姓名

- 倪诗宇

学号

- 201900180065

实验日期

- 2021.10.10

实验题目

- 图像滤波

实验过程中遇到的问题和解决方法

高斯滤波实验：

- 问题一：
  - 问题：直接计算二维高斯滤波函数，进行归一化后，去中间的一行作为行列分离时的高斯滤波核，结果很暗
  - 解决：因为二维滤波核的归一化是针对整个二维滤波核的，归一化时的分母是整个滤波核所有数值的核，只取中间一行，加起来小于1，用这个滤波核去滤波得到的像素值小，因此得到的图像比较暗。直接使用一维滤波函数，得到一维滤波核
- 问题二：
  - 问题：本次实验，先按行滤波，再按列滤波。将按行滤波的结果存在middle_image中，得到的结果在sigma大的时候有彩色出现

    ```
    Mat middle_image;
    middle_image = Mat::zeros(padding_image.rows , source_image.cols ,
    source_image.type());
    ```

  - 解决：因为图像的存储使用的是类型，每个通道的像素值都是unchar类型

    ```
    middle_image.at<Vec3b>(r , step)[c] +=
             padding_image.at<Vec3b>(r , step + i)[c] * G_d1[i];
    ```

    每次相加的时候都会加一个double类型的数，而double在middle_image中存储是unchar类型，因此在转换时会丢失精度。而且每计算一个c，都会与卷积核长度个数相加，精度丢失很严重。

    因此直接将中间存储状态改为double类型的三维数组

    ```
    db G_d1[10000000];
    ```

    相加时改为

```
for(int i = 0 ; i < size ; i++){//一次加权经过size次相加
    middle_image[r][step][c] +=
        padding_image.at<Vec3b>(r , step + i)[c] * G_d1[i];
}
```

这样的话在第一次进行滤波时就不会丢失精度。

考虑到第一次滤波会丢失精度，那么第二次滤波也会。以前的第二次滤波相加方式为

```
for(int col = 0 ; col < source_image.cols ; col++)
    for(int step = 0 ; step < source_image.rows ; step++)
        for(int c = 0 ; c < 3 ; c++){
            transformed_image.at<Vec3b>(step , col)[c] = 0;
            for(int i = 0 ; i < size ; i++)//一次加权经过size次相加
                transformed_image.at<Vec3b>(step , col)[c] +=
middle_image[step+i][col][c] * G_d1[i];
        }
```

这样的话每次相加都会丢失精度

直接改为

```
for(int col = 0 ; col < source_image.cols ; col++)
    for(int step = 0 ; step < source_image.rows ; step++)
        for(int c = 0 ; c < 3 ; c++){
            db temp = 0;
            for(int i = 0 ; i < size ; i++)//一次加权经过size次相加
                temp += middle_image[step+i][col][c] * G_d1[i];
            transformed_image.at<Vec3b>(step , col)[c] = int(temp);
        }
```

使用一个double类型的中间变量temp，将相加进行完后再赋值给transformed_image，避免了中间过程的精度丢失。

因此，最大化地减少了精度丢失问题，解决了模糊不正确的问题

- 问题三
    - 问题：自己处理的图片总是比调用opencv内置函数处理的图片模糊得快
    - 解决：高斯函数写错了，**exp() 里面的分母中不用乘以 pi**

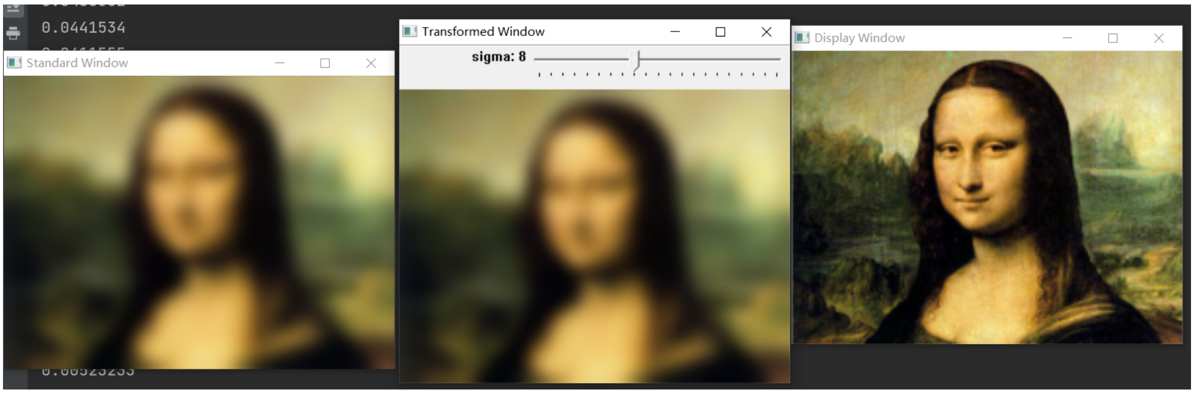
均值滤波实验

- 问题一：
    - 问题：在使用积分图求快速均值滤波的时候，左上角的坐标应该是滤波核左上角位置x， y坐标都减去1。没减1导致图像有彩色边缘

结论分析与体会

高斯滤波实验

- 左图为opencv内置高斯滤波函数
- 填充值为边缘值的映射

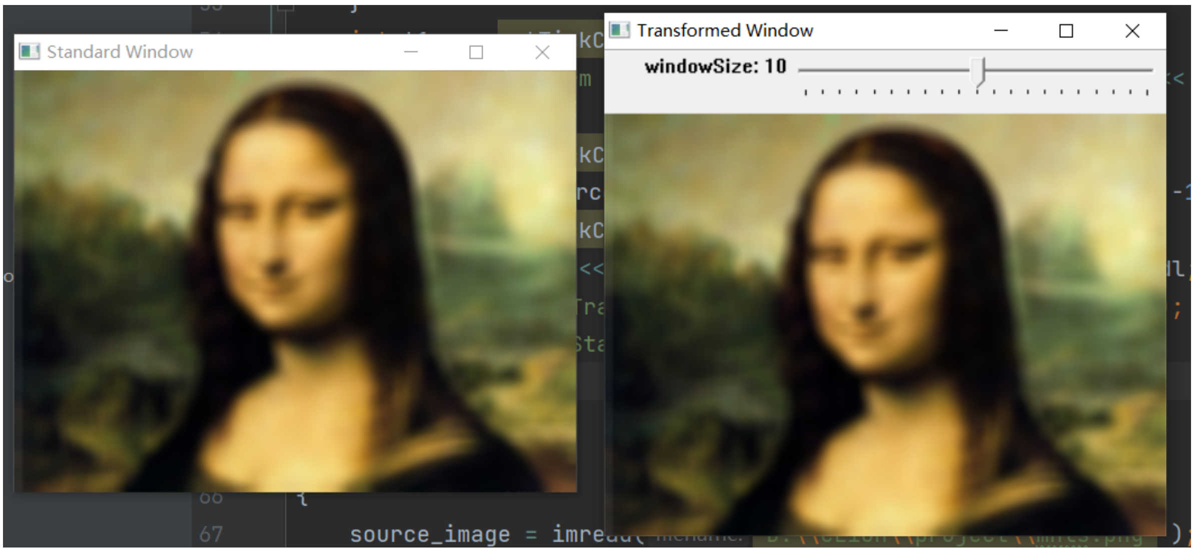


均值滤波实验

- 左图为boxFilter函数
- 填充值为边缘值的映射

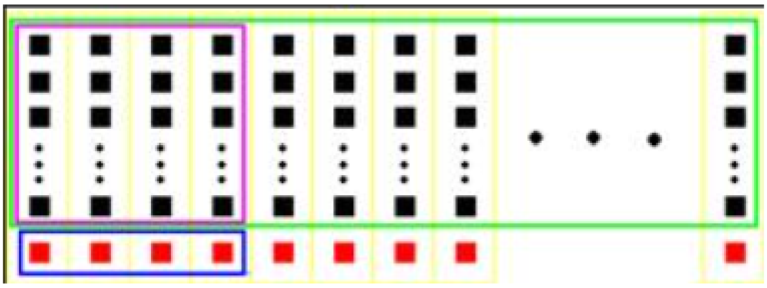
- 本次实验算法计算积分图的过程中，又开了一个前缀和数组，用来计算当前行的当前位置及之前的数值和

```
for(int i = 1 ; i <= padding_image.rows ; i++){
    init();
    for(int j = 1 ; j <= padding_image.cols ; j++){
        for(int c = 0 ; c < 3 ; c++){
            sum2d[i][j][c] = 0;
            sum_row[c] += padding_image.at<Vec3b>(i - 1 , j - 1)[c];//计算当前行的sum
            sum2d[i][j][c] += sum2d[i - 1][j][c] + sum_row[c];
        }
    }
}
```



```
transform :0.0012164 blur :0.0001199
transform :0.0015189 blur :0.0001679
transform :0.0017662 blur :0.0002608
transform :0.0052497 blur :0.0002515
transform :0.0010802 blur :0.0002536
transform :0.0012513 blur :0.0001213
transform :0.0040466 blur :0.0003599
transform :0.0012867 blur :0.0001866
transform :0.0046361 blur :0.0001526
transform :0.0011247 blur :0.0002497
transform :0.0015462 blur :0.0002409
transform :0.0011354 blur :0.0002575
transform :0.001438 blur :0.0003032
transform :0.0013012 blur :0.0002547
transform :0.0013999 blur :0.0002565
transform :0.0018278 blur :0.0002979
transform :0.001477 blur :0.0002582
transform :0.0013851 blur :0.0002604
transform :0.0050528 blur :0.0010171
transform :0.0026287 blur :0.0012113
transform :0.0012546 blur :0.0003229
transform :0.0048406 blur :0.0015412
transform :0.0034216 blur :0.0007963
transform :0.0015547 blur :0.0003473
```

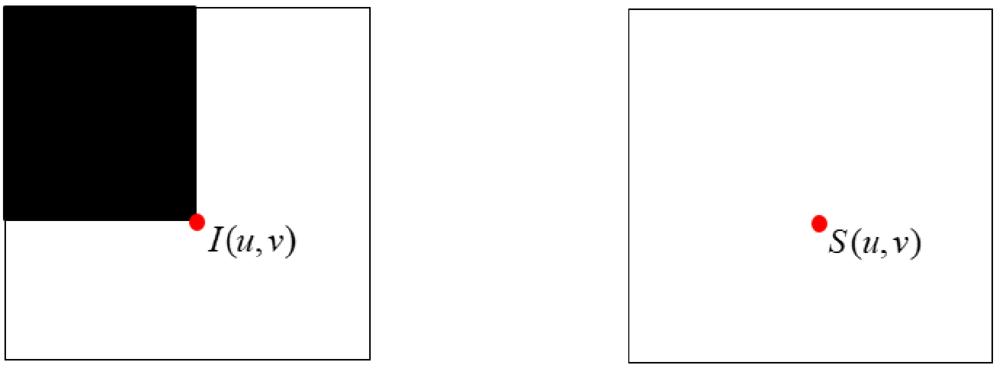
- boxFilter() 算法利用了加法的行列可分离行，复杂度为 windowSize * Width * Height



紫色的框为卷积核大小，往右平推，算每一列的和（每次只需要多算一列）。下面的红色点为算出的结果，然后在对红色点求和，得到总的和。

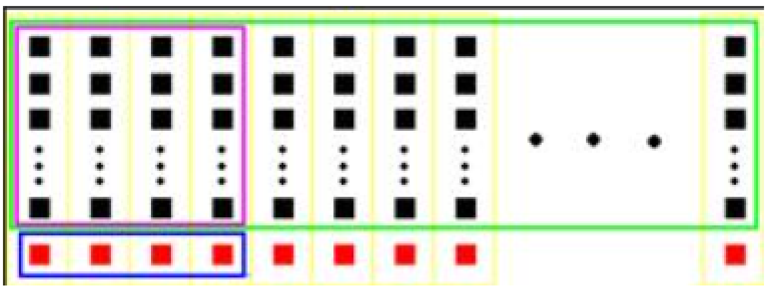
算完之后直接找就行了，不用经过额外运算

- 积分图的计算过程



$$S(u,v) = S(u,v-1) + \mathrm{sum}(I[1:u,v])$$

这里可以看到，积分图的加法除了对数据本身求和之外，还要加上前一次的值，分析一下计算紫色框内的值要进行多少次加法

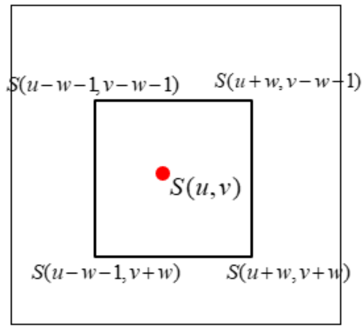


假如是一个 5 * 5 的框，需要计算 5 * 5 + 5 * 5 - 1 = 49 次。或者是 50次

boxFilter需要计算 5 * 5 + 4 = 29次。基本上为积分图的一半。

而且在计算完和之后，积分图还要利用以下公式来求解框内的数据和，boxFilter则是可以直接访问，因为boxFilter上一步的求和结果就是框内数据和。所以积分图这里又比boxFilter多进行运算。所以boxFilter更快

$$O(u,v)=\frac{1}{Z}[S(u+w,v+w)+S(u-w-1,v-w-1)-S(u+w,v-w-1)-S(u-w-1,v+w)]$$



Z=(2w+1)*(2w+1)为像素个数；
中括号内即为滤波窗口覆盖的像素颜色值之和；