计算机科学与技术 学院

计算机视觉 课程实验报告

学号：	姓名：	班级： 人工智能班

实验题目：实验 4：图像滤波

实验内容：

4.1 实现图像的高斯滤波：

通过调整高斯函数的标准差(sigma)来控制平滑程度；

void Gaussian(const MyImage &input, MyImage &output, double sigma);

- 滤波窗口大小取为[6*sigma-1]，[.]表示取整；
- 利用二维高斯函数的行列可分离性进行加速；
- 先对每行进行一维高斯滤波，再对结果的每列进行同样的一维高斯滤波；

4.2 快速均值滤波

实现图像的均值滤波

- 滤波窗口大小通过参数来指定：

void MeanFilter(const MyImage &input, MyImage &output, int window_size);

- 采用积分图进行加速，实现与滤波窗口大小无关的效率；
-

实验过程中遇到和解决的问题：

（记录实验过程中遇到的问题，以及解决过程和实验结果。可以适当配以关键代码辅助说明，但不要大段贴代码。）

问题一：卷积核的大小问题：

通过创建进度条来动态的交互更改核的大小，其中高斯采用[6*sigma-1]进度条值为 sigma*0.01，为保证程序能正常健壮执行，对于计算出的核大小需要检查是否太小或为偶数：

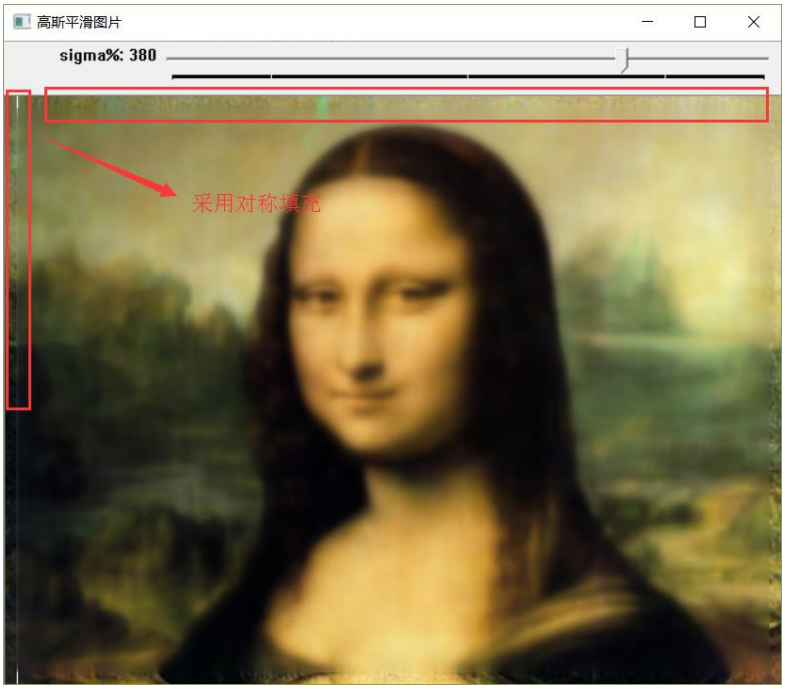
```
int ksize = sigma * 6 - 1;
if (ksize < 3) {
    cout << "ksize is too small!error" << endl;
    return;
}
int iRes = ksize % 2;
if (iRes == 0) {
    ksize += 1;
 }
```

问题二：边缘处理，实用高斯和均值时都需考虑进行填充后才能进行滤波

常见的填充方式有三种：对称复制填充、最边缘复制填充、常量填充：

可以使用 opencv 内建的 copyMakeBorder 函数实现填充

不同填充效果如下；


高斯平滑图片
sigma%: 380
采用对称填充


高斯平滑图片
sigma%: 480
采用最边缘复制填充

结合效果发现对称填充对高斯和均值滤波有更好的效果，也更符合客观事实。

问题三：行列可分离的具体实现

描述：由于高斯函数可以写成可分离的形式，因此可以采用可分离滤波器实现来加速。所谓的可分离滤波器，就是可以把多维的卷积化成多个一维卷积。具体到二维的高斯滤波，就是指先对行做一维卷积，再对列做一维卷积。这样就可以将计算复杂度从 O(M*M*N*N)降到 O(2*M*M*N)，M，N 分别是图像和滤波器的窗口大小。 这样分解开来，算法的时间复杂度为 O(ksize) ，运算量和滤波器的模板尺寸呈线性增长。

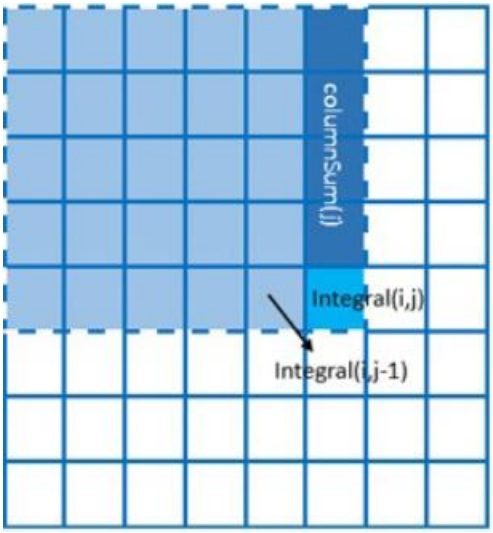
得到高斯滤波的一维数组，注意需要进行归一化：

```
int half = ksize / 2;
//初始化滤波核
for (int i = 0; i < ksize; i++)
{
    // 只需计算指数部分，高斯函数前的常数可以不用计算，会在归一化的过程中给消去
    //以（half,half）为中心建立坐标系进行计算
    double g = exp(-(i - half) * (i - half) / (2 * sigma * sigma));
    sum += g;
    GS_filter[i] = g;
}
// 归一化
for (int i = 0; i < ksize; i++)
        GS_filter[i] /= sum;
```

并且需要对单通道和三通道图像分别进行处理：

问题四：积分图的算法实现问题

1、采用一维数组 int * integral 来存储像素的积分，计算积分时注意可以使用迭代来降低复杂度，如下图：



Integral(i,j) = Integral(i,j-1) + prow(j) 其中 prow(j)为当前位置上的列的像素之和。

2、注意数组的大小由传统的 W * H 改为 (W + 1) * (H + 1)会进一步使算法简便，

使某个点的积分图反映的是原图中此位置左上角所有像素之和，这里是的累加和是不包括这个点像素本身的。可以简化算法，在计算时可以不用判断是否为原图的边界像素,如下：

```
for (int yi = 1; yi < height+1; ++yi, Integral += 3 * (width + 1)) {
            //对第一列像素值单独处理
            Integral[0] = 0;
            Integral[1] = 0;
            Integral[2] = 0;
            Vec3b rgb = src.at<Vec3b>(yi-1, 0);
            prow[0] += rgb[0];
            prow[1] += rgb[1];
            prow[2] += rgb[2];
            Integral[3] = prow[0];
            Integral[4] = prow[1];
            Integral[5] = prow[2];
    for (int xi = 2; xi < width+1; ++xi)
    {
            rgb = src.at<Vec3b>(yi-1, xi-1);
            prow[3*(xi-1)+0] += rgb[0];
            prow[3*(xi-1)+1] += rgb[1];
            prow[3*(xi-1)+2] += rgb[2];
    Integral[3 * xi + 0] = Integral[3 * (xi - 1) + 0] + prow[3 * (xi-1) + 0];
    Integral[3 * xi + 1] = Integral[3 * (xi - 1) + 1] + prow[3 * (xi-1) + 1];
    Integral[3 * xi + 2] = Integral[3 * (xi - 1) + 2] + prow[3 * (xi-1) + 2];
    }
```

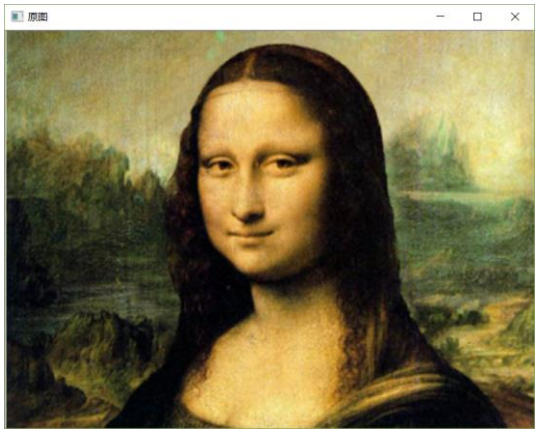
}

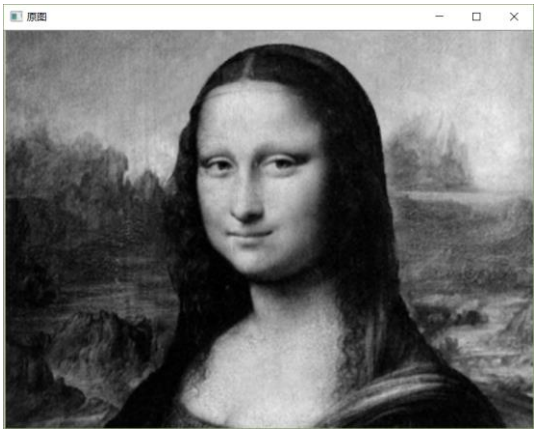
查阅资料，发现 opencv 中的内建函数也是这样定义的：

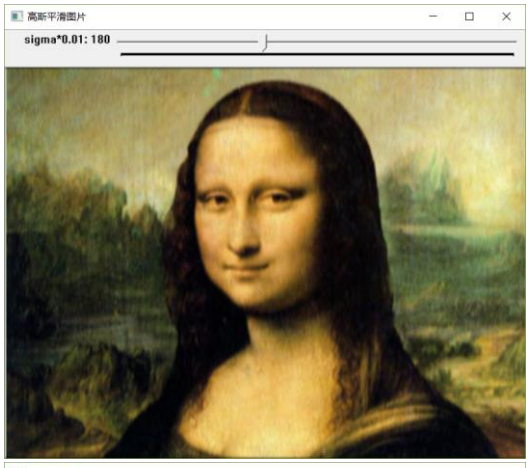
C++: void integral(InputArray **image**, OutputArray **sum**, int **sdepth**=-1

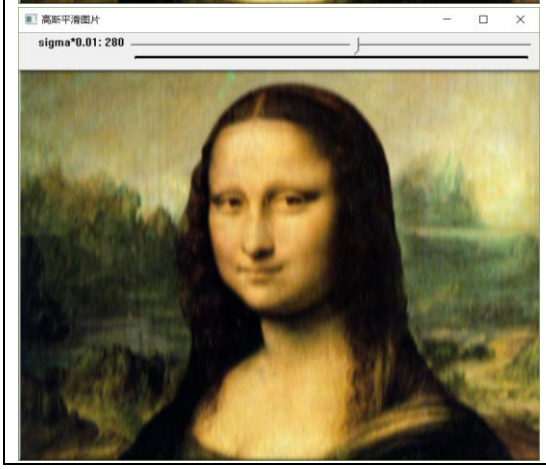
Parameters:	• **image** – Source image as **W X H** , 8-bit or floating-point (32f or 64f). • **sum** – Integral image as **(W + 1) X (H + 1)** , 32-bit integer or floating-point (32f or 64f). • **sdepth** – Desired depth of the integral and the tilted integral images, CV_32S, CV_32F, or CV_64F.

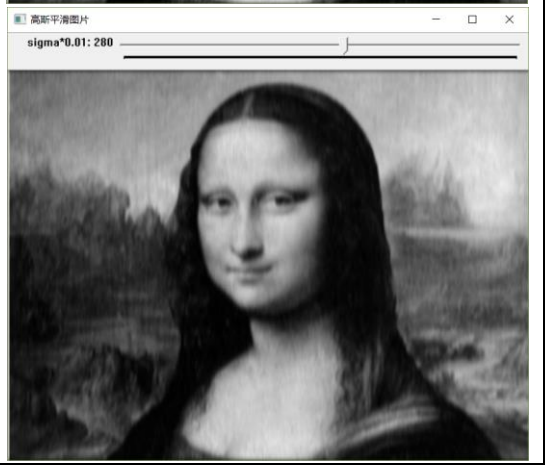
最终实验效果如下：

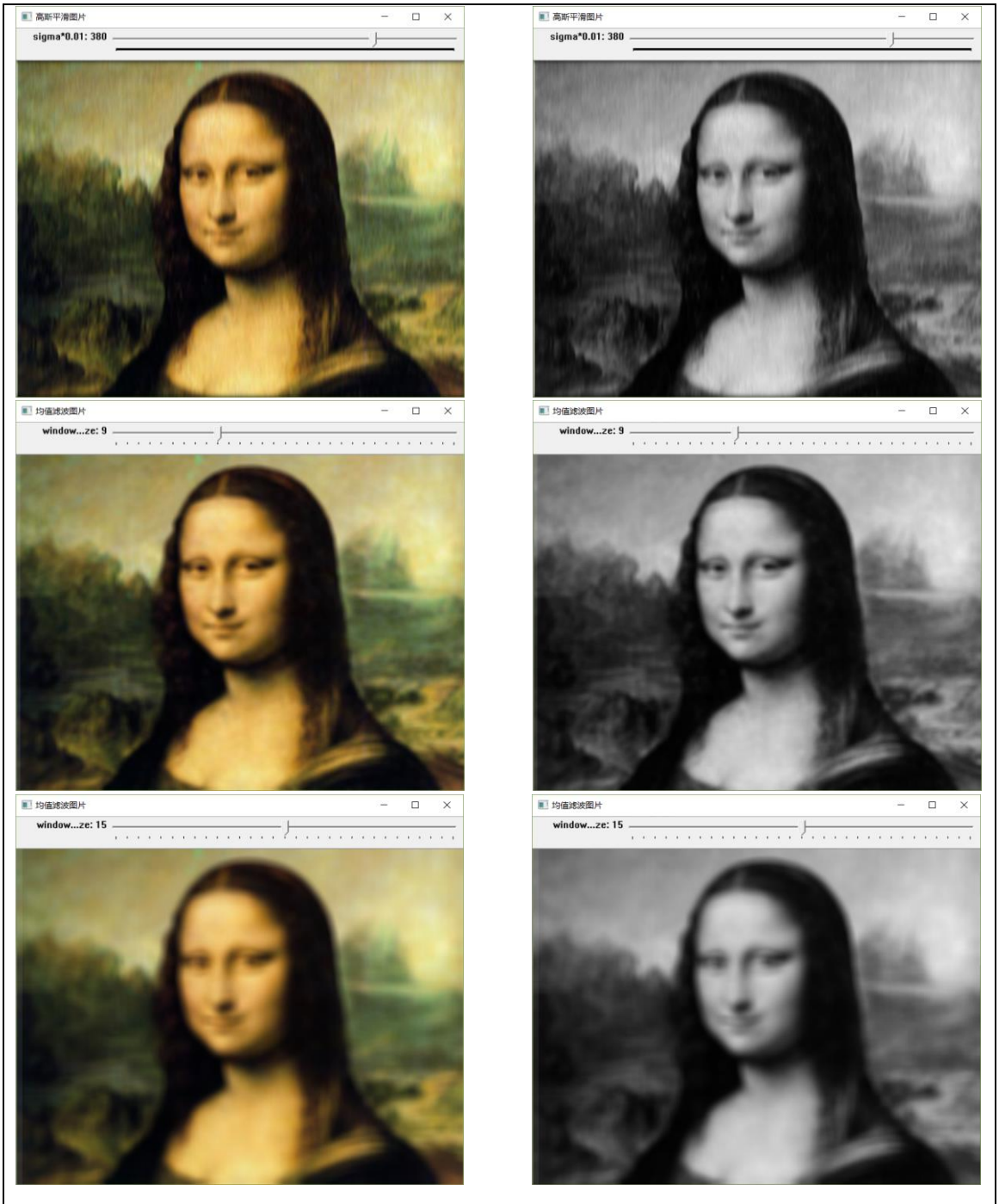











高斯平滑图片
sigma*0.01: 380
高斯平滑图片
sigma*0.01: 380
均值滤波图片
window...ze: 9
均值滤波图片
window...ze: 9
均值滤波图片
window...ze: 15
均值滤波图片
window...ze: 15