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：2023.11.13

实验名称	图像去噪
实验内容	1、均值滤波 具体内容：利用 OpenCV 对灰度图像像素进行操作，分别利用算术均值滤 波器、几何均值滤波器、谐波和逆谐波均值滤波器进行图像去噪。模板大小为 5*5。（注：请分别为图像添加高斯噪声、胡椒噪声、盐噪声和椒盐噪声，并观察 滤波效果） 2、中值滤波 具体内容：利用 OpenCV 对灰度图像像素进行操作，分别利用 5*5 和 9*9 尺寸的模板对图像进行中值滤波。（注：请分别为图像添加胡椒噪声、盐噪声和 椒盐噪声，并观察滤波效果） 3、自适应均值滤波。 具体内容：利用 OpenCV 对灰度图像像素进行操作，设计自适应局部降 低噪声滤波器去噪算法。模板大小 7*7（对比该算法的效果和均值滤波器的效果） 4、自适应中值滤波 具体内容：利用 OpenCV 对灰度图像像素进行操作，设计自适应中值滤波算 法对椒盐图像进行去噪。模板大小 7*7（对比中值滤波器的效果） 5、彩色图像均值滤波 具体内容：利用 OpenCV 对彩色图像 RGB 三个通道的像素进行操作，利用算术均值滤波器和几何均值滤波器进行彩色图像去噪。模板大小为 5*5。
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）	/* 均值滤波 * 具体内容：利用 OpenCV 对灰度图像像素进行操作，分别利用算术均值滤波器、几何均值滤波器、谐波和逆谐波均值滤波器进行图像去噪。 * 模板大小为 5*5。（注：请分别为图像添加高斯噪声、胡椒噪声、盐噪声和椒盐噪声，并观察 滤波效果） * */ //算数均值 Mat digitalMeanFilter(Mat& image, int size) { Mat dst = image.clone(); int rows = image.rows, cols = image.cols; int start = size / 2; for (int m = start; m < rows - start; m++) { for (int n = start; n < cols - start; n++) {

```
                        //灰色图
                        if (dst.channels() == 1) {
                                int sum = 0;
                                for (int i = m - start; i <= m + start; i++) {
                                        for (int j = n - start; j <= n + start;
j++) {
                                                //求和
                                                sum += dst.at<uchar>(i, j);
                                        }
                                }
                                //取平均
                                dst.at<uchar>(m, n) = uchar(sum / size / size);
                        }
                        //彩色图
                        else {
                                Vec3b pixel;
                                int sum1[3] = { 0 };
                                for (int i = m - start; i <= m + start; i++) {
                                        for (int j = n - start; j <= n + start;
j++) {
                                                pixel = dst.at<Vec3b>(i, j);
                                                for (int k = 0; k <
dst.channels(); k++) {
                                                        sum1[k] += pixel[k];
                                                }
                                        }

                                }
                                for (int k = 0; k < dst.channels(); k++) {
                                        pixel[k] = sum1[k] / size / size;
                                }
                                dst.at<Vec3b>(m, n) = pixel;
                        }
                }
        }
        return dst;
}


//几何均值
Mat geometryMeanFilter(Mat& image, int size)
{
        Mat dst = image.clone();
        int row, col;
        int h = image.rows;
        int w = image.cols;
        double mul;
        double dc;
        int mn;
        //计算每个像素的去噪后 color 值
        for (int i = 0; i < image.rows; i++) {
                for (int j = 0; j < image.cols; j++) {
                        int left = -size / 2;
                        int right = size / 2;
                        //灰度图
                        if (image.channels() == 1) {
                                mul = 1.0;
                                mn = 0;
```

```
                                //统计邻域内的几何平均值，邻域大小 5*5
                                for (int m = left; m <= right; m++) {
                                        row = i + m;
                                        for (int n = left; n <= right; n++) {
                                                col = j + n;
                                                if (row >= 0 && row < h &&
col >= 0 && col < w) {
                                                        //邻域内的非零像素点相
乘，最小值设定为1
                                                        int s =
image.at<uchar>(row, col);
                                                        mul = mul * (s == 0 ?
1 : s);
                                                        mn++;
                                                }
                                        }
                                }
                                //计算开 mn 次根号
                                dc = pow(mul, 1.0 / mn);
                                //统计成功赋给去噪后图像。
                                int res = (int)dc;
                                dst.at<uchar>(i, j) = res;
                        }
                        //彩色图
                        else {
                                double multi[3] = { 1.0,1.0,1.0 };
                                mn = 0;
                                Vec3b pixel;
                                for (int m = left; m <= right; m++) {
                                        row = i + m;
                                        for (int n = left; n <= right; n++) {
                                                col = j + n;
                                                if (row >= 0 && row < h &&
col >= 0 && col < w) {
                                                        pixel =
image.at<Vec3b>(row, col);
                                                        for (int k = 0; k <
image.channels(); k++) {
                                                                //邻域内的非零像
素点相乘，最小值设定为1
                                                                multi[k] =
multi[k] * (pixel[k] == 0 ? 1 : pixel[k]);
                                                        }
                                                        mn++;
                                                }
                                        }
                                }
                                double d;
                                for (int k = 0; k < image.channels(); k++) {
                                        d = pow(multi[k], 1.0 / mn);
                                        pixel[k] = (int)d;
                                }
                                dst.at<Vec3b>(i, j) = pixel;
                        }
                }
        }
        return dst;
}
```

```
//谐波均值
Mat harmonicMeanFilter(Mat& image, int size)
{
	Mat dst = image.clone();
	int row, col;
	int h = image.rows;
	int w = image.cols;
	double sum;
	int mn;
	//计算每个像素的去噪后 color 值
	for (int i = 0; i < image.rows; i++) {
		for (int j = 0; j < image.cols; j++) {
			sum = 0.0;
			mn = 0;
			//统计邻域,5*5 模板
			int left = -size / 2;
			int right = size / 2;
			for (int m = left; m <= right; m++) {
				row = i + m;
				for (int n = left; n <= right; n++) {
					col = j + n;
					if (row >= 0 && row < h && col >= 0 &&
col < w) {
						int s = image.at<uchar>(row,
col);
						//如果是0，设定为255，否则除以s
						sum = sum + (s == 0 ? 255 :
255.0 / s);
						mn++;
					}
				}
			}
			//统计成功赋给去噪后图像。
			dst.at<uchar>(i, j) = mn * 255.0 / sum;
		}
	}
	return dst;
}

//逆谐波均值
Mat inverseHarmonicMeanFilter(Mat& image, int size, double Q) {
	Mat dst = image.clone();
	int row, col;
	int h = image.rows;
	int w = image.cols;
	double sum;
	double sum1;
	//计算每个像素的去噪后 color 值
	for (int i = 0; i < image.rows; i++) {
		for (int j = 0; j < image.cols; j++) {
			sum = 0.0;
			sum1 = 0.0;
			//统计邻域
			int left = -size / 2;
			int right = size / 2;
			for (int m = left; m <= right; m++) {
				row = i + m;
				for (int n = left; n <= right; n++) {
```

```
                                        col = j + n;
                                        if (row >= 0 && row < h && col >= 0 &&
col < w) {
                                                int s = image.at<uchar>(row,
col);
                                                sum = sum + pow(s, Q + 1);
                                                sum1 = sum1 + pow(s, Q);
                                        }
                                }
                        }
                        //统计成功赋给去噪后图像。
                        dst.at<uchar>(i, j) = sum1 == 0 ? 0 : (sum / sum1);
                }
        }
        return dst;
}

/*中值滤波
 * 具体内容：利用 OpenCV 对灰度图像像素进行操作，分别利用 5*5 和 9*9 尺寸的模
板对图像进行中值滤波。
 * （注：请分别为图像添加胡椒噪声、盐噪声和 椒盐噪声，并观察滤波效果）
 * */
Mat MedianFilter(Mat& image, int size) {
        Mat dst = image.clone();
        int rows = image.rows, cols = image.cols;
        int start = size / 2;
        for (int m = start; m < rows - start; m++) {
                for (int n = start; n < cols - start; n++) {
                        //用来统计附近取值的向量
                        vector<uchar> model;
                        for (int i = m - start; i <= m + start; i++) {
                                for (int j = n - start; j <= n + start; j++) {
                                        model.push_back(dst.at<uchar>(i, j));
                                }
                        }
                        //采用快速排序进行
                        sort(model.begin(), model.end());
                        //取中值
                        dst.at<uchar>(m, n) = model[size * size / 2];
                }
        }
        return dst;
}

/*自适应均值滤波
 * 具体内容：利用 OpenCV 对灰度图像像素进行操作，设计自适应局部降 低噪声滤波
器去噪算法。
 * 模板大小 7*7（对比该算法的效果和均值滤波器的效果）
 * */
Mat selfAdaptMeanFilter(Mat& image, int size)
{
        Mat dst = image.clone();
        blur(image, dst, Size(size, size));
        int row, col;
        int h = image.rows;
        int w = image.cols;
        int mn;
        double Sl;
```

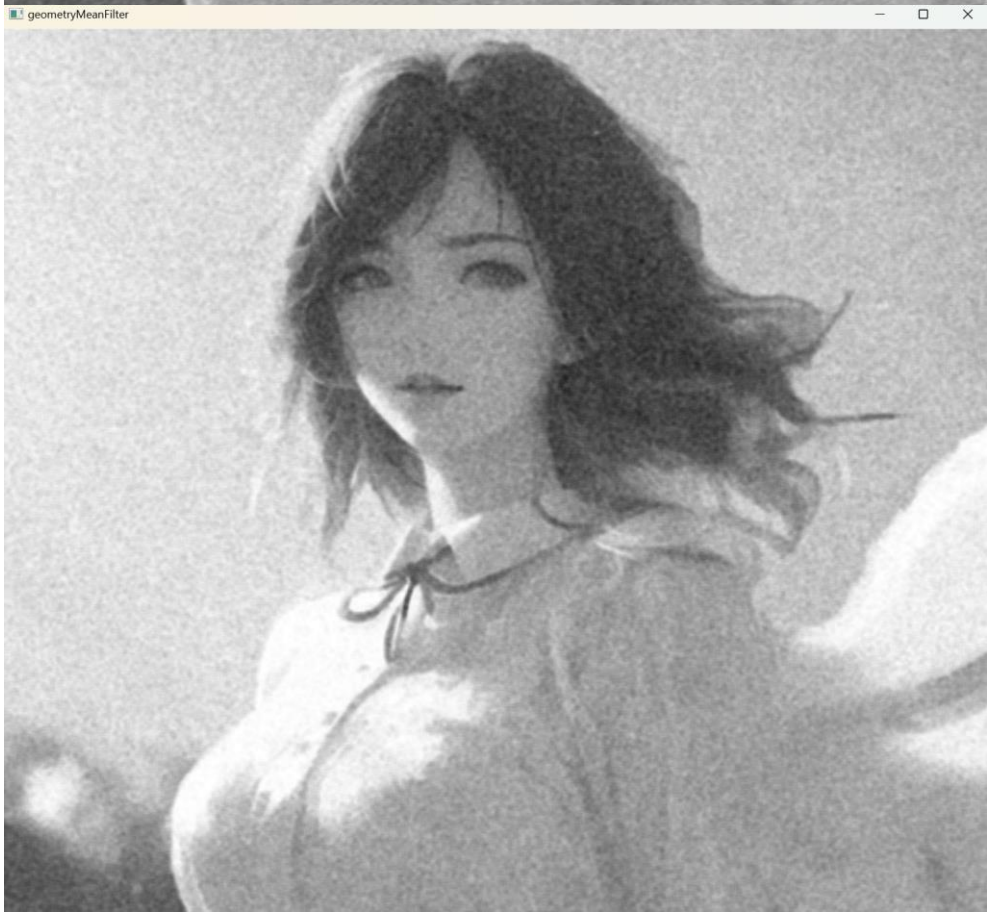
```
        double Sn = 100;
        for (int i = 0; i < image.rows; i++)
        {
                for (int j = 0; j < image.cols; j++)
                {
                        int Zxy = image.at<uchar>(i, j);
                        int Zmed = image.at<uchar>(i, j);
                        Sl = 0;
                        mn = 0;
                        int left = -size / 2;
                        int right = size / 2;
                        for (int m = left; m <= right; m++) {
                                row = i + m;
                                for (int n = left; n <= right; n++) {
                                        col = j + n;
                                        if (row >= 0 && row < h && col >= 0 &&
col < w) {
                                                int Sxy = image.at<uchar>(row,
col);
                                                Sl = Sl + pow(Sxy - Zmed, 2);
                                                mn++;
                                        }
                                }
                        }
                        Sl = Sl / mn;
                        dst.at<uchar>(i, j) = (int)(Zxy - Sn / Sl * (Zxy -
Zmed));
                }
        }
        return dst;
}

/*自适应中值滤波
 * 具体内容：利用 OpenCV 对灰度图像像素进行操作，设计自适应中值滤波算 法对椒盐图像进行去噪。
 * 模板大小 7*7（对比中值滤波器的效果）
 * */
Mat selfAdaptMedianFilter(Mat& image, int size) {
        Mat dst = image.clone();
        int row, col;
        //图像高与宽
        int h = image.rows;
        int w = image.cols;
        int Zmin, Zmax, Zmed, Zxy, Smax = size;
        int wsize;
        //计算每个像素的去噪后 color 值
        for (int i = 0; i < image.rows; i++) {
                for (int j = 0; j < image.cols; j++) {
                        //统计邻域
                        //最开始窗口大小为1
                        wsize = 1;
                        while (wsize <= size / 2) {
                                //当前像素值
                                Zxy = image.at<uchar>(i, j);
                                vector<uchar> pixels;
                                //在当前尺寸模板扫描
                                for (int m = -wsize; m <= wsize; m++) {
                                        row = i + m;
```

	`` for (int n = -wsize; n <= wsize; n++) { col = j + n; //保证模板游标不越界 if (row >= 0 && row < h && col >= 0 && col < w) { //模板游标当前指向的像素值 pixels.push_back(image.at<uchar>(row, col)); } } } sort(pixels.begin(), pixels.end()); Zmax = pixels[pixels.size() - 1]; Zmin = pixels[0]; Zmed = pixels[pixels.size() / 2]; //如果中值不超范围 if ((Zmed - Zmin) > 0 && (Zmed - Zmax) < 0) { //且当前像素值不超范围 if ((Zxy - Zmin) > 0 && (Zxy - Zmax) < 0) { dst.at<uchar>(i, j) = Zxy; } else { dst.at<uchar>(i, j) = Zmed; } //完成当前像素点操作，退出循环 break; } else { //扩大模板 wsize++; //如果超出最大模板范围，直接赋中值 if (wsize > size / 2) { dst.at<uchar>(i, j) = Zmed; break; } } } } } return dst; } ```
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)	在几何均值滤波器实现的过程中，灰度值为 0 的点不能直接参与相乘，要把灰度值最小值设置成 1 才能参与运算。 对于边缘像素，不对其进行特殊处理，只处理邻域内存在的像素即可。

实验结果

（实验完成后的源码和打包文件的说明）

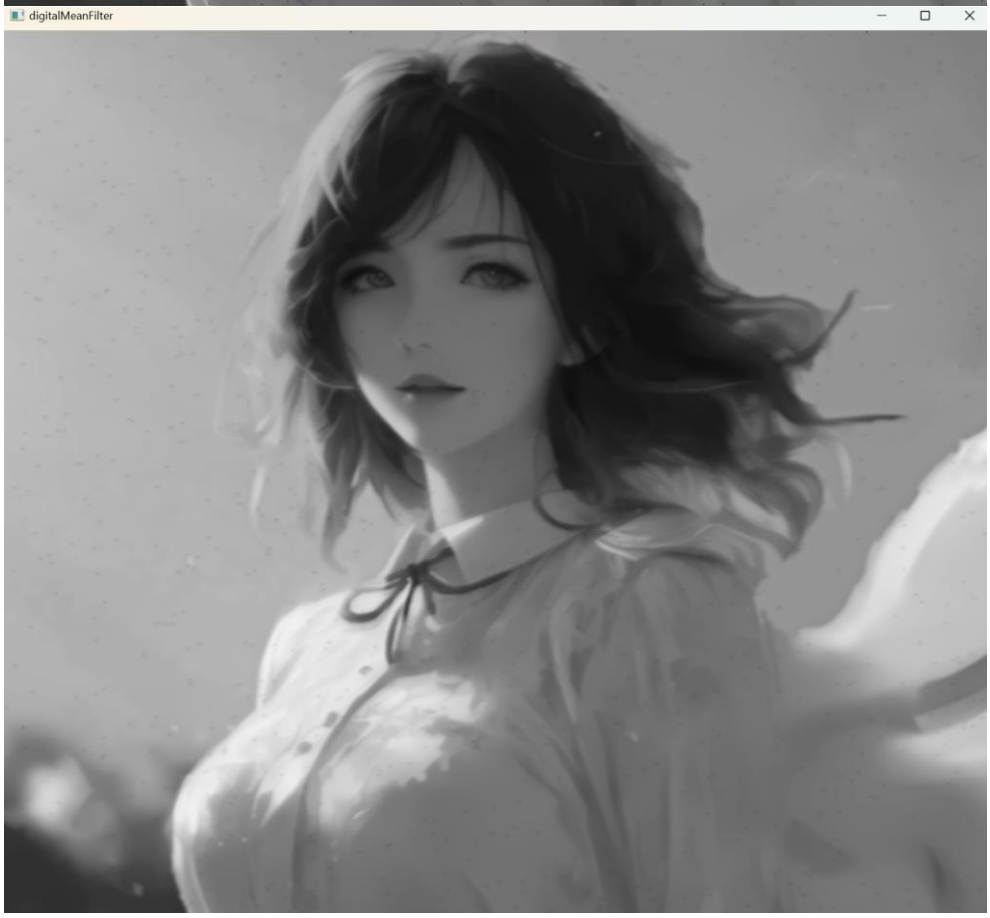

digitalMeanFilter


geometryMeanFilter

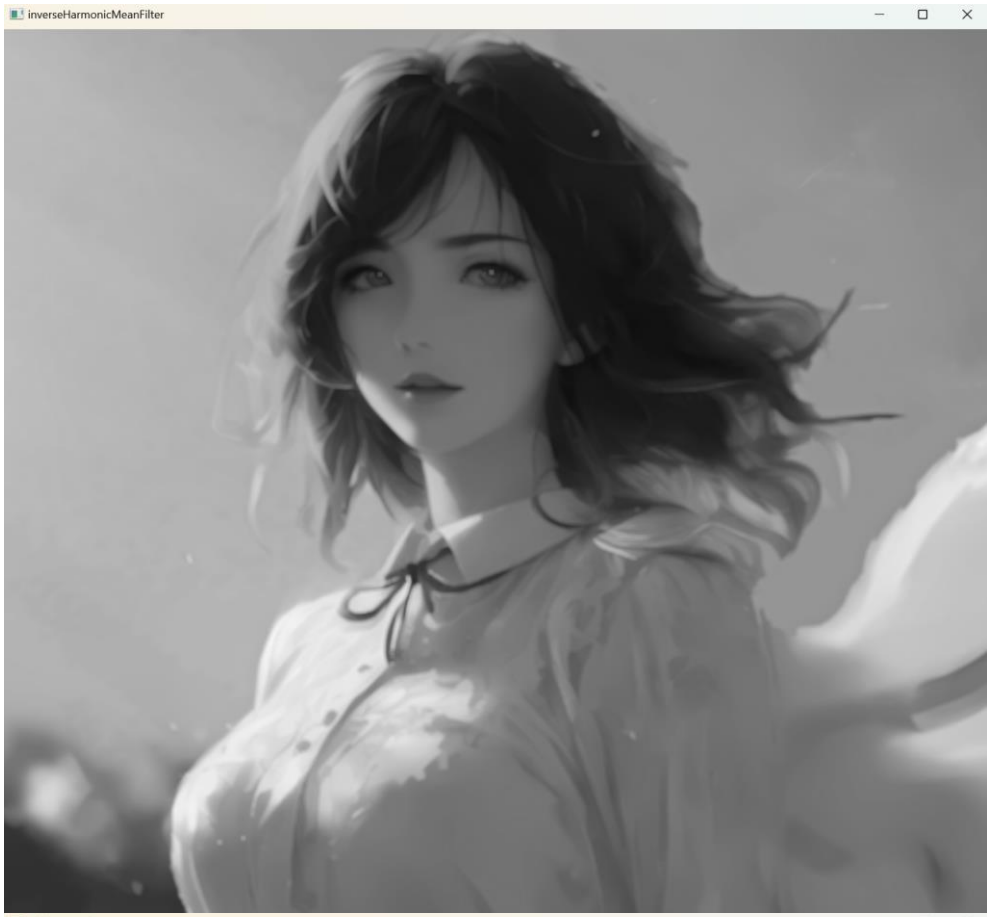

harmonicMeanFilter

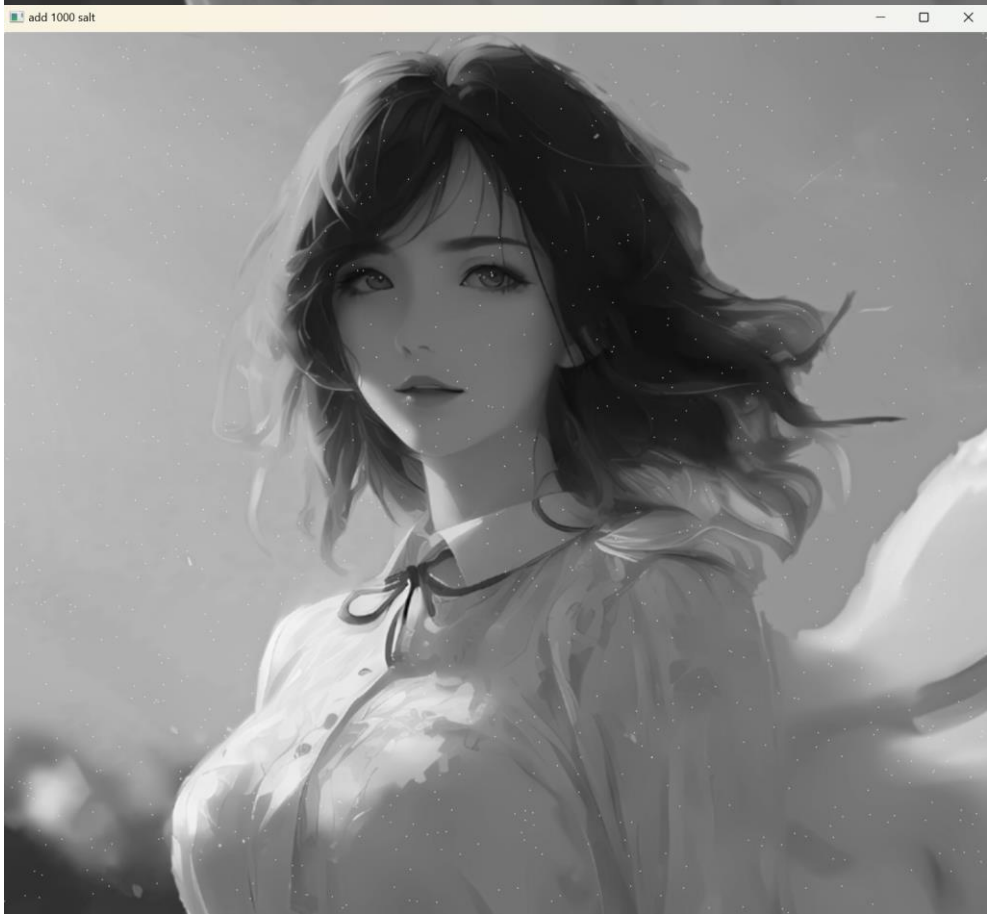

inverseHarmonicMeanFilter

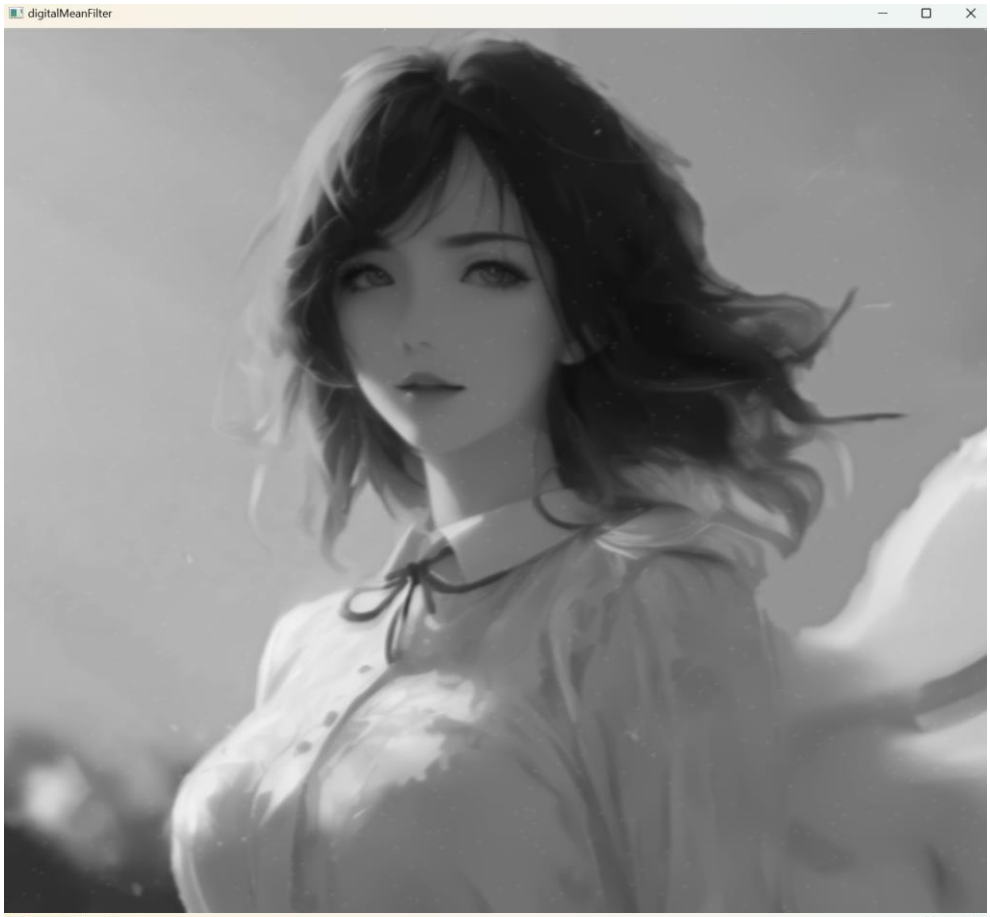

add 1000 pepper

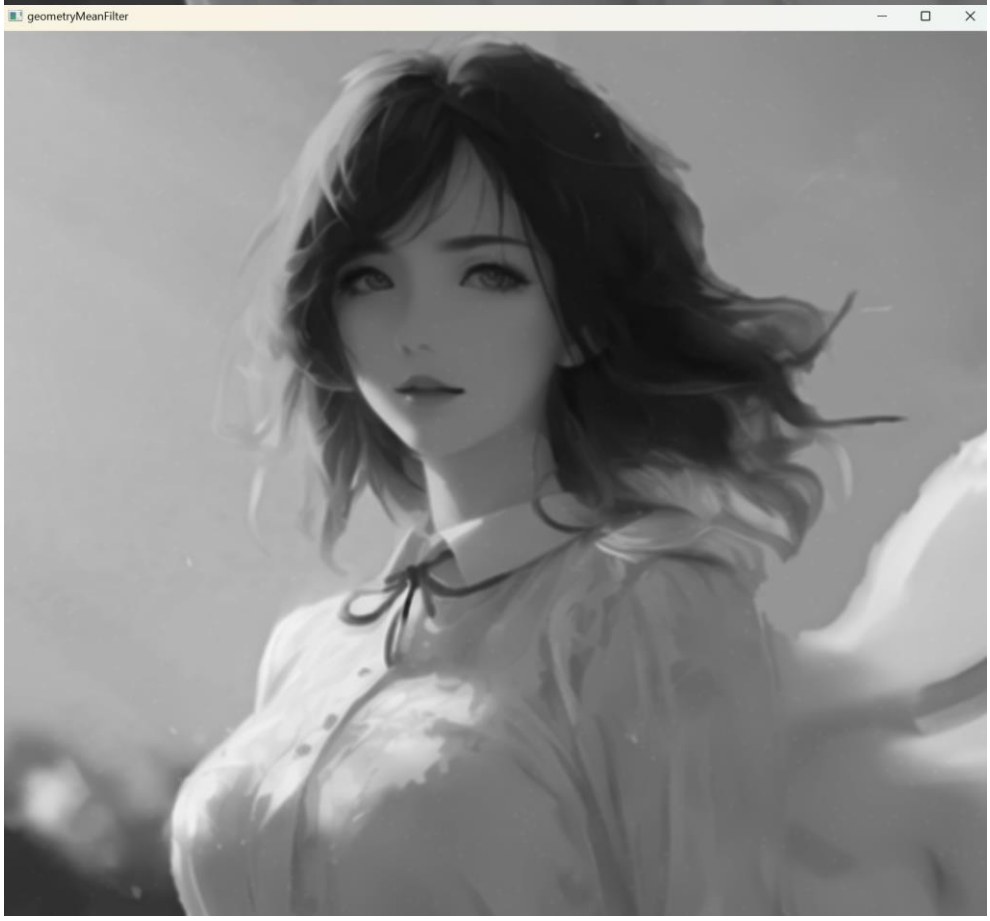

digitalMeanFilter


geometryMeanFilter
harmonicMeanFilter


inverseHarmonicMeanFilter

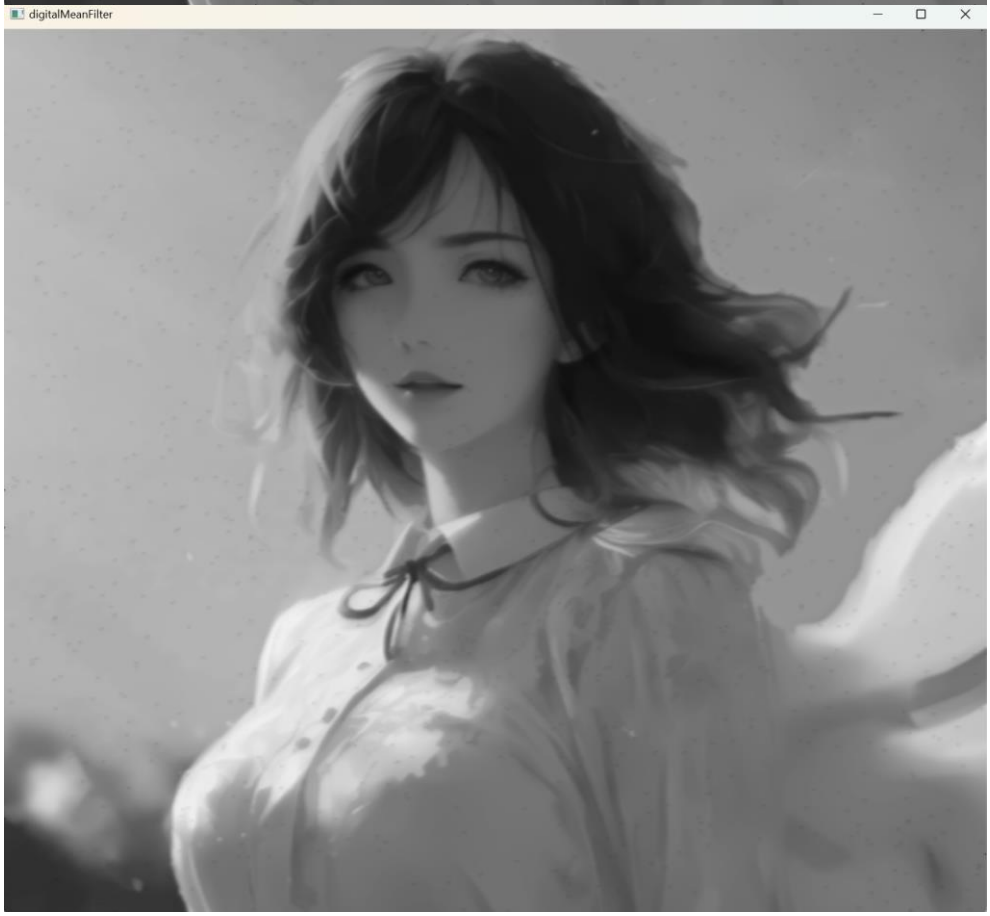

add 1000 salt


digitalMeanFilter


geometryMeanFilter

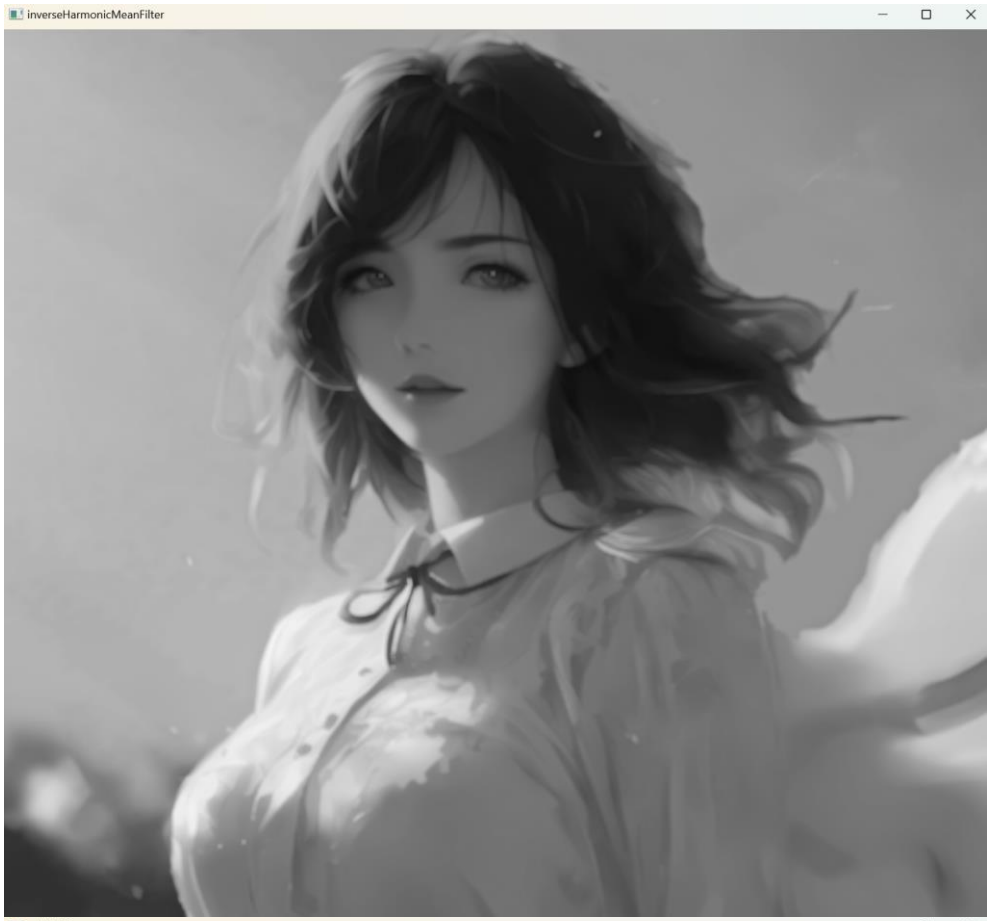

harmonicMeanFilter
inverseHarmonicMeanFilter

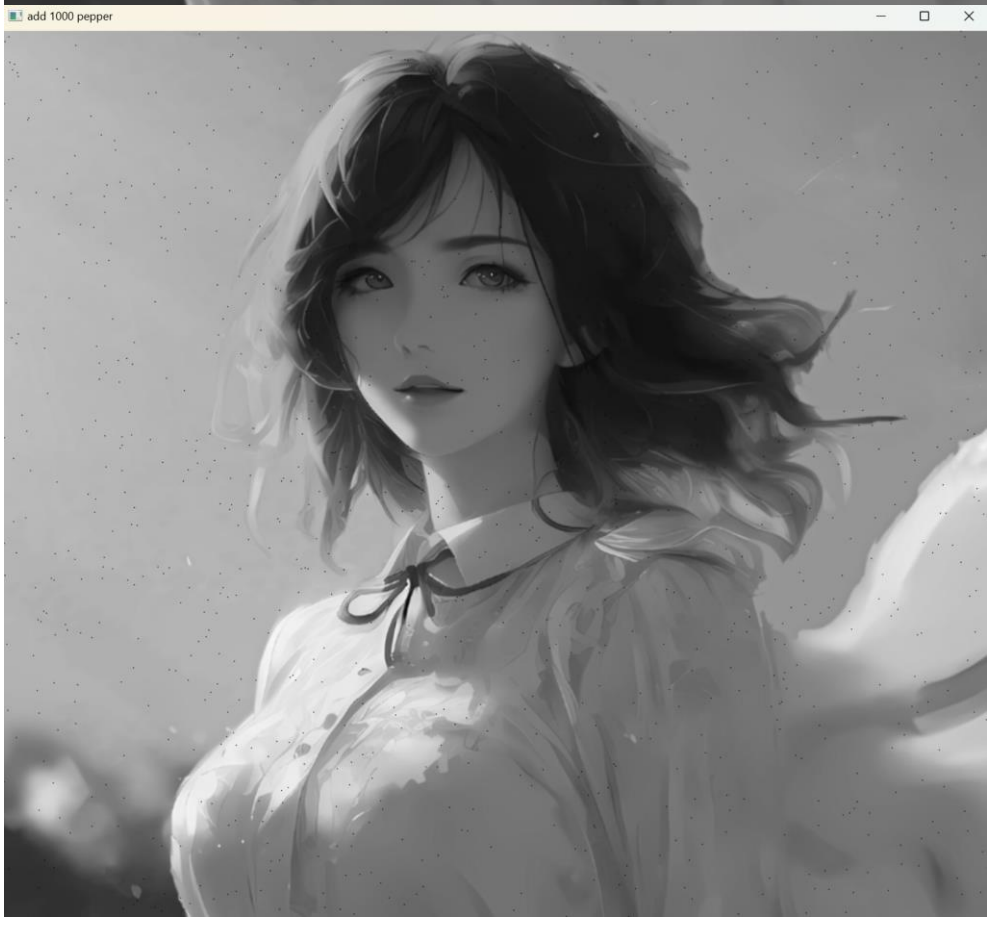

add 1000 pepper and 1000 salt

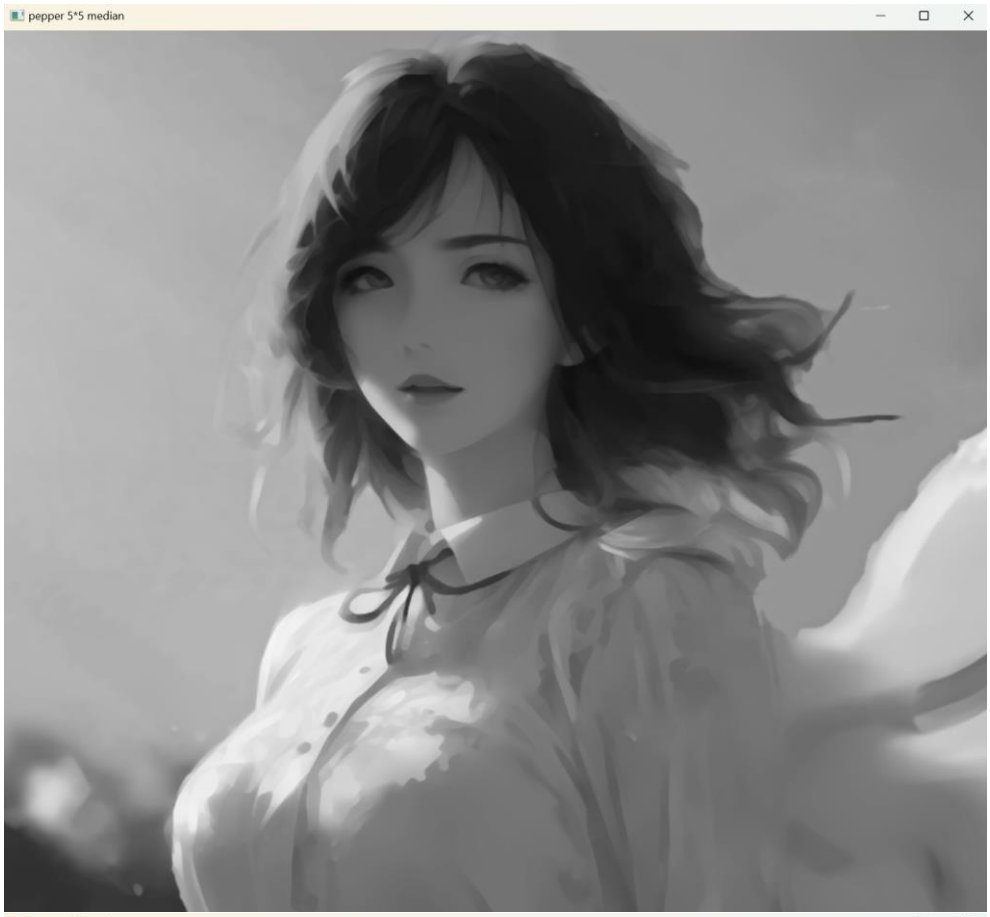

digitalMeanFilter

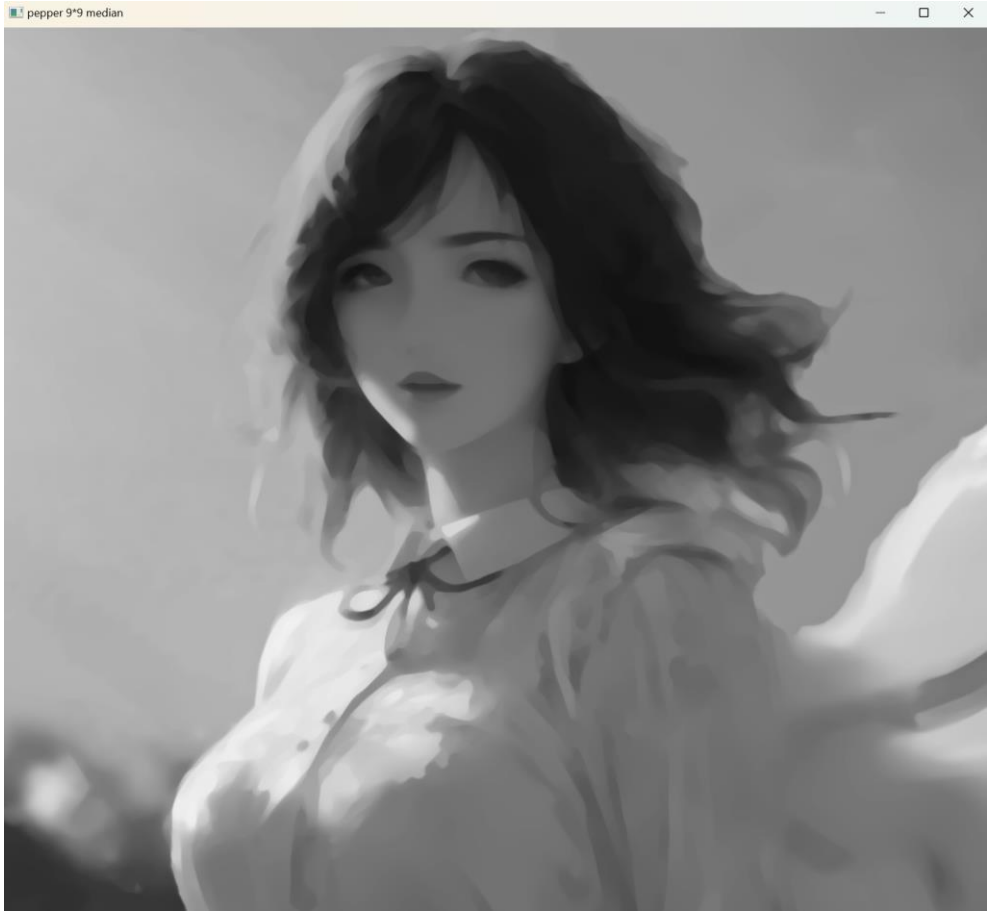

geometryMeanFilter

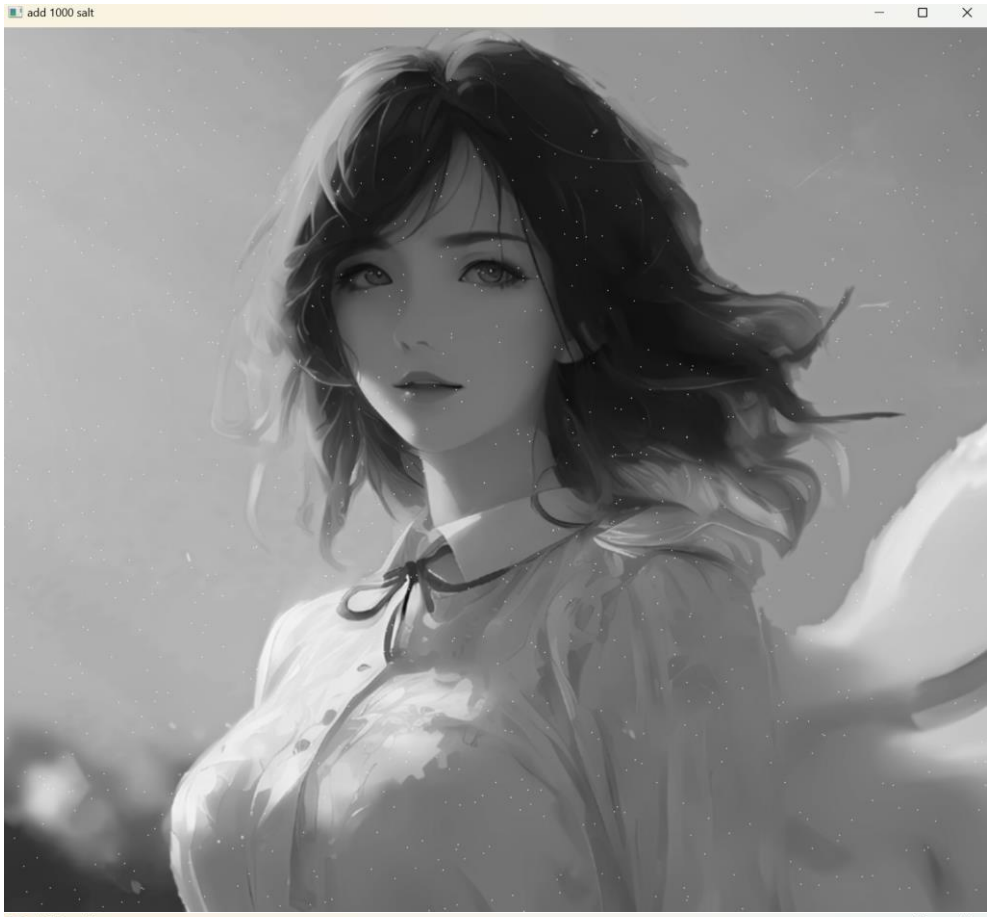

harmonicMeanFilter

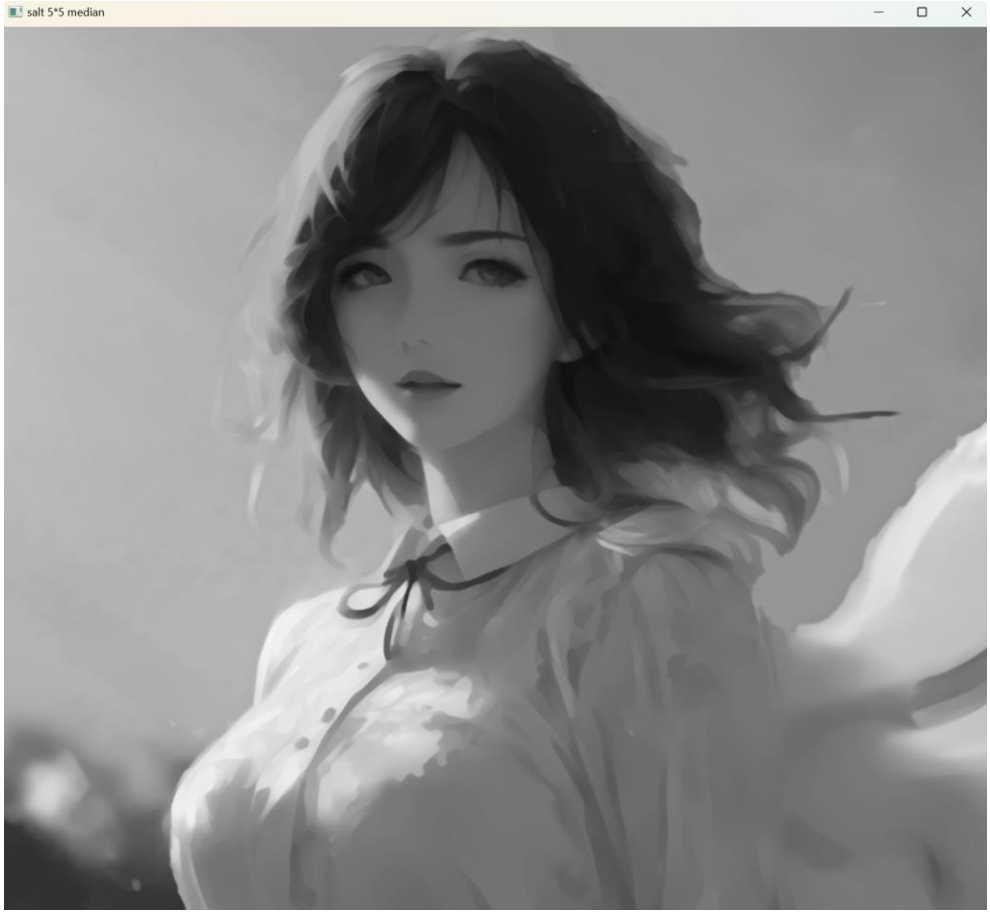

inverseHarmonicMeanFilter

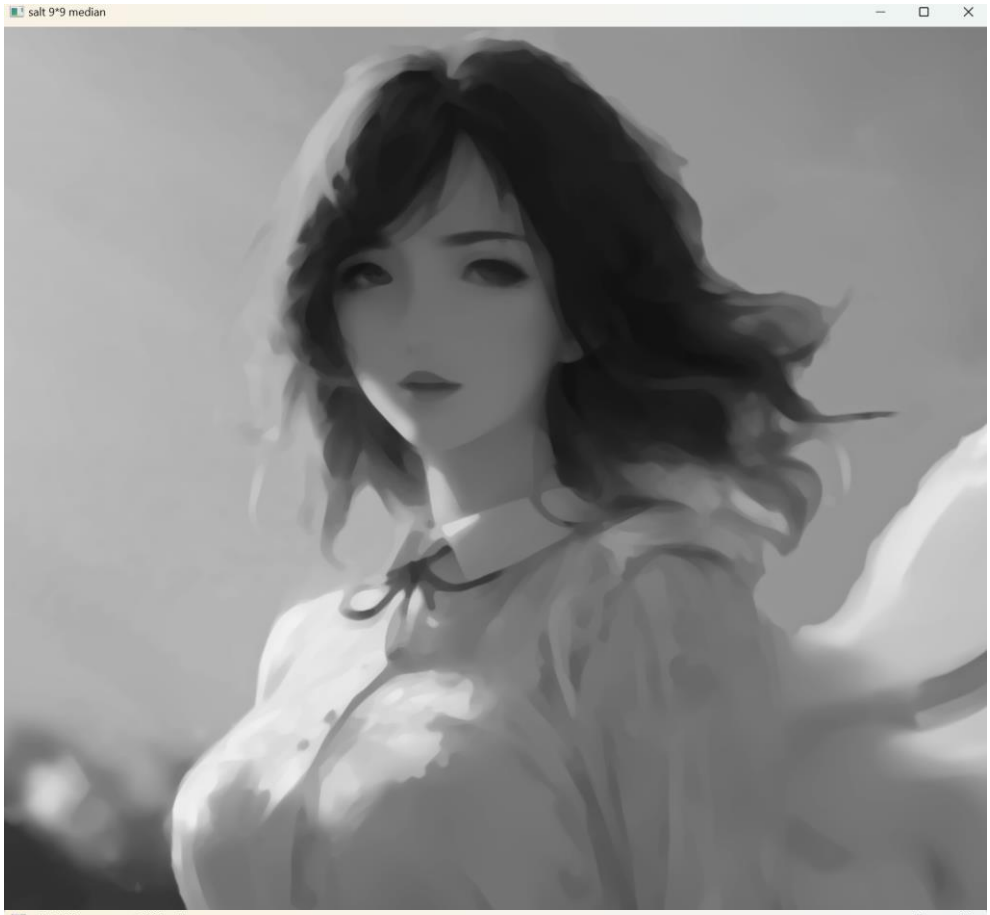

add 1000 pepper

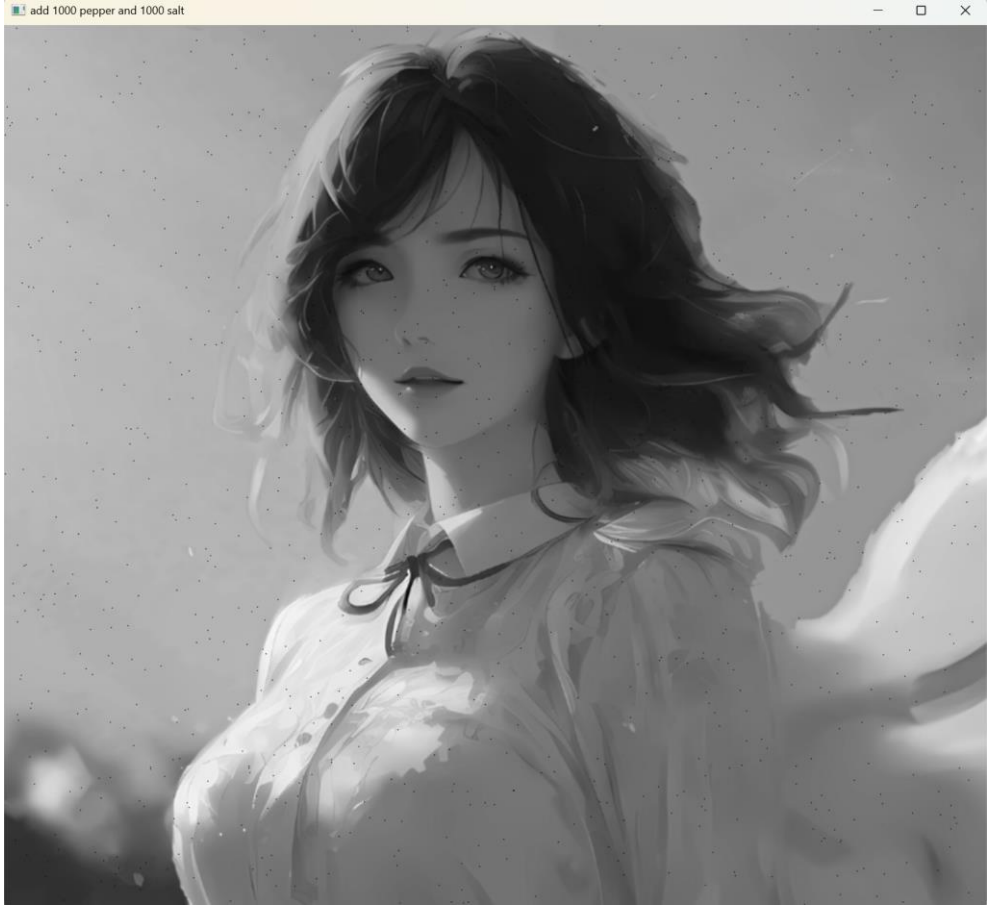

pepper 5*5 median


pepper 9*9 median

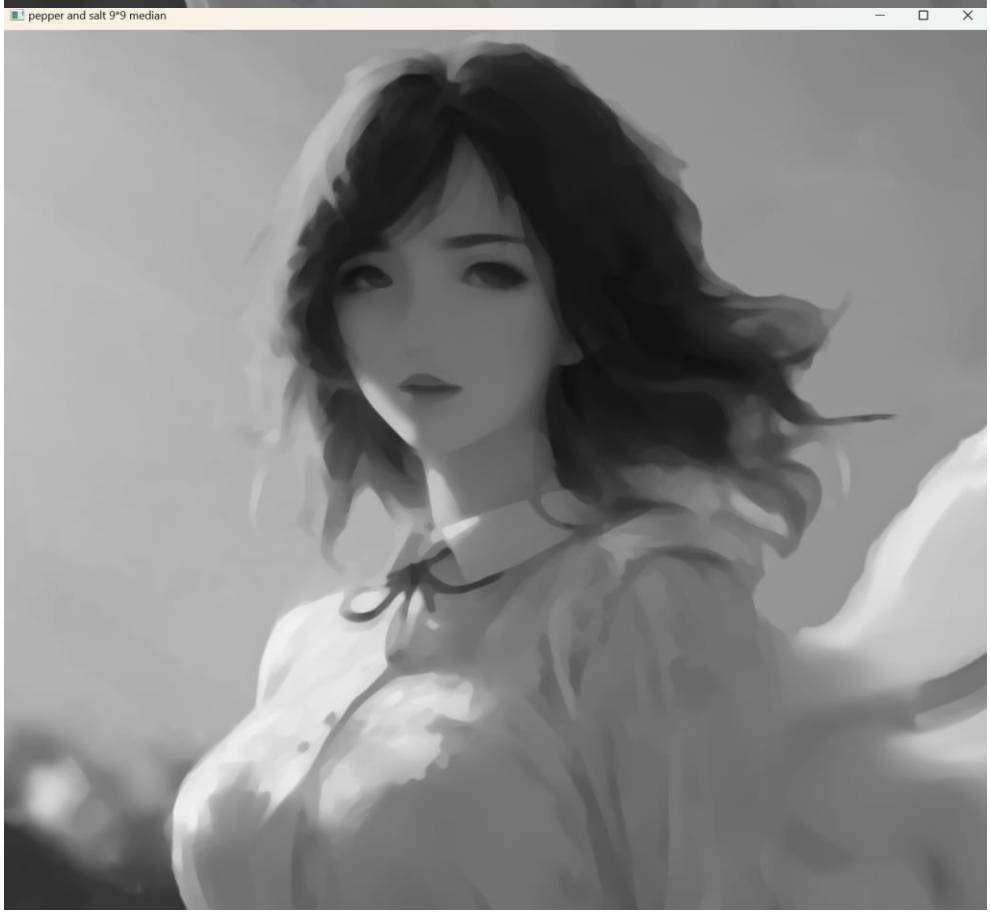

add 1000 salt

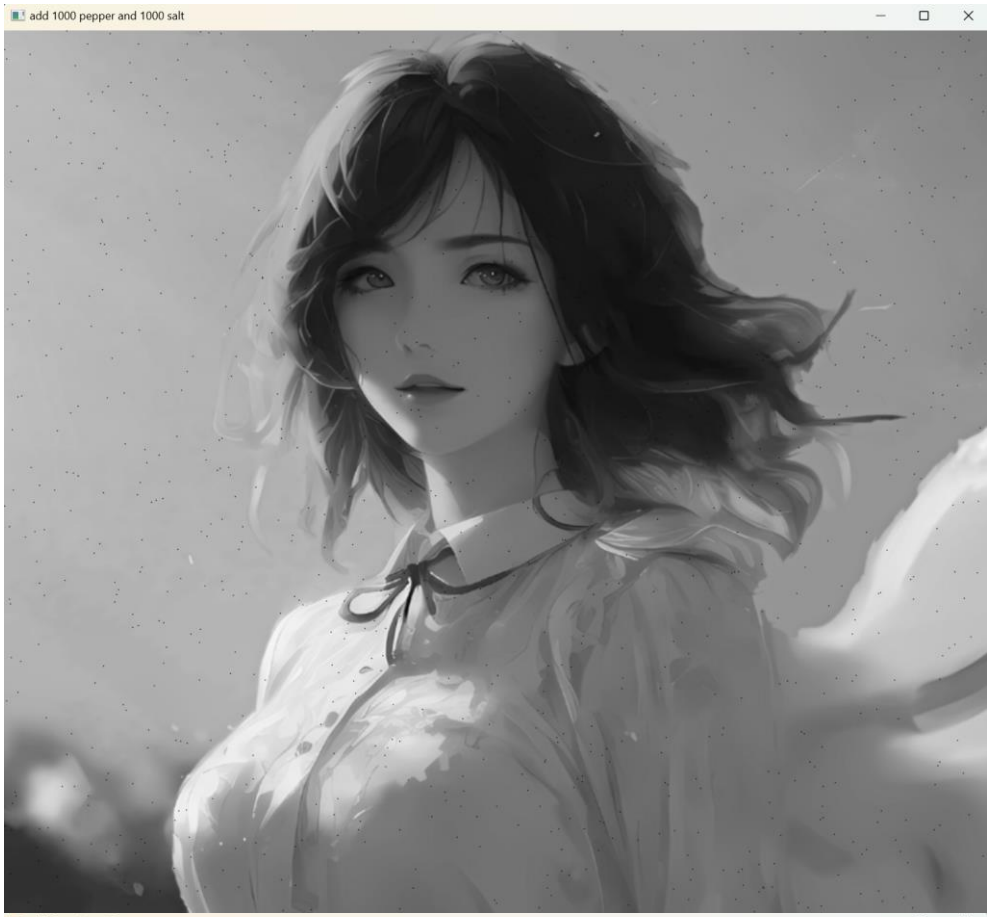

salt 5*5 median


salt 9*9 median

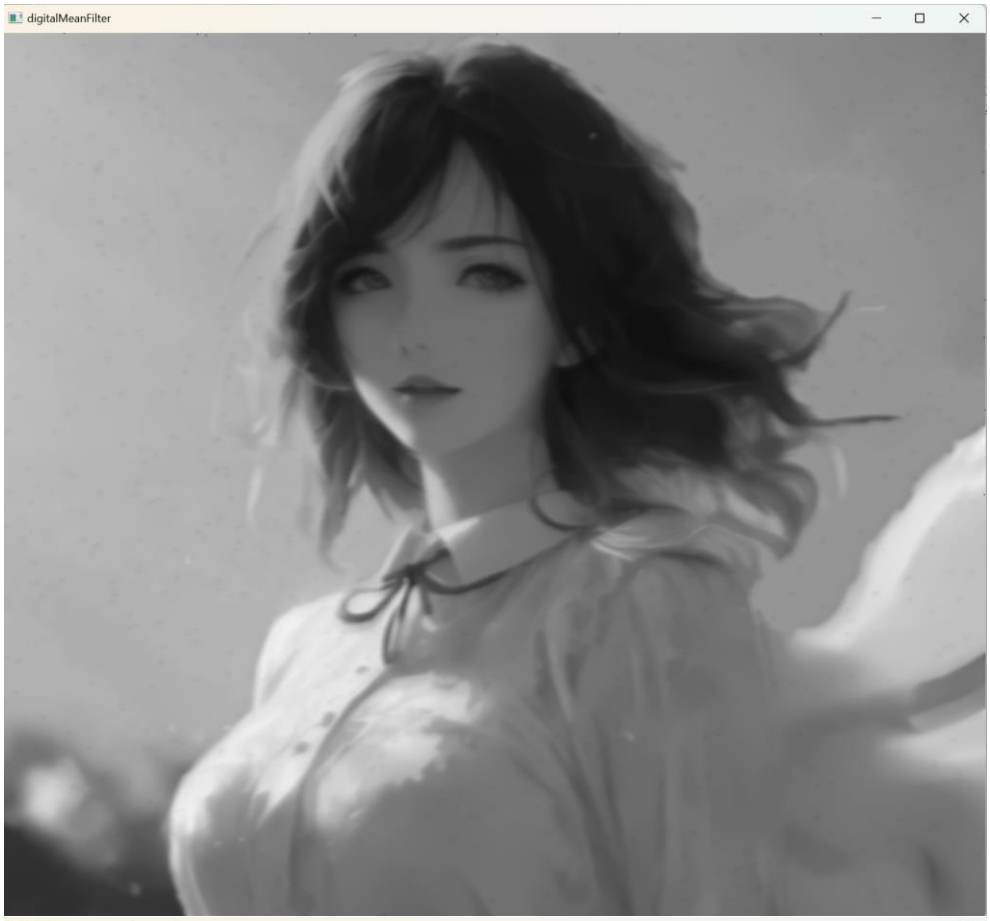

add 1000 pepper and 1000 salt

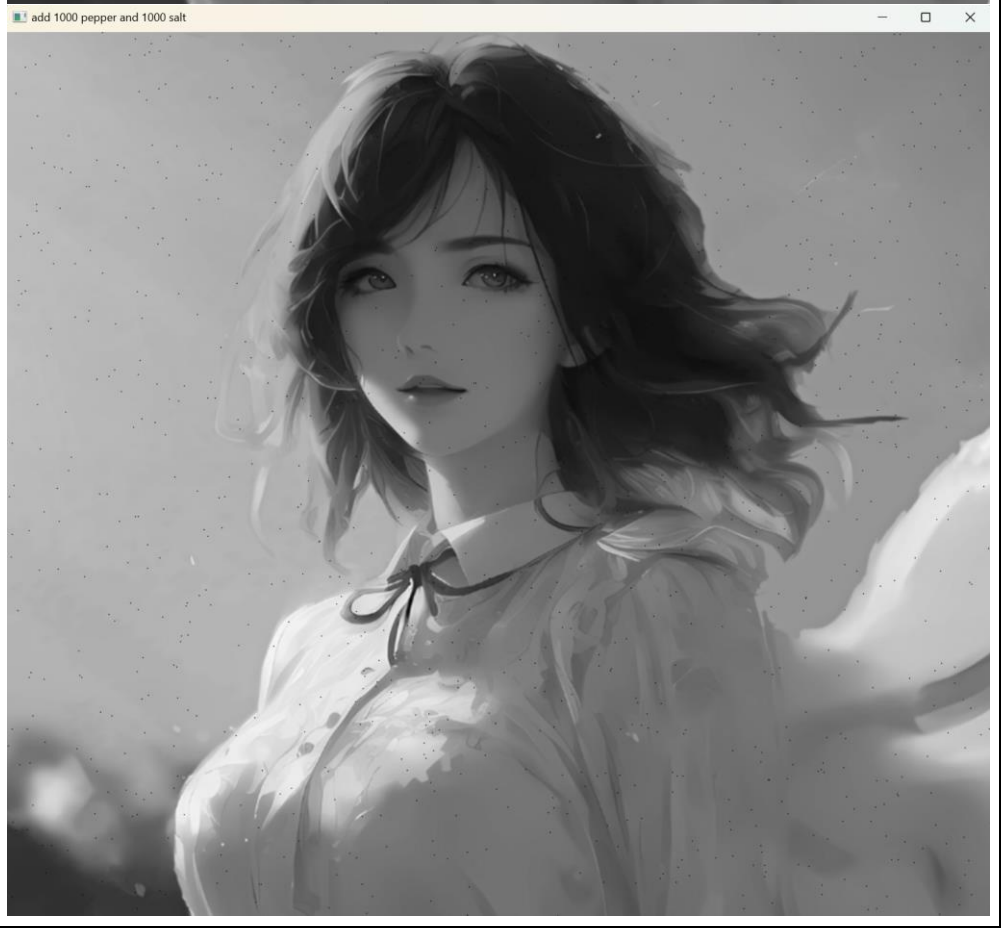

pepper and salt 5*5 median


pepper and salt 9*9 median

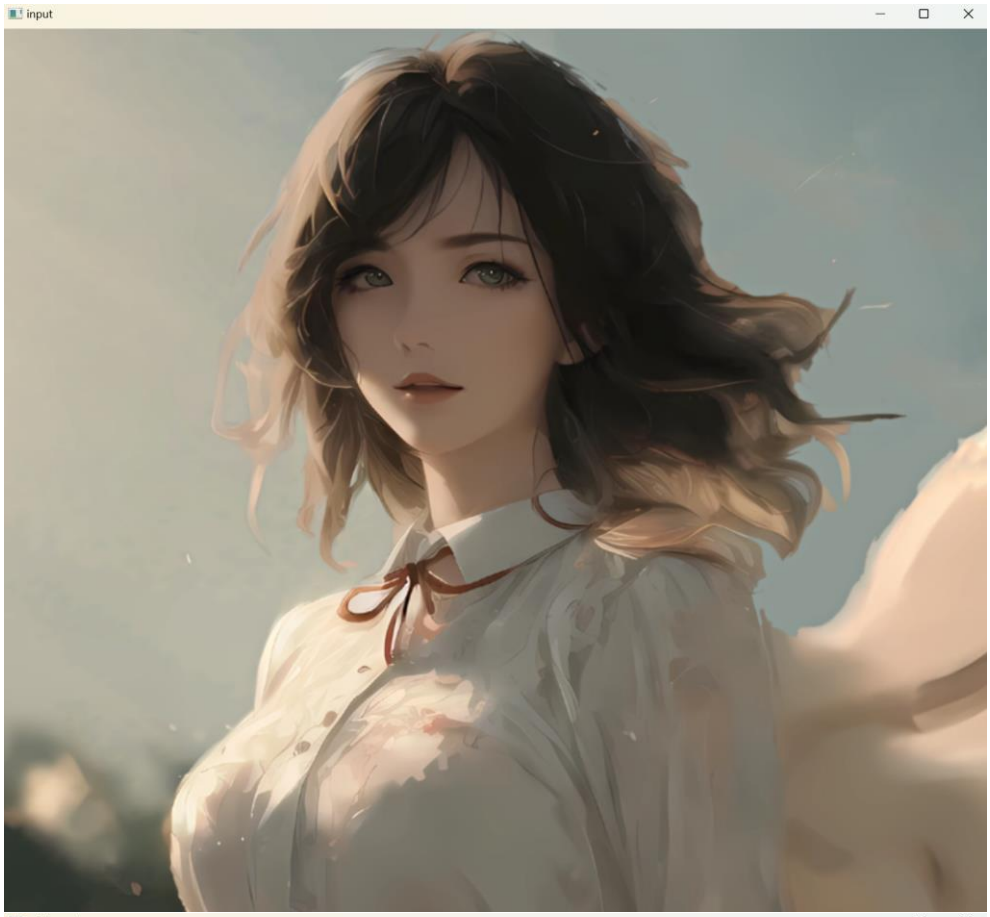

add 1000 pepper and 1000 salt

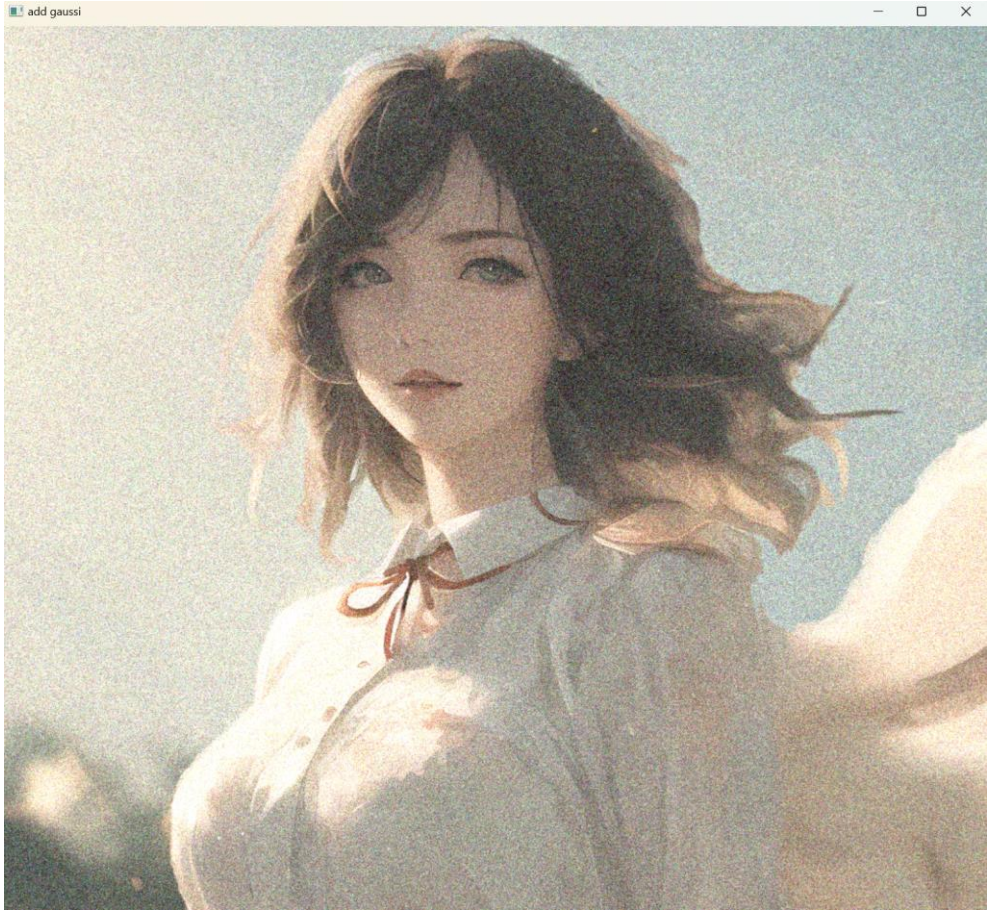

selfAdaptMeanFilter


digitalMeanFilter


add 1000 pepper and 1000 salt


selfAdaptMedianFilter
medianFilter


input


add gaussi


digitalMeanFilter
geometryMeanFilter