, j) +
fResult + 0.5;
                }
        }
        int nMax = 0;//最大灰度和值
        int nMin = 65535;//最小灰度和值
        //统计最大值最小值
        for (i = nTempMY; i < nHeight - (nTempH - nTempMY) + 1; i++)
        {
                for (j = nTempMX; j < nWidth - (nTempW - nTempMX) + 1; j++)
                {
                        if (GrayMat[i][j] > nMax)
                                nMax = GrayMat[i][j];
                        if (GrayMat[i][j] < nMin)
                                nMin = GrayMat[i][j];
                }
        }
        //将按比例叠加后的灰度图像取值范围重新归一化到[0,255]
        int nSpan = nMax - nMin;
        for (i = nTempMY; i < nHeight - (nTempH - nTempMY) + 1; i++)
        {
                for (j = nTempMX; j < nWidth - (nTempW - nTempMX) + 1; j++)
                {
                        int b;
                        if (nSpan > 0)
                                b = (GrayMat[i][j] - nMin) * 255 / nSpan;
                        else if (GrayMat[i][j] <= 255)
                                b = GrayMat[i][j];
                        else
                                b = 255;
                        dst.at<uchar>(i, j) = b;
                }
        }
}



void enhanceFilter(string& src, int flag) {
        Mat img = imread(src, flag);
        imshow("原图", img);
        Mat dst = img.clone();
        //常用滤波模板数组
        //平均平滑1/9
        float Template_Smooth_Avg[9] = {
                1, 1, 1,
                1, 1, 1,
                1, 1, 1
        };
        //Gauss平滑1/16
        float Template_Smooth_Gauss[9] = {
```

```
                1, 2, 1,
                2, 4, 2,
                1, 2, 1
        };
        //Sobel垂直边缘检测
        float Template_Smooth_HSobel[9] = {
                -1, 0, 1,
                -2, 0, 2,
                -1, 0, 1
        };
        //Sobel水平边缘检测
        float Template_Smooth_VSobel[9] = {
                -1, -2, -1,
                 0,  0,  0,
                 1,  2,  1
        };
        //LOG边缘检测
        float Template_Log[25] = {
                 0,  0, -1,  0,  0,
                 0, -1, -2, -1,  0,
                -1, -2, 16, -2, -1,
                 0, -1, -2, -1,  0,
                 0,  0, -1,  0,  0
        };
        //Laplacian边缘检测
        float Template_Laplacian1[9] = {
                  0, -1,  0,
                 -1,  4, -1,
                  0, -1,  0
        };//对90度各向同性
        float Template_Laplacian2[9] = {
                -1, -1, -1,
                -1,  8, -1,
                -1, -1, -1
        };//对45度各向同性
        /*************************************************************************
*****************************************
        高提升滤波
        dProportion：高提升滤波中原图像的混合比例
        nTempH：模板高度，nTempW：模板宽度
        nTempMY：模板中心元素坐标，nTempMX：模板中心元素坐标
        fpArray：指向模板数组的指针，可以选取不同模板实现不同滤波的高提升版本
        fCoef：模板系数
        *************************************************************************
*****************************************/
        if (flag == 1) {
                EnhanceFilter(img, dst, 1.8, 3, 3, 1, 1, Template_Laplacian2,
1);
        }
        else {
                EnhanceFilterForGray(img, dst, 1.8, 3, 3, 1, 1,
Template_Laplacian2, 1);
        }

        imshow("Enhance Laplacian", dst);
        waitKey(0);
        destroyAllWindows();
}
```

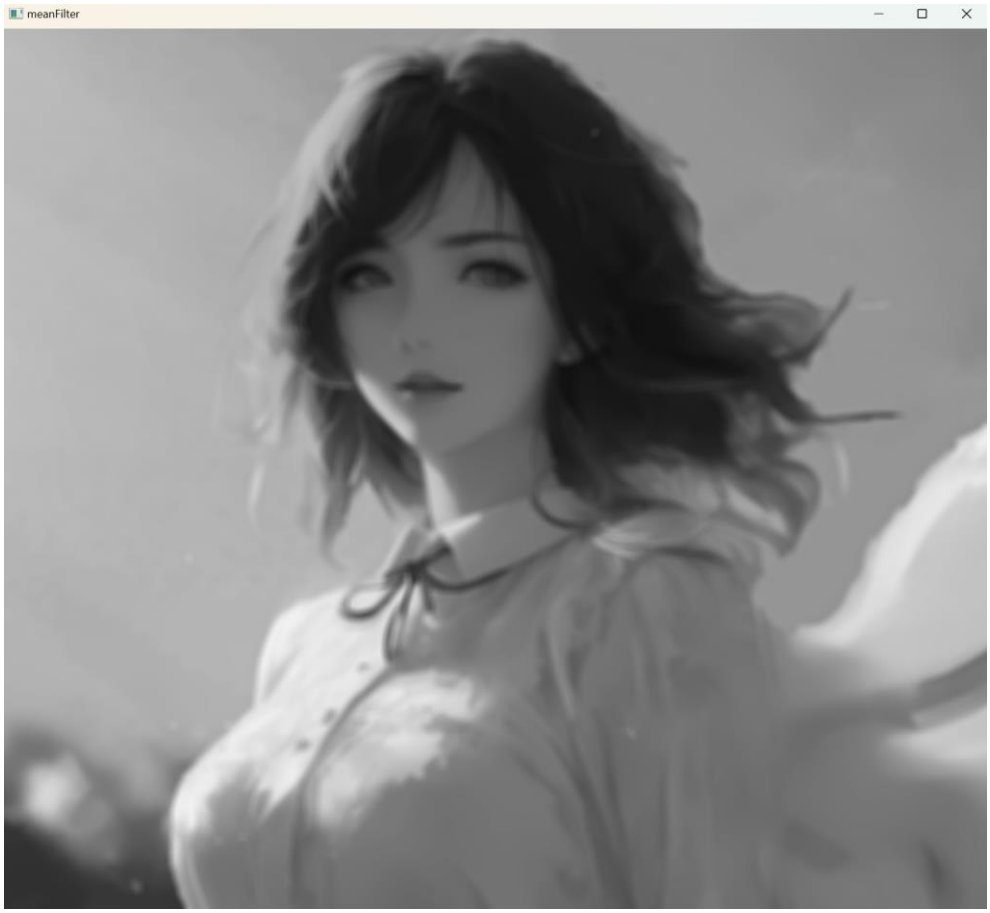
实验中的问题 (包括在实验中遇到的问题，以及解决问题的方法)	1、在利用不同的模板对灰色和彩色图像进行处理的时候，要对图像像素进行操作，相比于灰度图像的平滑，彩色图像的平滑就是分别对图像像素的 RGB 三个通道进行操作。 2、均值和高斯模板调用 cv 库，参考网上的高提升滤波算法。
实验结果 (实验完成后的源码和打包文件的说明)	 均值 3*3：



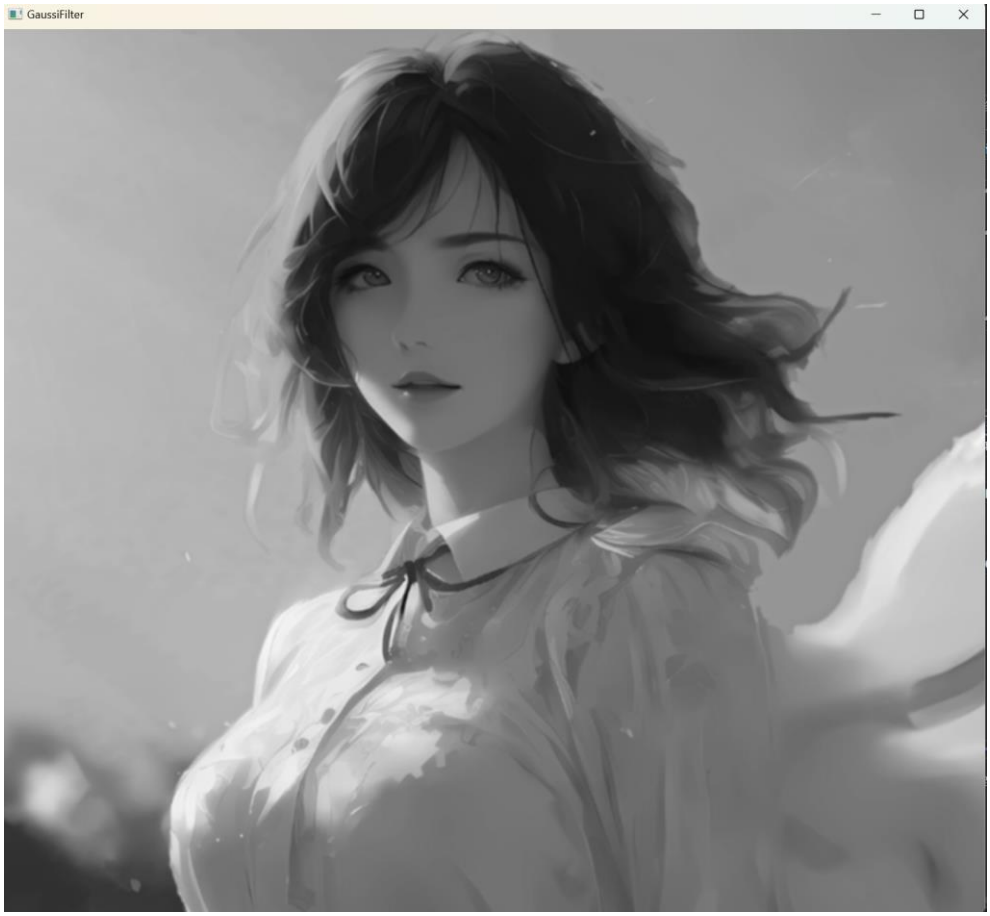
均值 5*5：



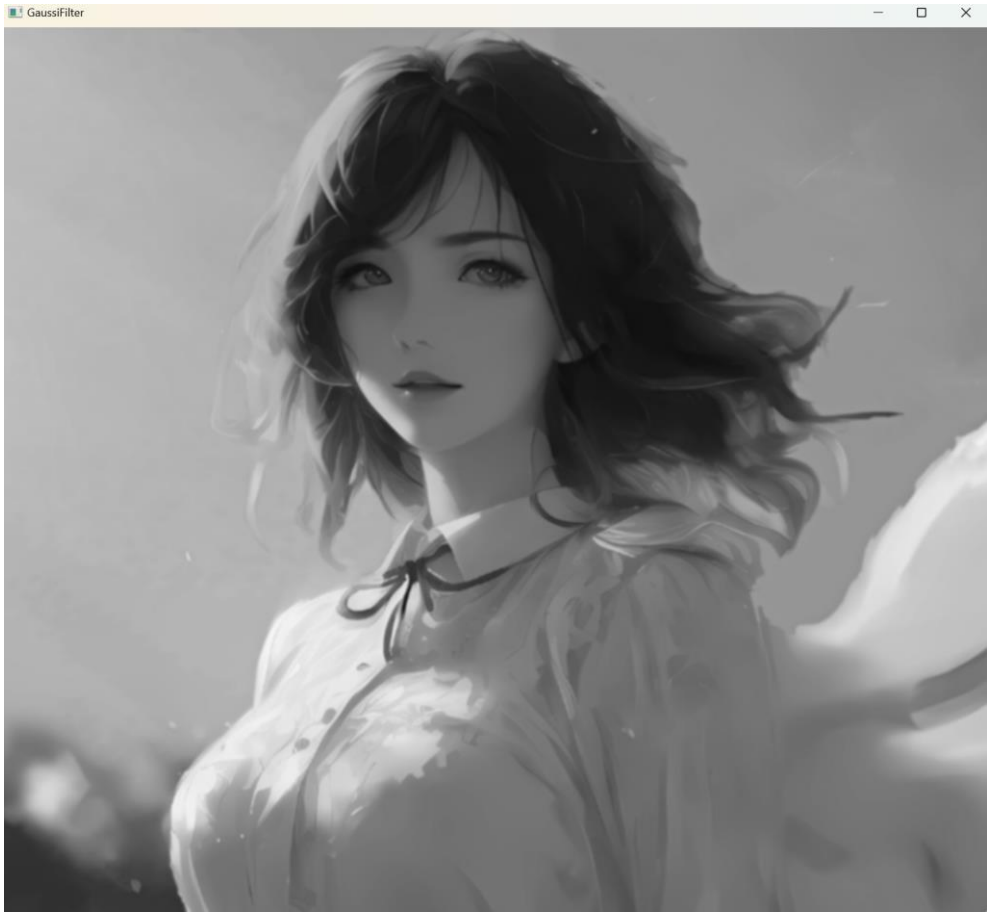
均值 9*9：



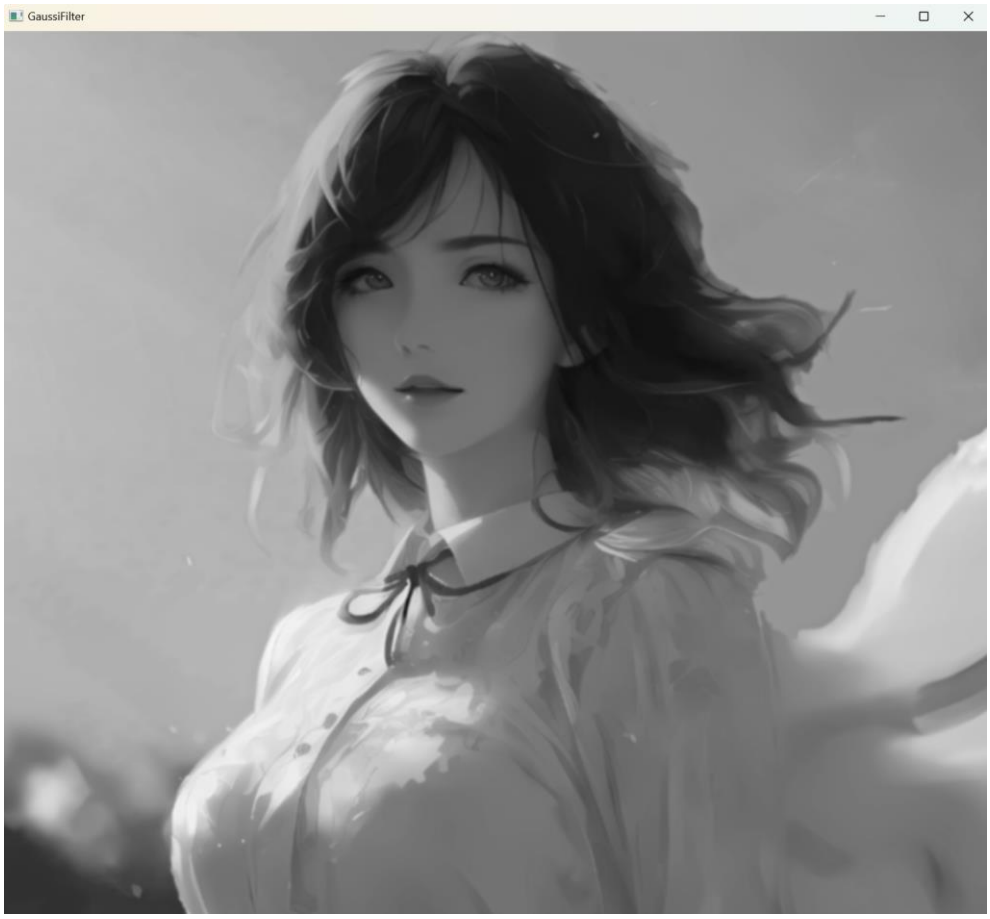
高斯 3*3：

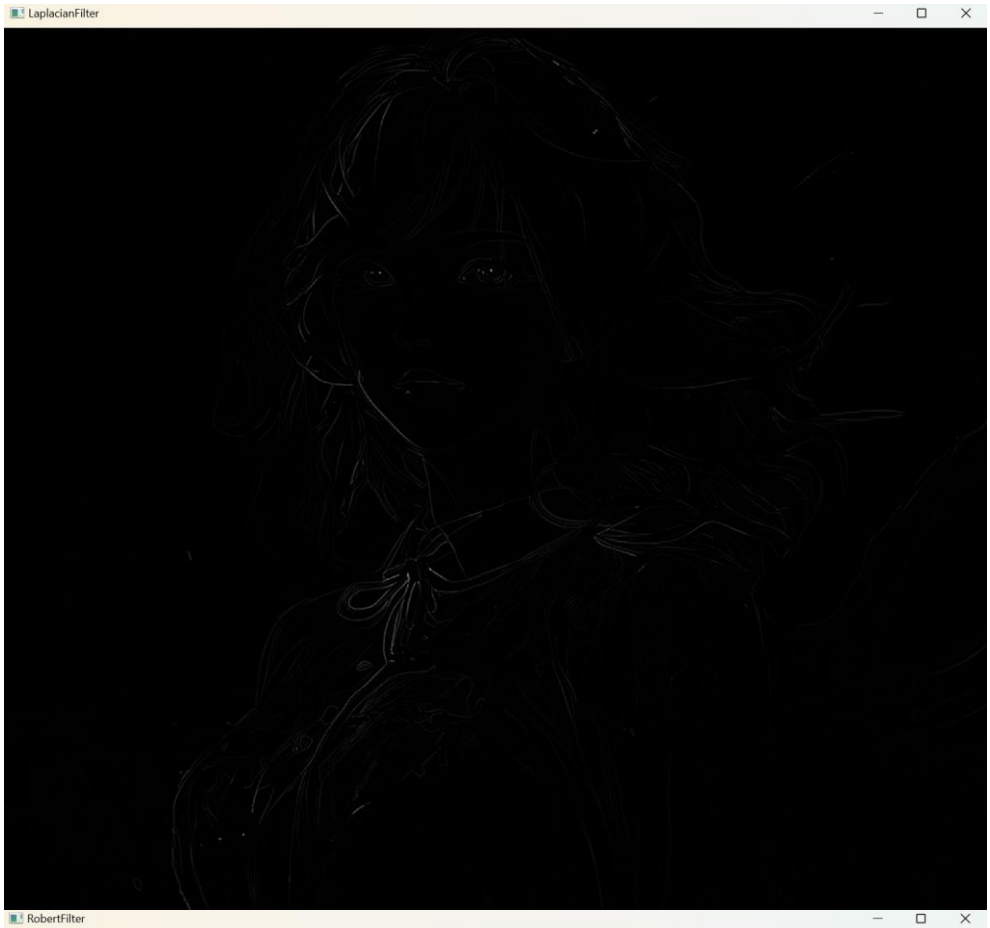


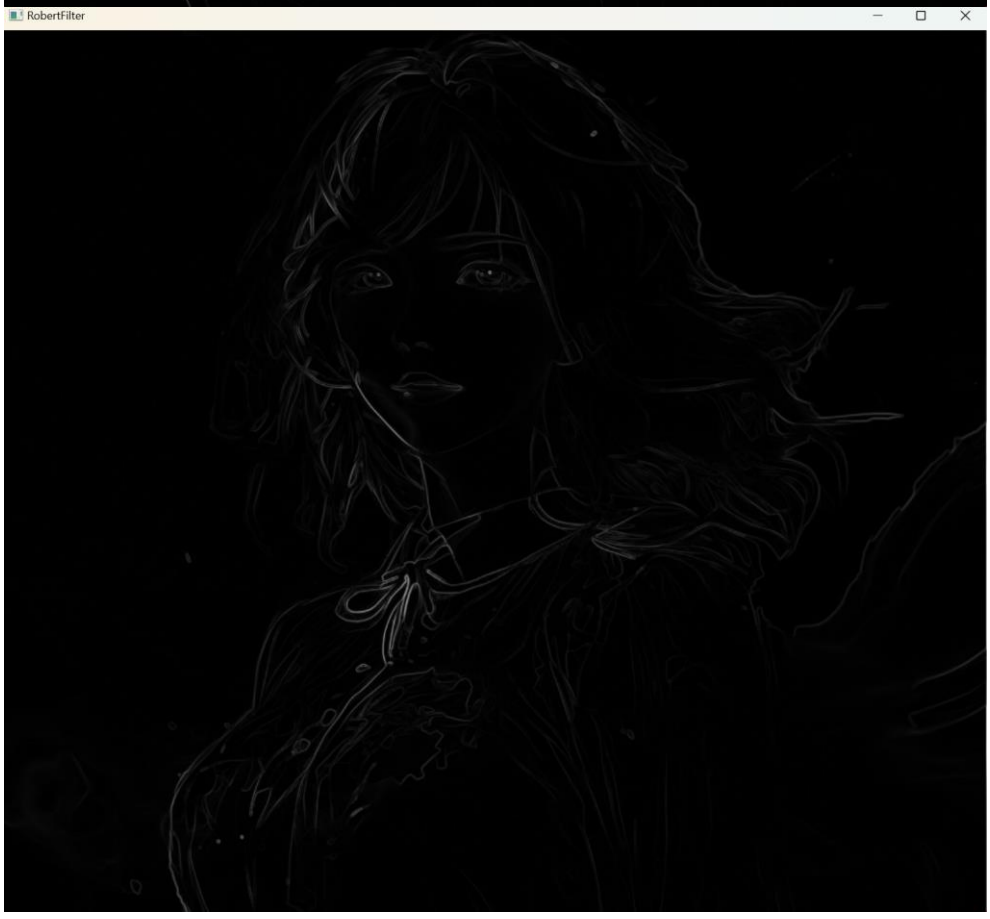
高斯 5*5：

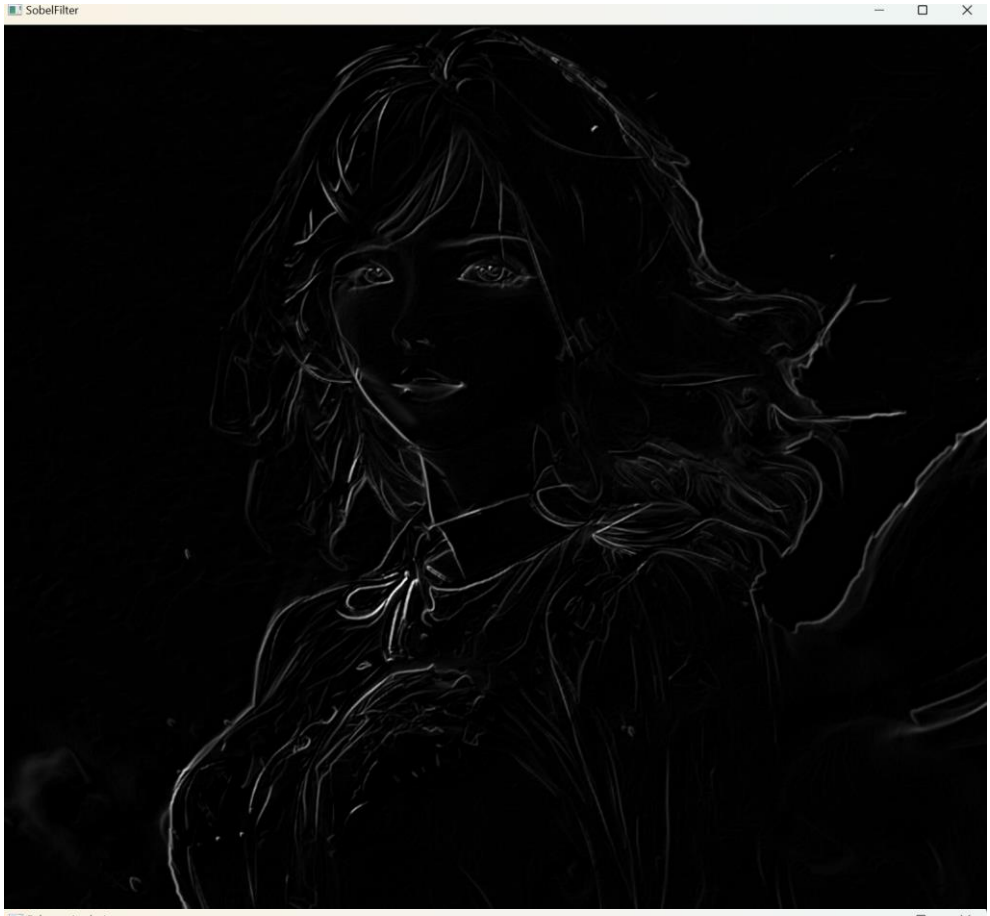


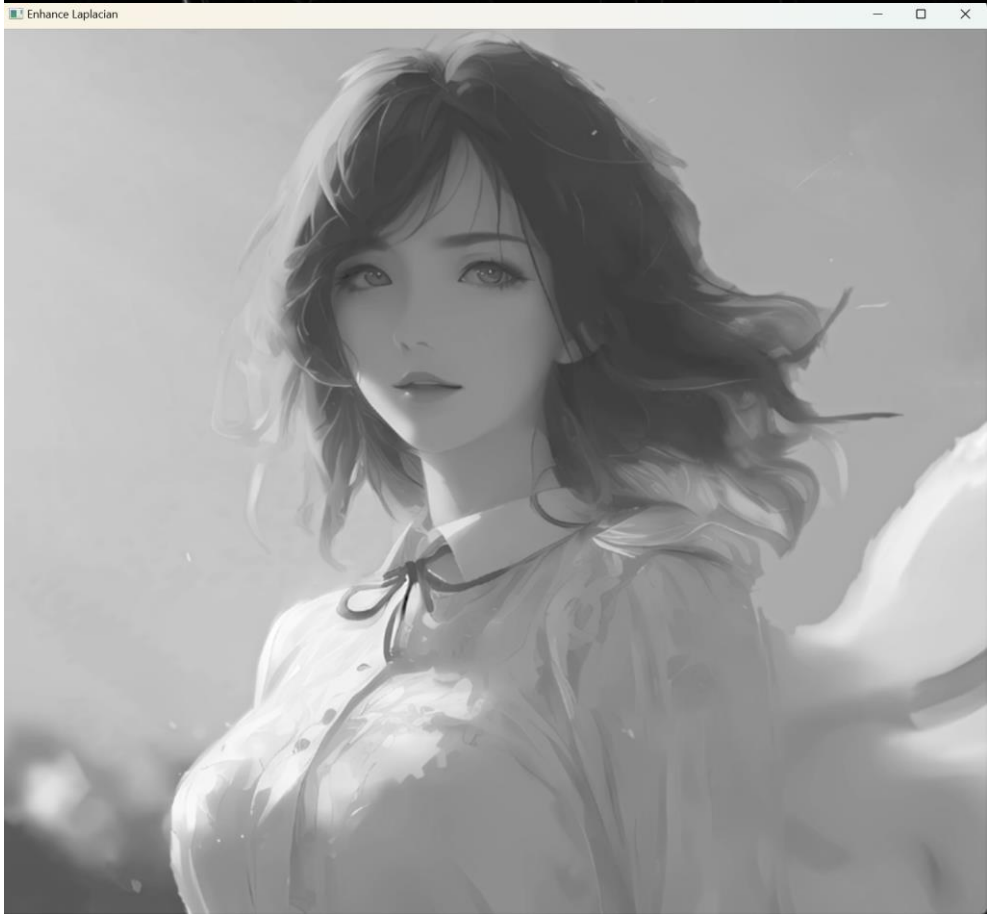
高斯 9*9：



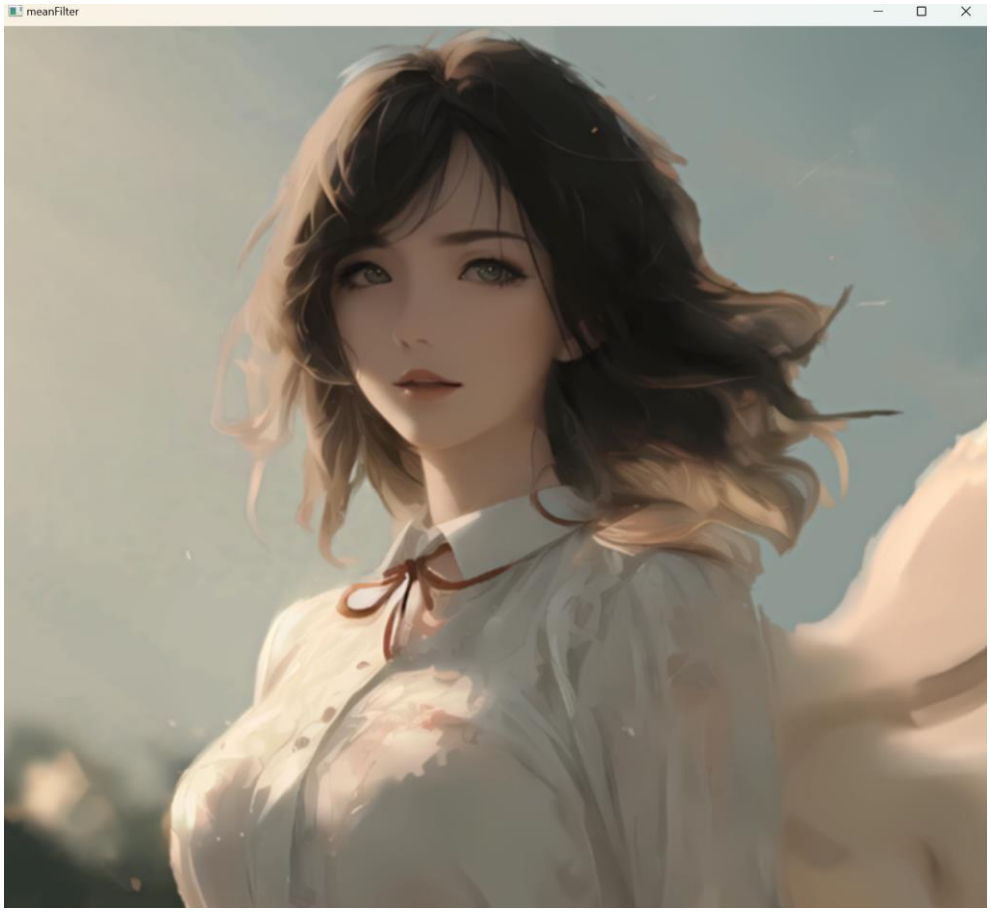

LaplacianFilter


RobertFilter

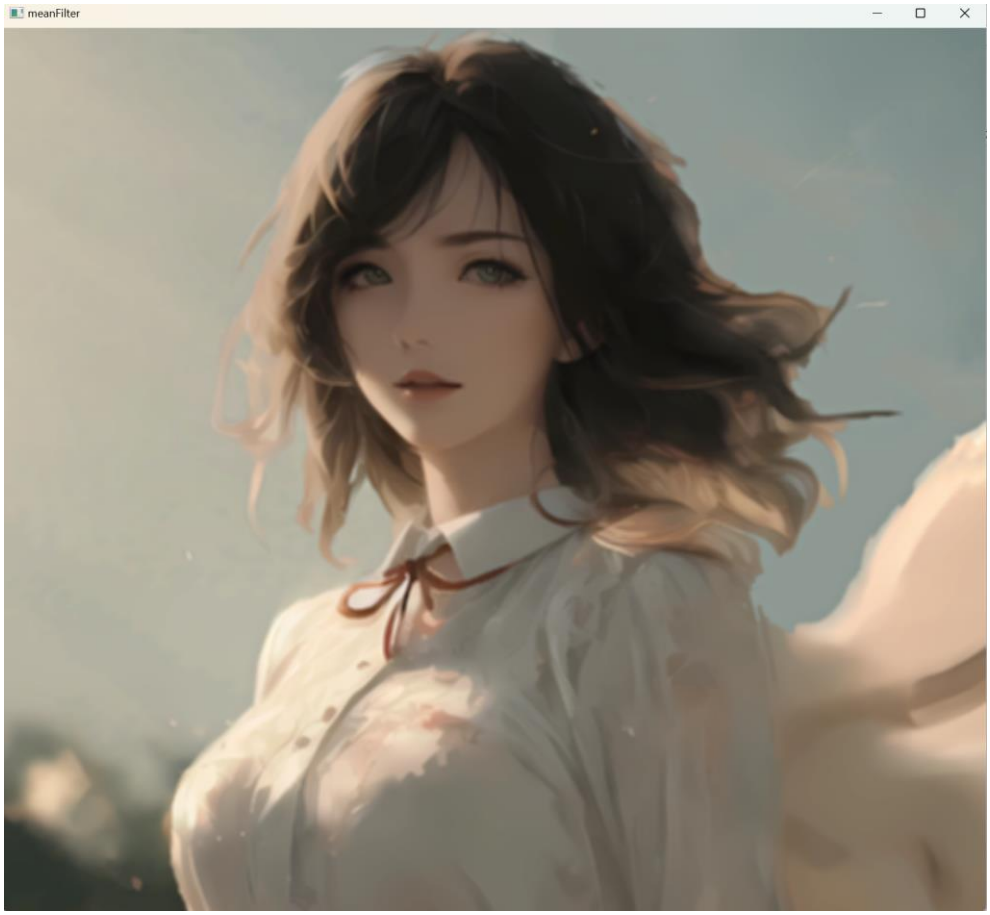

SobelFilter


Enhance Laplacian

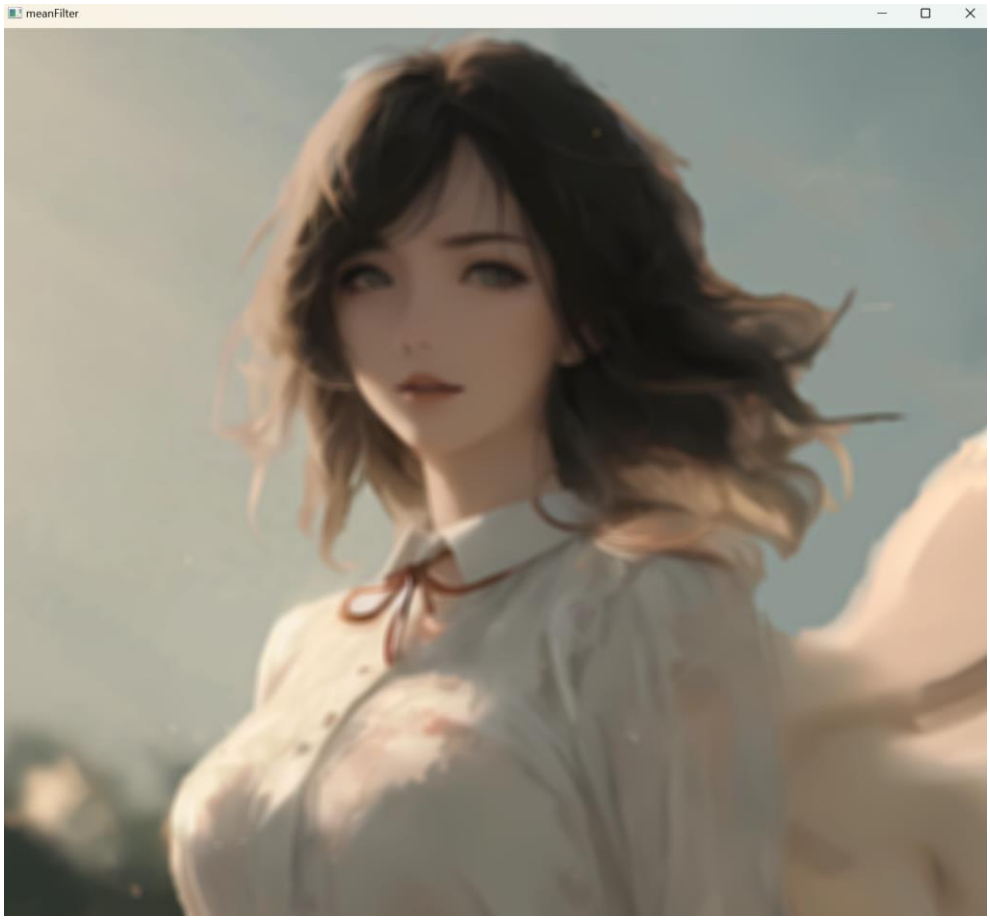
均值 3*3：



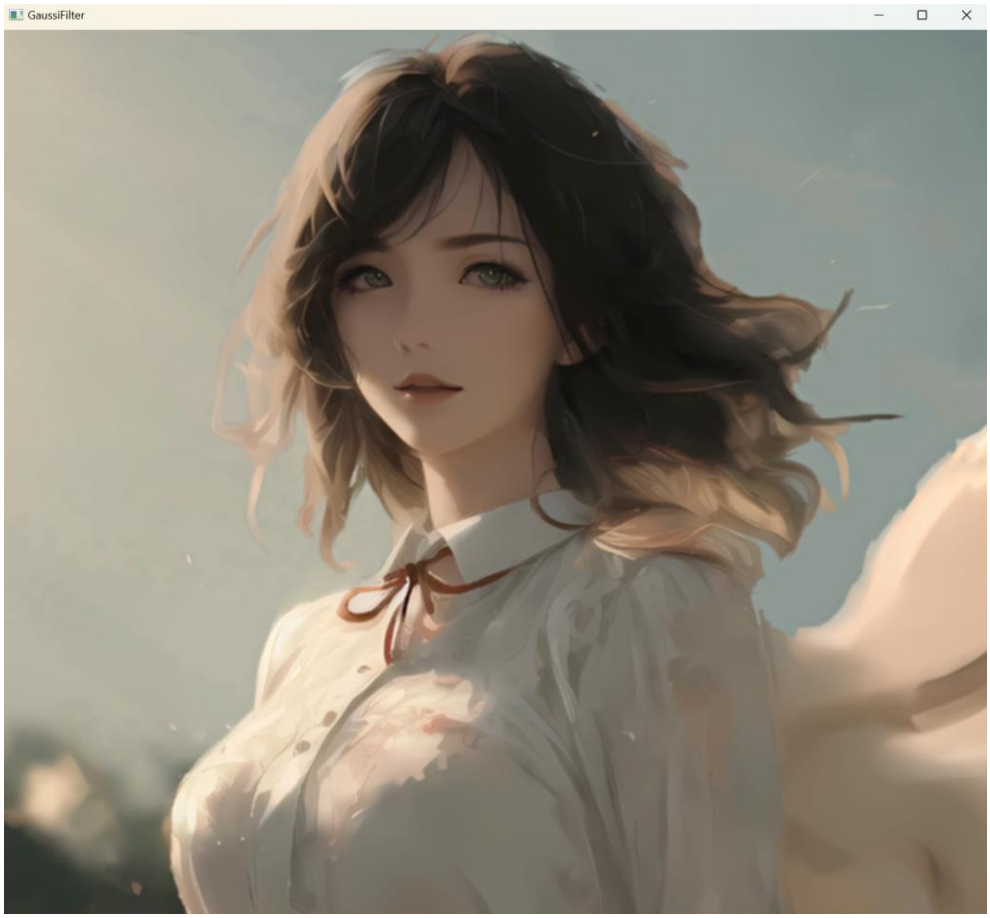
均值 5*5：



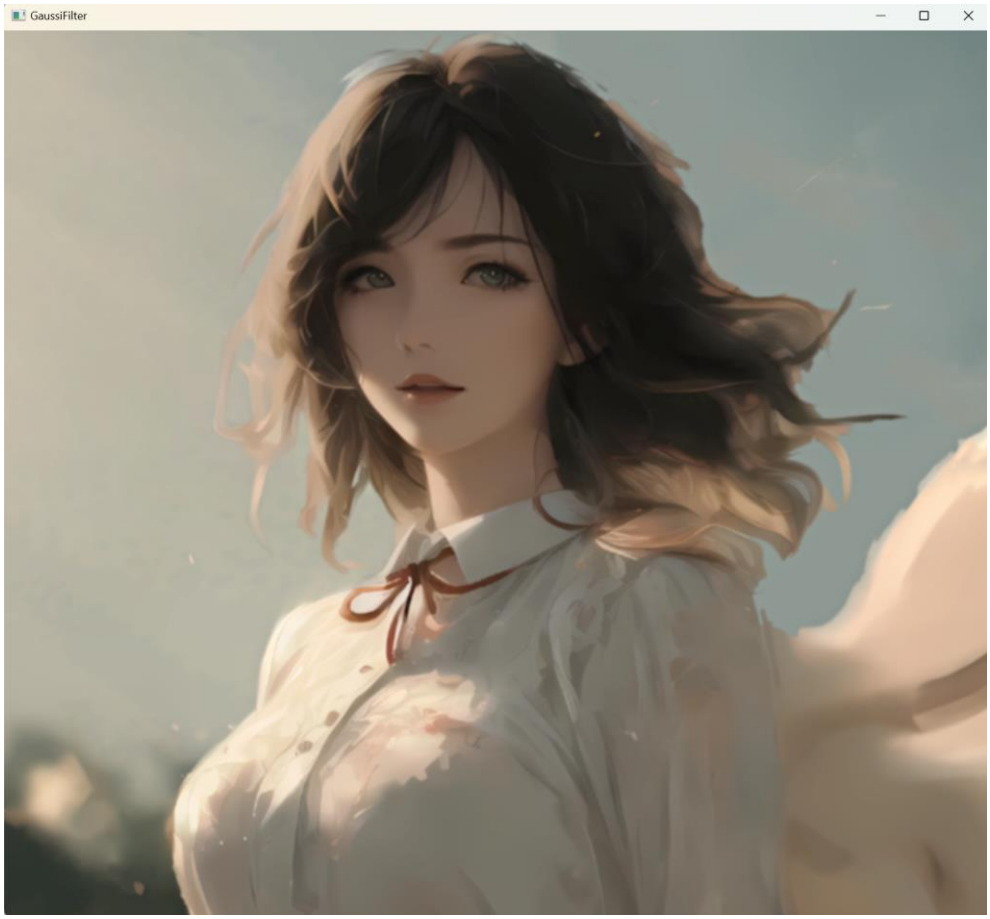
均值 9*9：



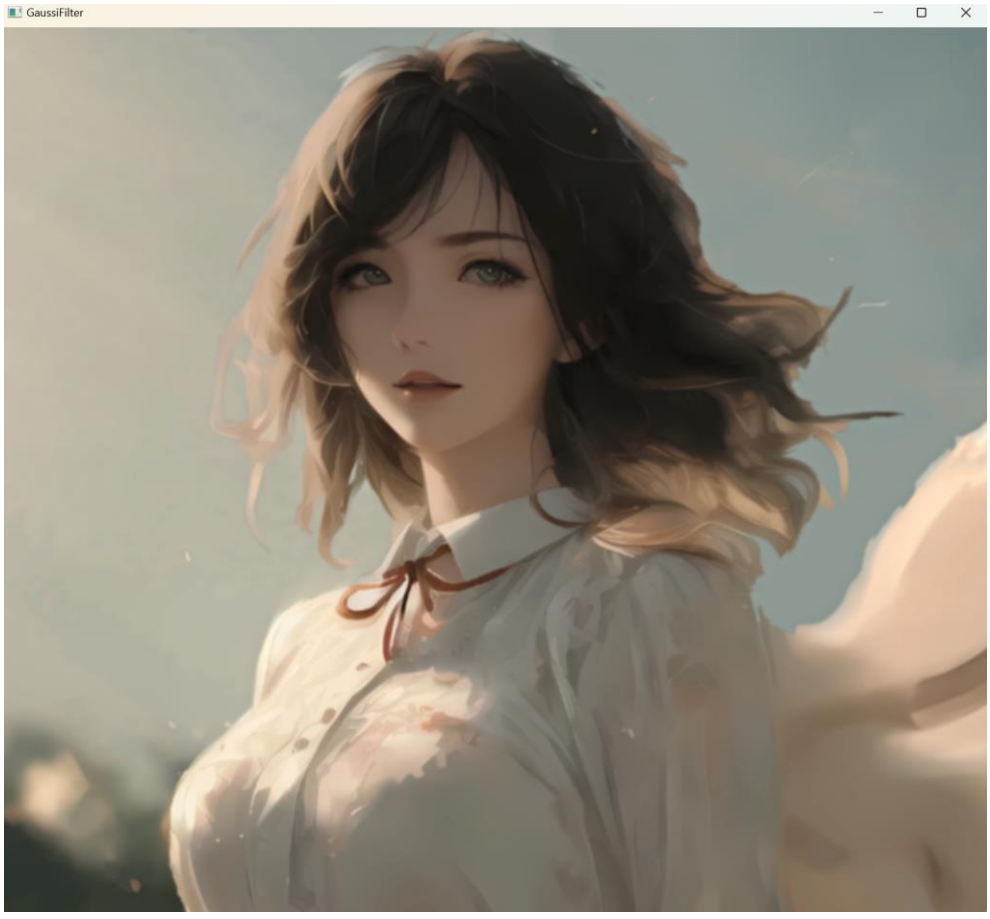
高斯 3*3：

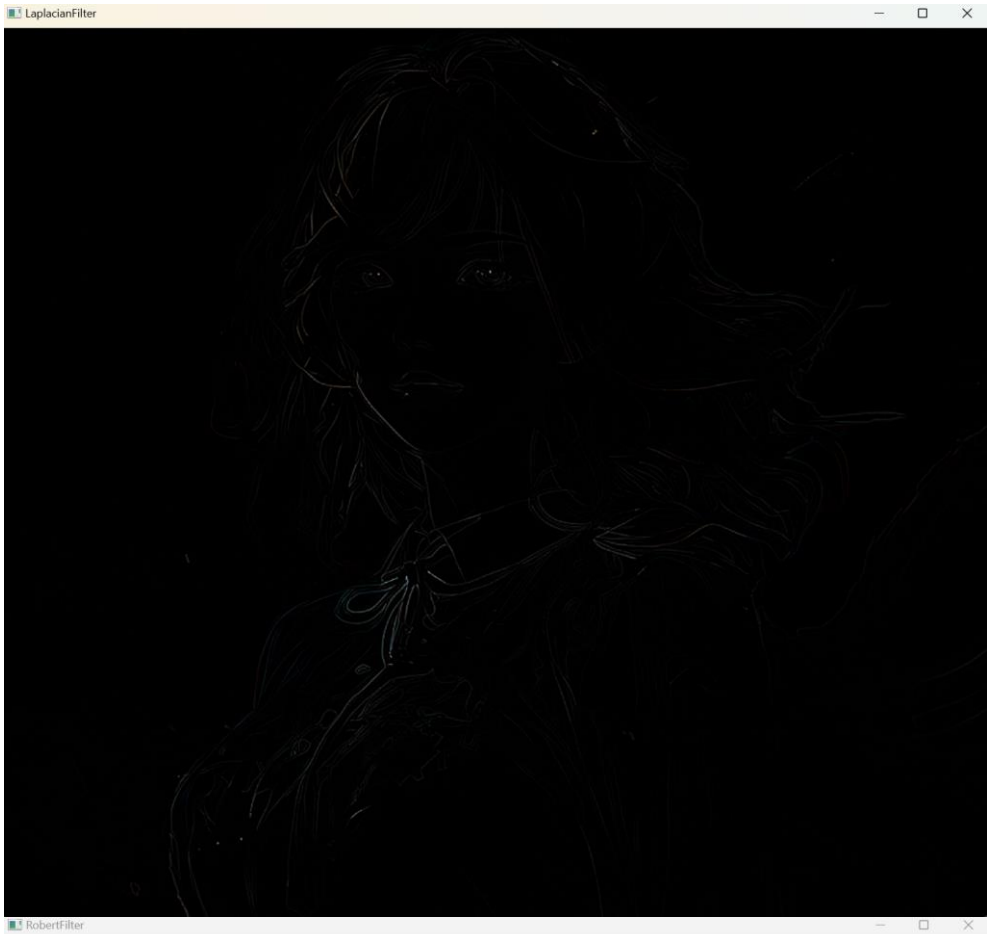


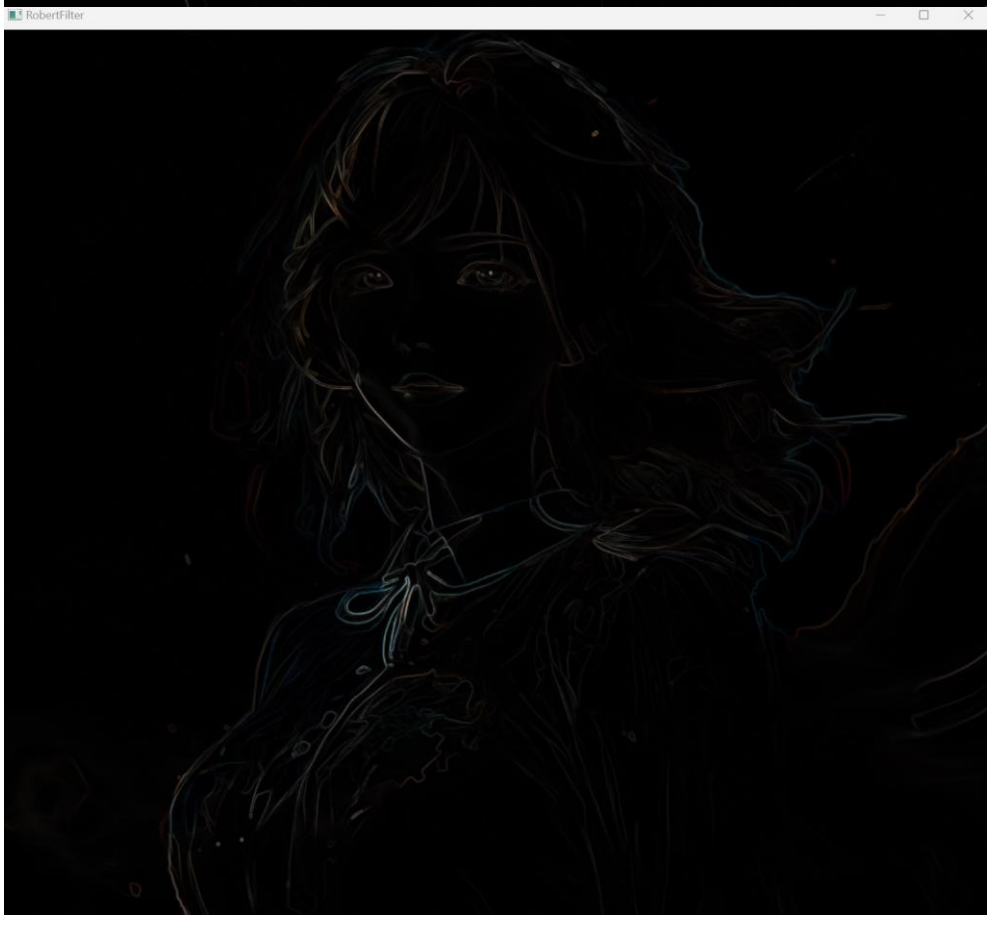
高斯 5*5：

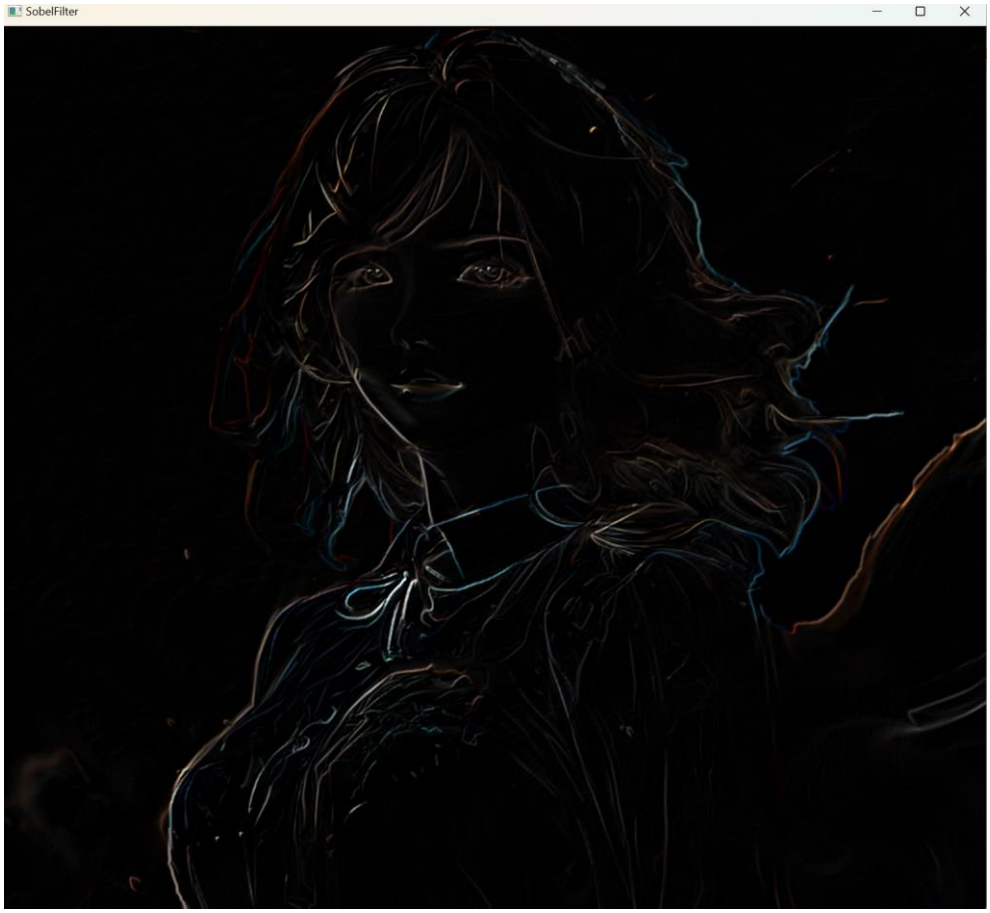


高斯 9*9：




LaplacianFilter


RobertFilter


SobelFilter