计算机视觉-Ex2

作业要求

1. 改写给出的 Code1 ，用 CImg 库进行所有的图像读写、数据处理。
2. 在原来的代码基础上, 增加一个函数:首先把相邻的边缘连成长的线条，并删除长度小于 20 的 Edge。

过程

改写代码

给出的 Code1 代码中本身是有不少 bug 的，包括处理选项参数时 `-s` 选项没有 break 会导致出错，一些地方的 if 条件括号位置圈错，以及有几处地方的图像的 x，y 坐标弄反导致处理非正方形图像时只处理了部分像素的情况等等。改写后的代码对找到了的 bug 进行了修正，对于一些部分进行了较大的改写。

程序结构

提交的源代码中包含四个 C++ 程序文件，`CImg.h`文件为使用的 CImg 库代码，`Canny.h`文件为封装的类的定义，`Canny.cpp` 文件为Canny 类方法的实现，`main.cpp` 文件为用于测试的代码。

程序运行时需从参数中指定两个阈值，并在程序启动后在标准输入中输入高斯模糊使用的 sigma 和 scale。主程序运行将会保存4个文件到运行的文件夹：`smoothed_image.bmp` 文件为高斯模糊后的图片，`sobel.bmp`文件为经过索贝尔算子处理后的结果，`ecanny_map.bmp`文件为经过完整 canny 边缘检测后的结果，`linesDeleted.bmp` 文件为新增的连接边缘与删除短边缘后的结果。

编译

压缩包中已包含了 GNU/Linux(Ubuntu)，Windows以及 Mac 下编译后的可执行文件，在三个平台下编译的指令分别如下：

```
#GNU/Linux
g++ canny.cpp main.cpp -std=c++11 -pthread -o canny-linux -lX11
#MacOS
clang++ -lX11 -L/usr/X11/lib -I/usr/X11/include canny.cpp main.cpp -
std=c++11 -o canny-mac
#Windows
g++ canny.cpp main.cpp -std=c++11 -lgdi32 -o canny.exe
```

程序简述

以下将讲述程序从输入开始整个过程的大致情况：

1. 命令行参数：原版代码中写的使用方式是"Usage:edge {-roberts,-prewitt,-sobel,-frei} [-skipNMS] hresh_low thresh_high] img "，然而实际的实现指定算子是没有作用的，因此在改写的版本中的命令行使用方式改为：`Usage:canny [-skipNMS] -t thresh_low thresh_high filename`

2. 处理参数：处理参数的部分在`main.cpp`文件的`process_args()`函数中。程序代码原版代码差别不大，仅做了一点调整修正了`-s`选项不 break 和`-t`选项中 argc 未随着 argv 的读取而变化的 bug。

3. 灰度图：代码为 Canny 类的`createGreyScale`方法。改写后的程序改用 bmp 格式的图片作为输入，使用 CImg 进行图像读取和存储。在开始进行边缘检测之前，需要先将图片转为灰度图，这里的方法是将原图 rgb 通道的数值分别按 0.299 * r + 0.587 * g + 0.114 * b 的公式计算后存入一个单通道的 CImg 对象中。

4. 宏：由于改写原版的代码过程中原版代码会出现访问矩阵的负索引的情况（如高斯模糊时需要对图像边缘的像素点邻域进行访问），为了在改写后的代码中也使用类似的方法，程序定义了三个宏：`scale1XY(img,x,y)`宏代表 scale 为1倍输入的 scale 值时访问 img 的 x,y 索引，类似地`scale2XY(img,x,y)`代表 scale为2倍输入scale 时的访问，`p1XY`代表 scale 为数值1时的访问。

5. Reflect：代码为 Canny 类中的`reflect`方法，其作用为建立一个大小为(rows+2*scale, cols*+2scale)的 CImg 对象，其边缘的一圈像素为实际图片边缘像素的反射。创建后的对象使用`scale1XY`宏进行索引访问。

6. 高斯模糊：代码为 Canny 类的`gaussSmoothing`方法，接收 sigma 和 scale 作为参数。首先使用 CImg 建立一个与当前 sigma 和 scale 对应的核，然后使用该核与先前生成的灰度图进行卷积完成高斯模糊。

7. 索贝尔：代码为 Canny类的`sobel`方法。方法对高斯模糊后的图像进行索贝尔算子，使用公式如下：

   $$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \times A, G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \times A$$

   计算后图像梯度使用如下公式计算：

   $$G = \sqrt{G_x^2 + G_y^2}$$

   像素的梯度角度如下：

$angle = arctan(\frac{G_y}{G_x})$

8. 非极大值抑制：代码为 Canny 类的`nms`方法。参数接收之前索贝尔所获得的梯度和方向，对于每一个梯度点，根据其方向将其与其梯度方向的其他梯度点进行对比，若该点不是最大的，则将该点梯度置零，全部完成后，大部分的边缘宽度应可以降至一个像素。
9. 双阈值处理：代码为 Canny 类的`hysThres`方法，每次执行遍历所有像素点，将所有梯度低于小阈值的置0，将所有高于大阈值的置255，介于中间的若其8邻域中存在强点，则置为强点，否则待定。为了将所有介于中间的点都正确标记，需要不断进行递归，直到一次执行中没有点被更改则递归终止。
10. Canny：代码为 Canny 类的`runCanny`方法，调用后对高斯模糊后的图像依次进行索贝尔运算，nms 及双阈值处理，并在最后将所有非255的点置0，从而使得图像仅存在0或255的点，实现二值化。
11. 连接相邻边缘：代码为 Canny 类的`connectEdges`方法，对传入的图像进行遍历，对于每一个非边缘像素点若其8邻域中存在两个或以上的不相邻的边缘点，则认为其在边缘相邻处，将其置为255。
12. 删除长度小于20的边缘：代码为 Canny 类的`deleteShortEdges`方法，不断遍历传入的图像，从找出的第一个未访问过的边缘点开始使用广度优先算法访问其八邻域各点，将各点推入队列，每访问一个点边缘长度加一，对于所有点访问后长度未达到20的边缘不加入到生成的结果图像中，从而达到最终生成的结果图像不存在长度小于20的边缘的效果。

在以上所述之外，Canny 类还包括了以下几个简单的方法：

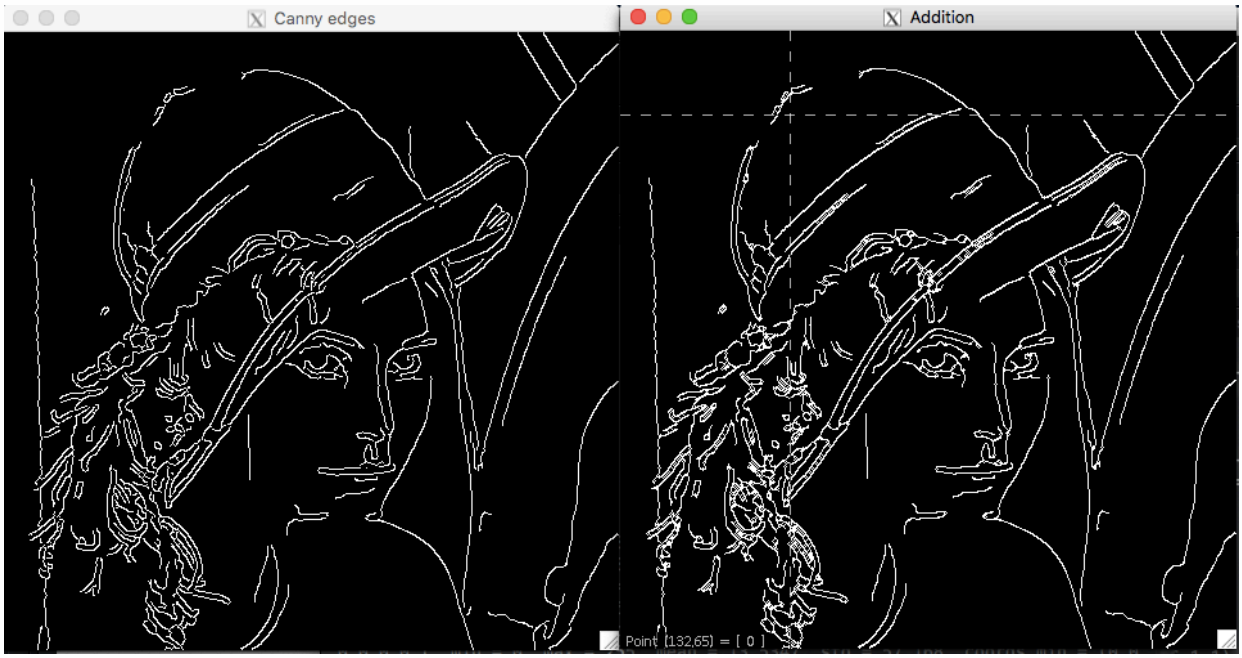
1. `getGreyScale`和`getSmoothImage`分别返回生成的灰度图和高斯模糊后图像的 CImg 对象
2. `connectAndDelete`方法将直接对传入的图像进行新添加的连接和删除操作。

测试

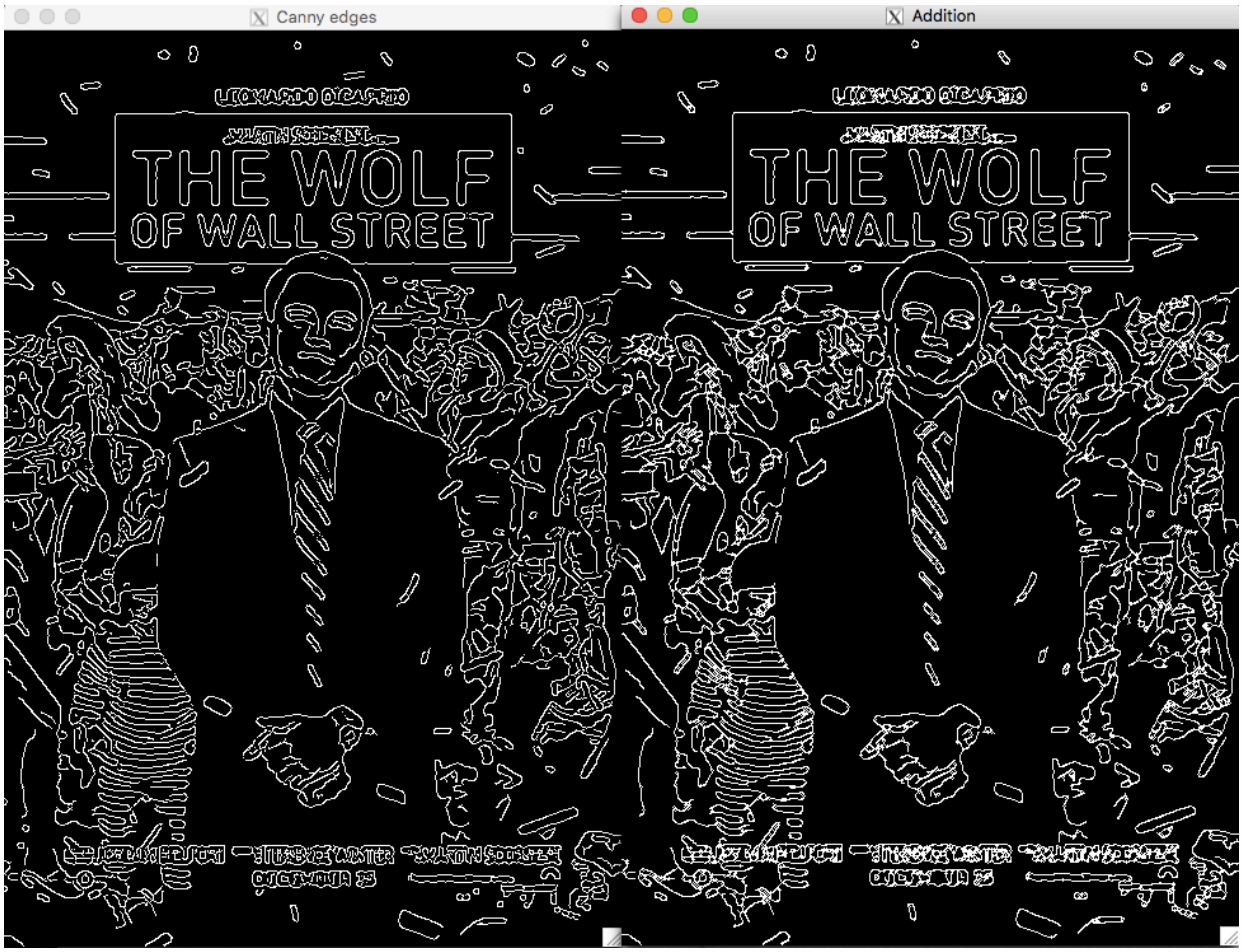
对代码的测试用了给出的四个测试用图，测试了高斯模糊 sigma，核尺寸，大小阈值对程序输出的影响。

下面是在 sigma=2，scale=2，low thresh=20, high thresh = 40的情况下图像输出效果（每组第一个为 canny 输出结果，第二个为连接和删除短边缘后的结果）：

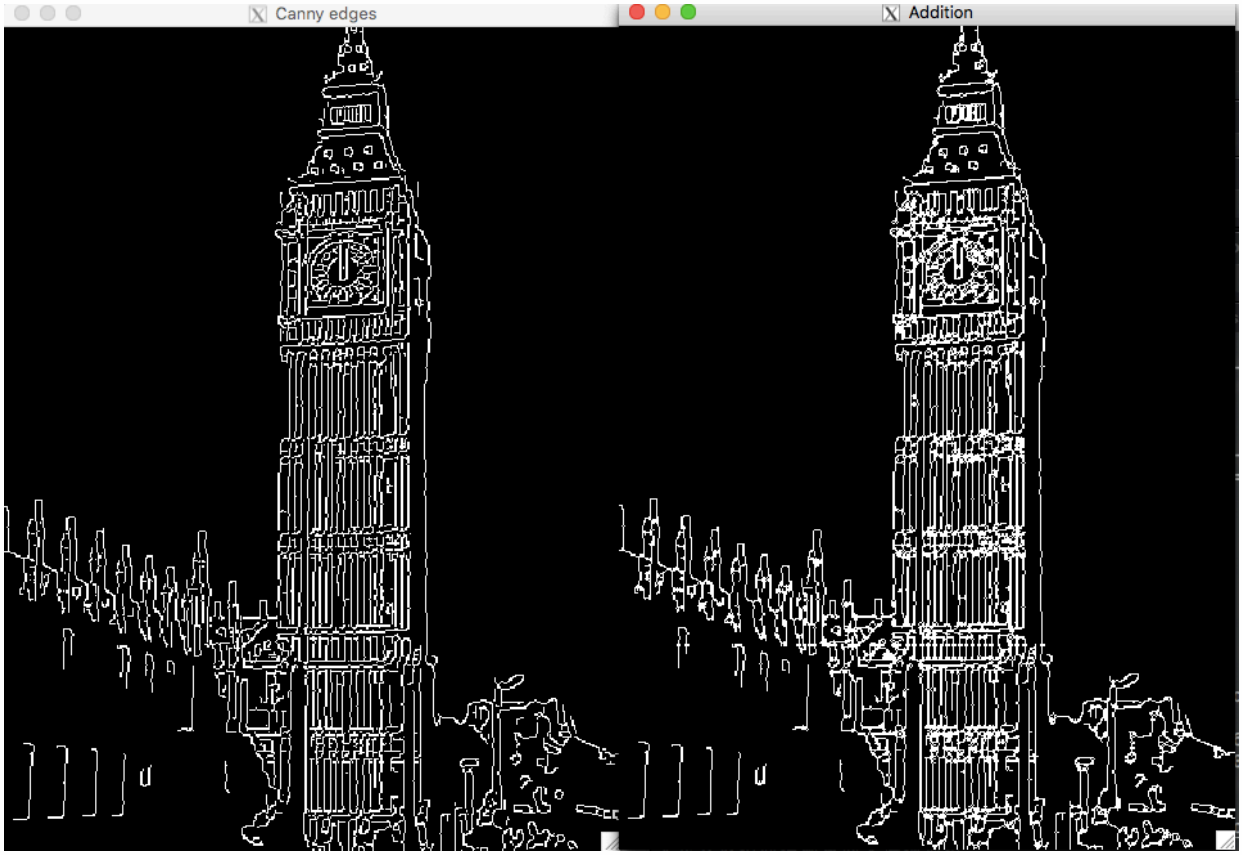
lena


Canny edges
Addition
Point (132,65) = [0]

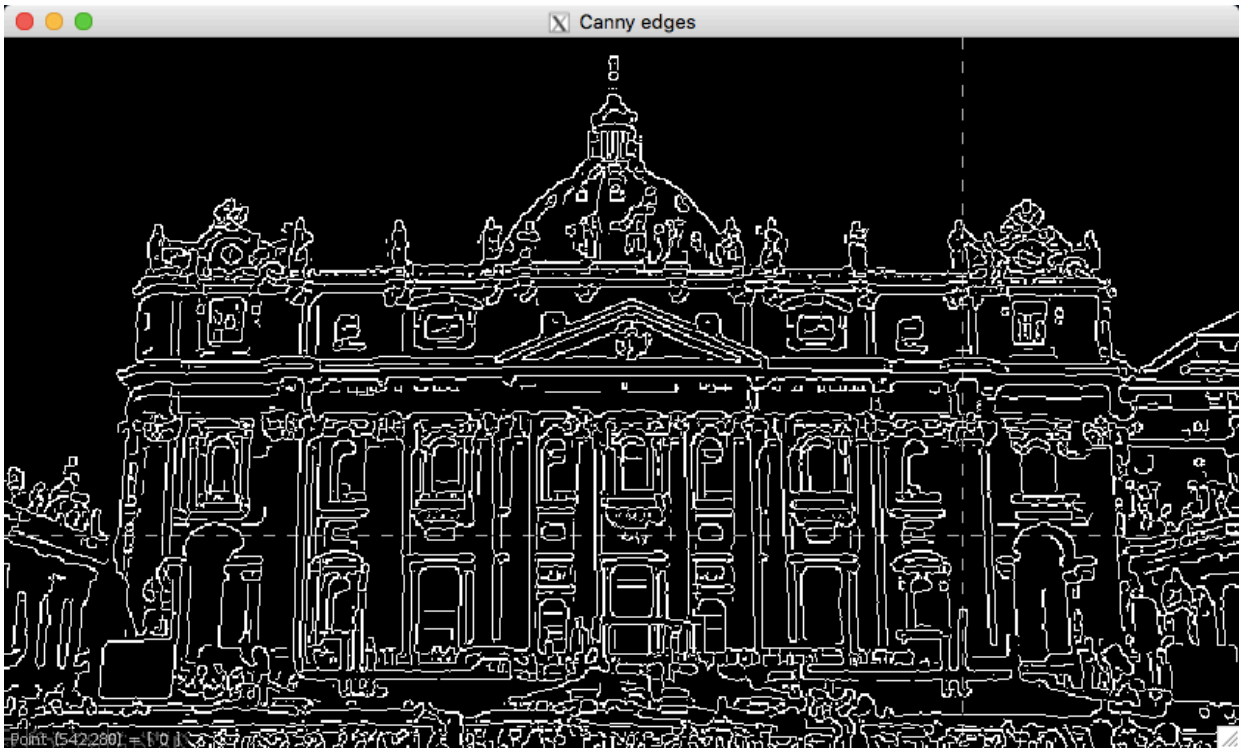
twows

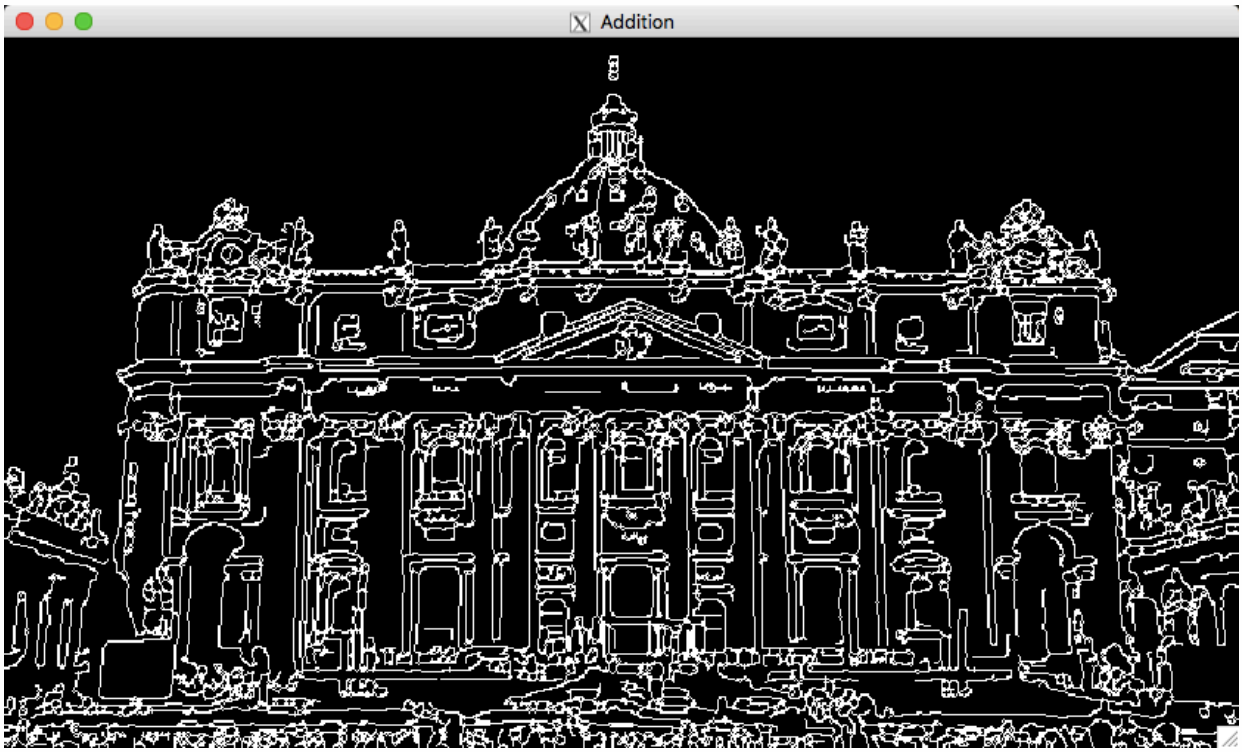

Canny edges
Addition
THE WOLF
OF WALL STREET
THE WOLF
OF WALL STREET

bigben



stpietro





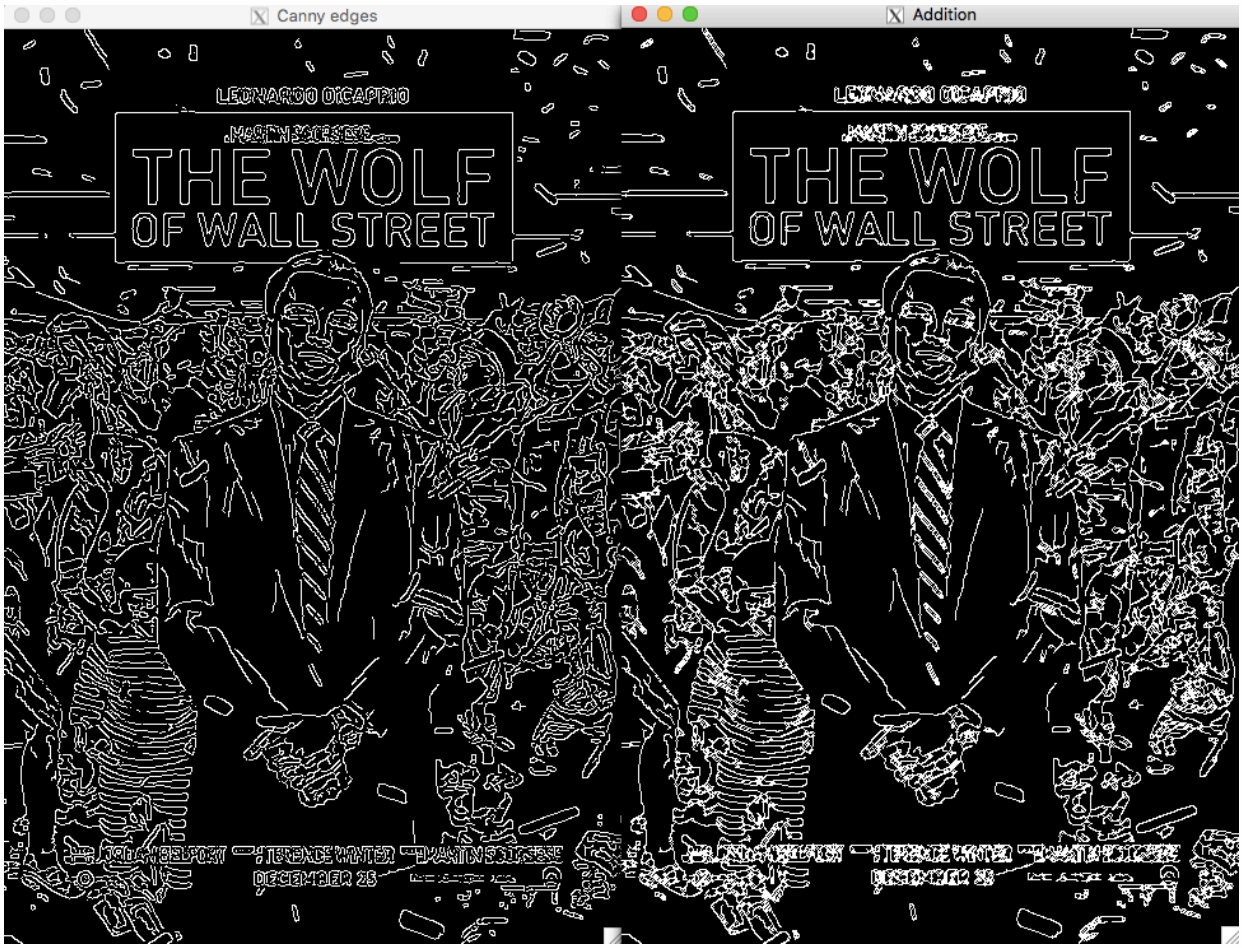
对照

以下将在其余参数不变的情况下调整其中一个参数，观察各参数的影响。

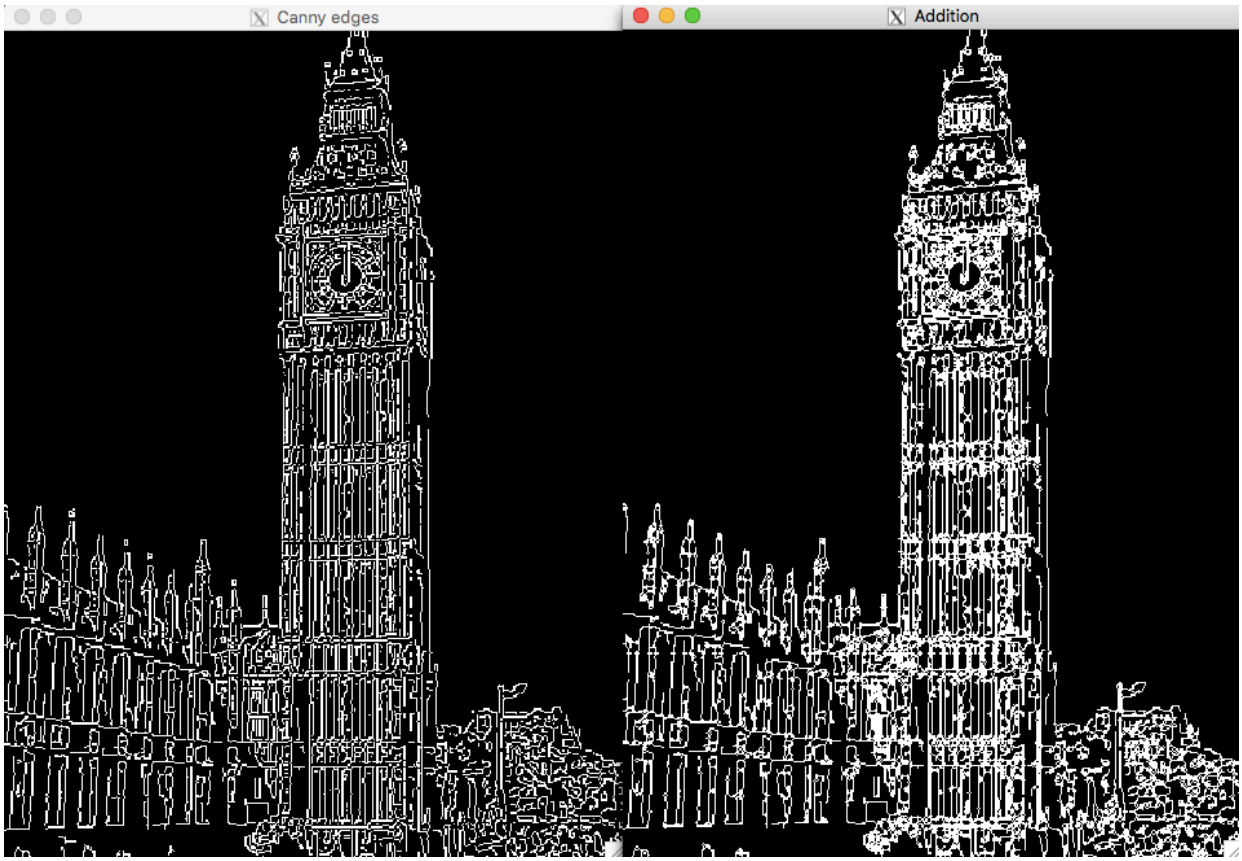
$$\sigma = 1$$

lena

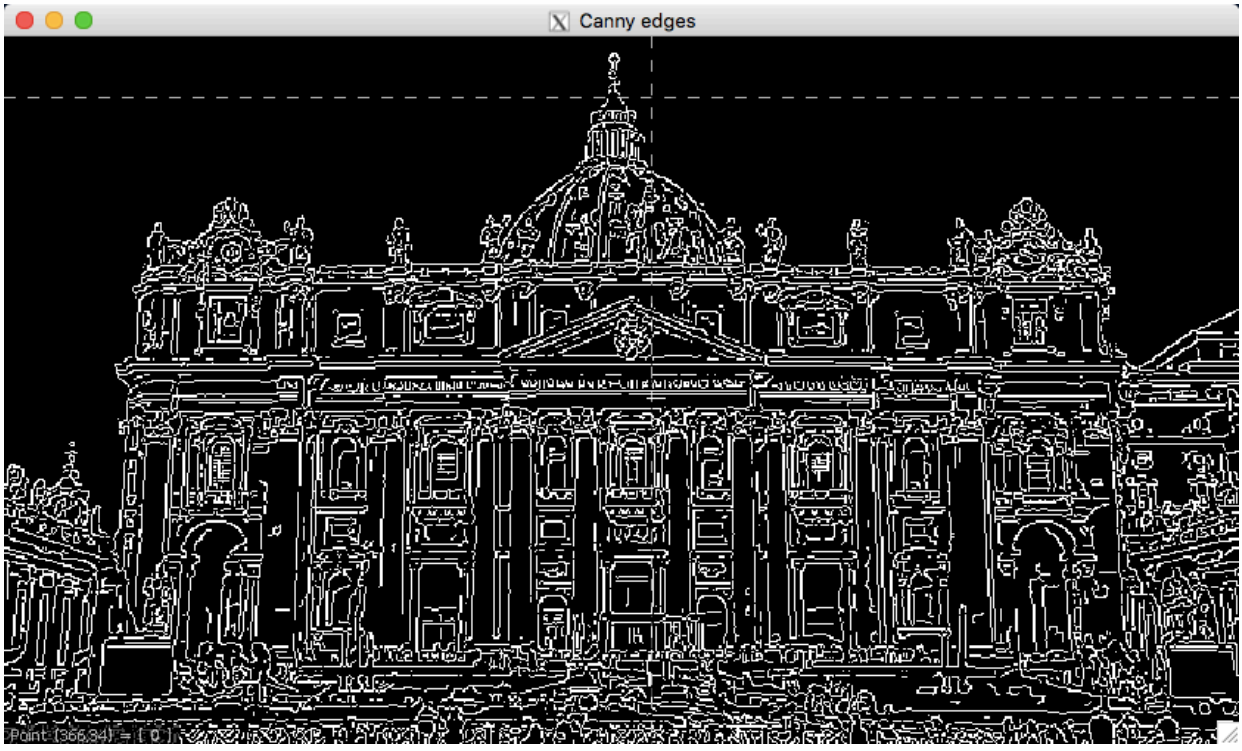
twows



bigben



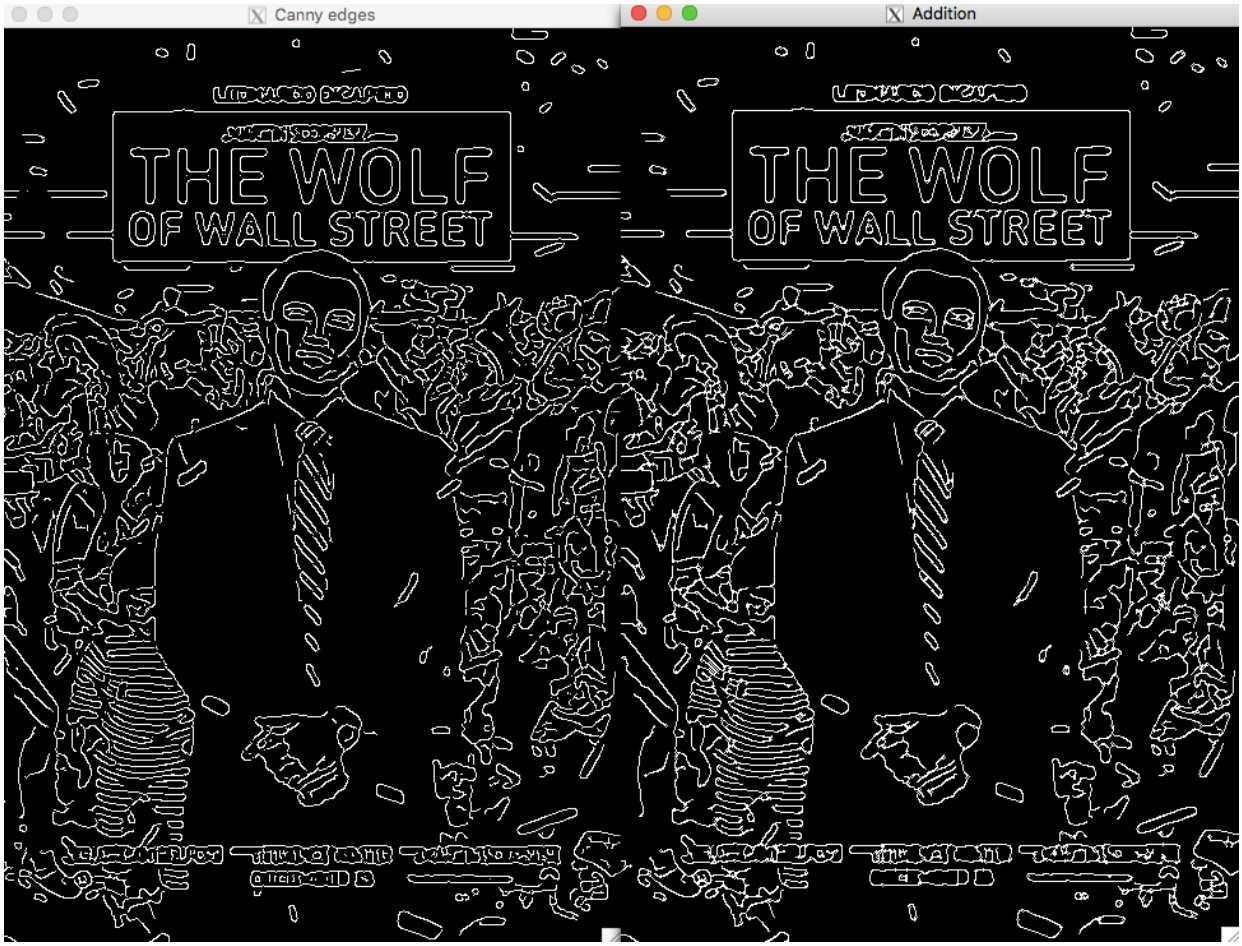
stpietro



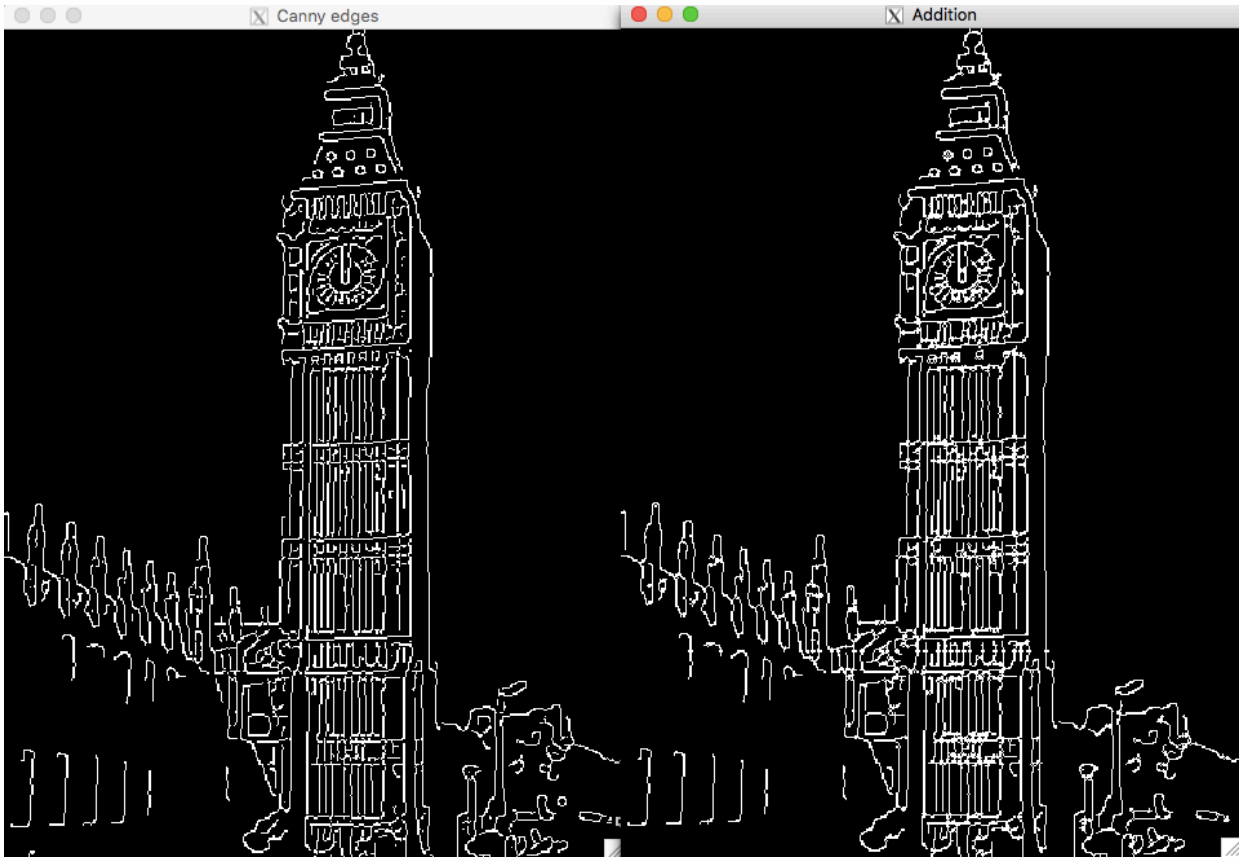
scale=5

lena

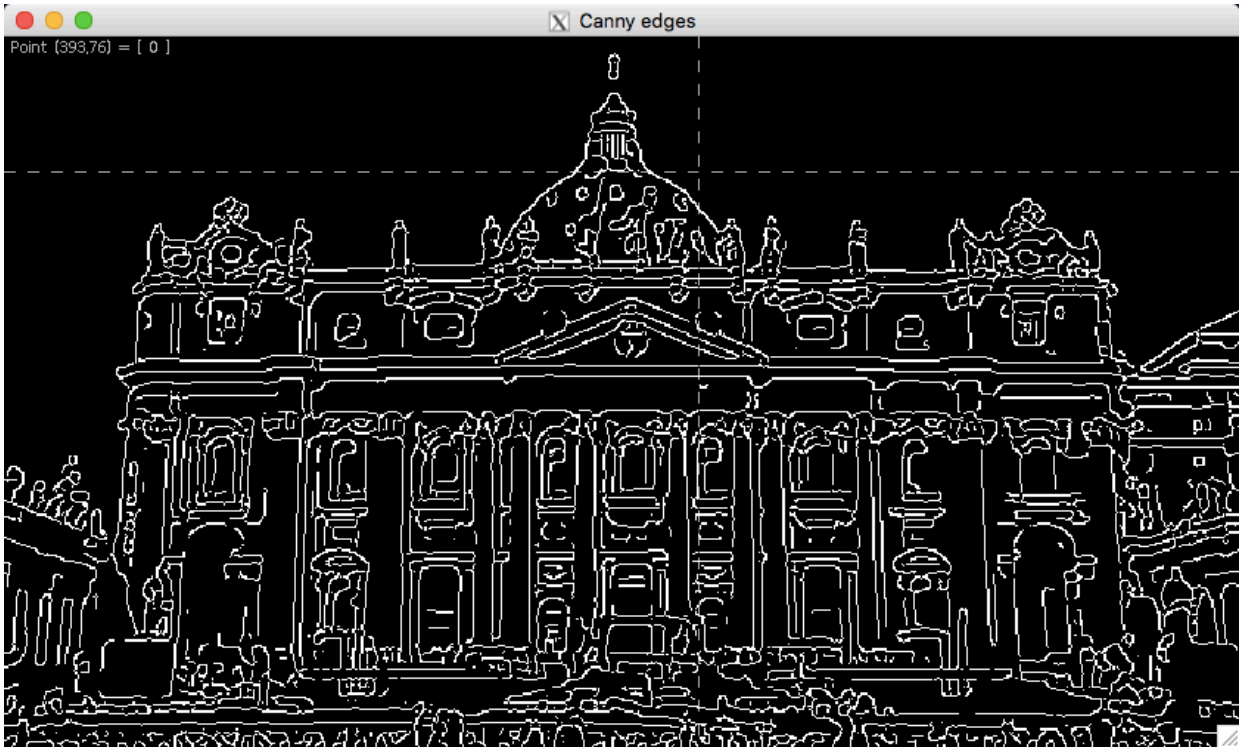
twows

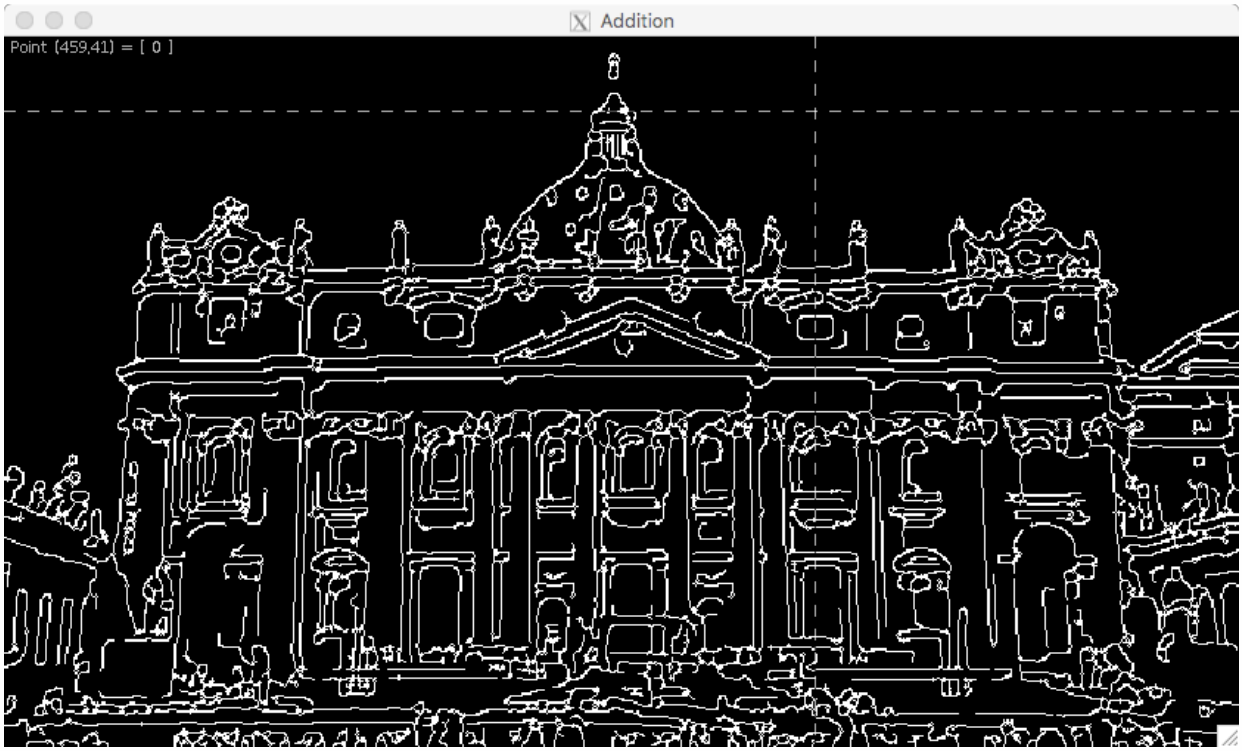


bigben



stpietro

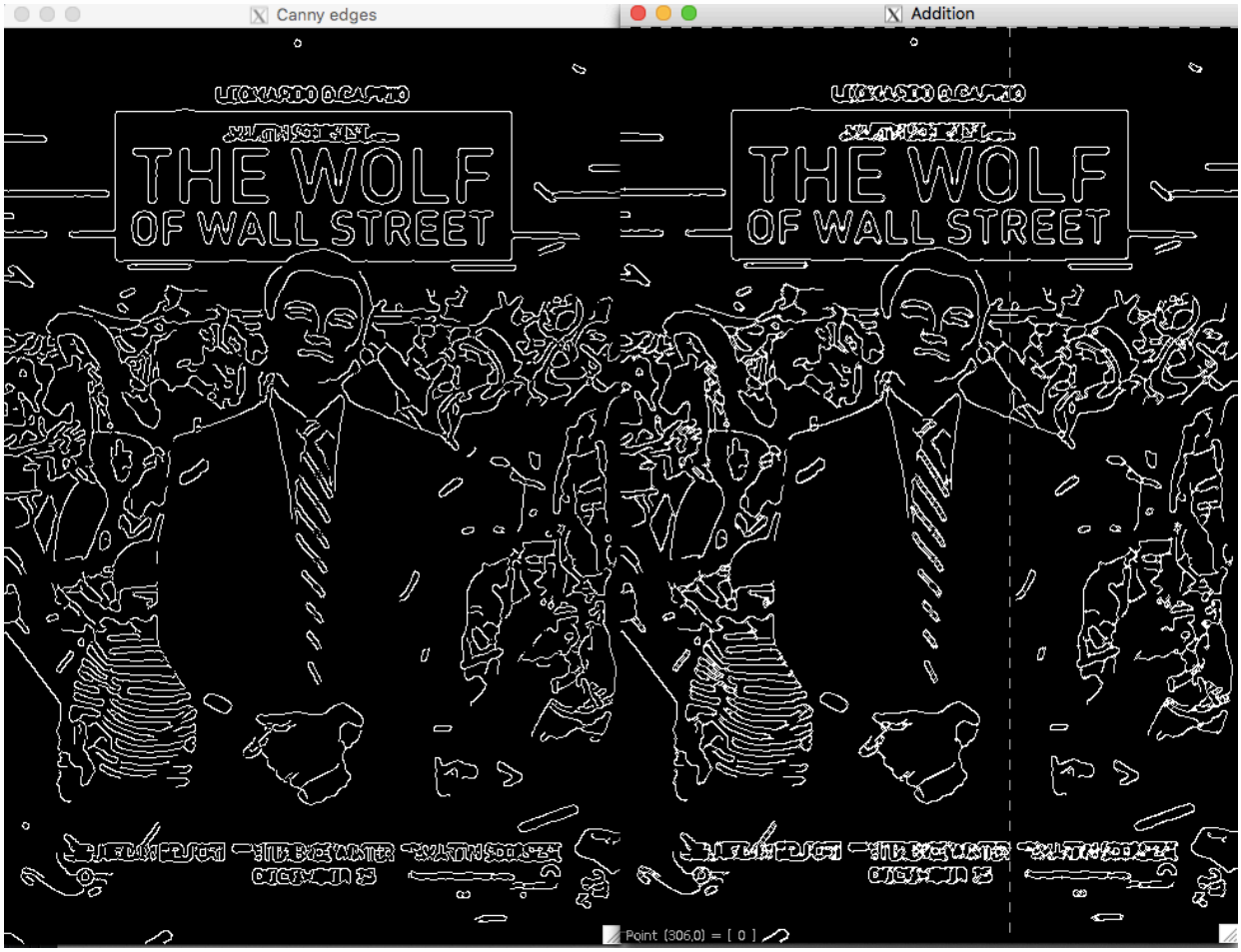




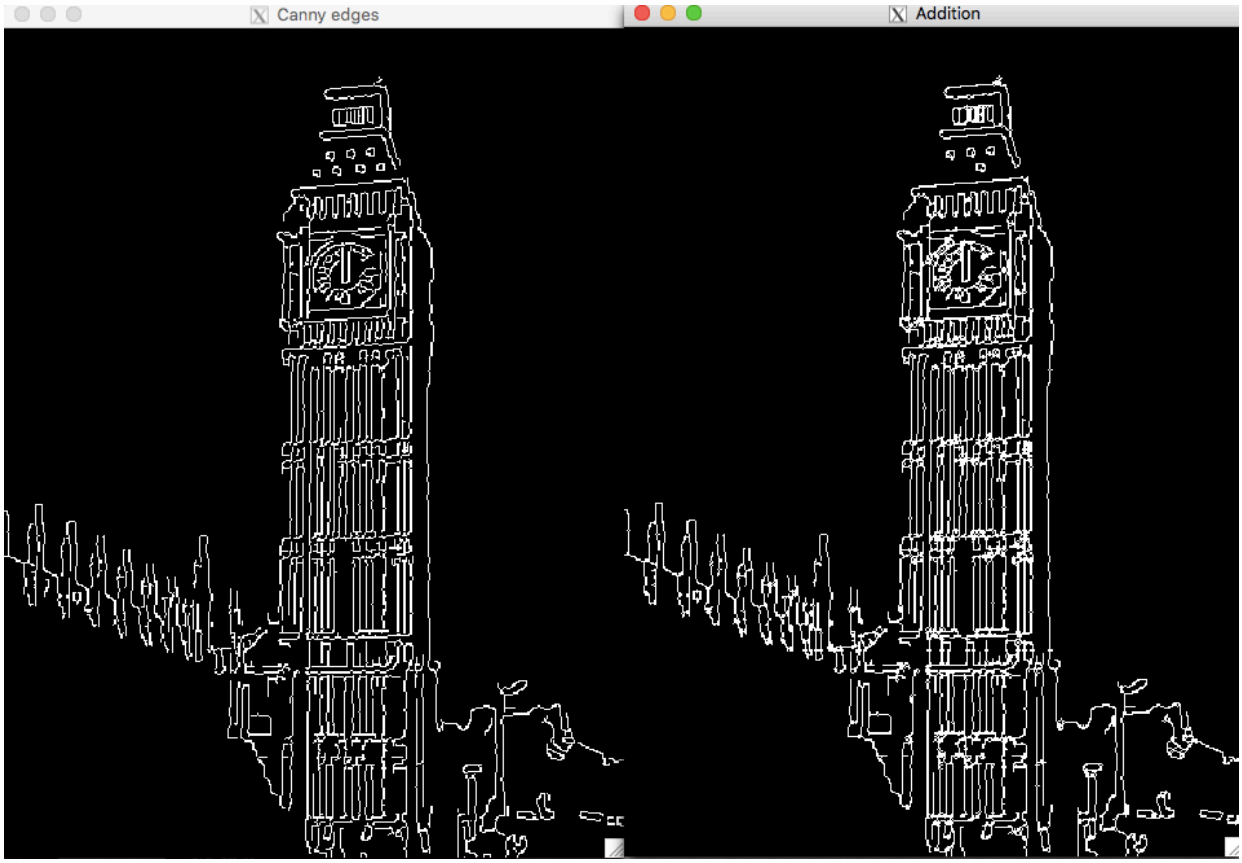
low thresh = 40, high thresh = 80

lena

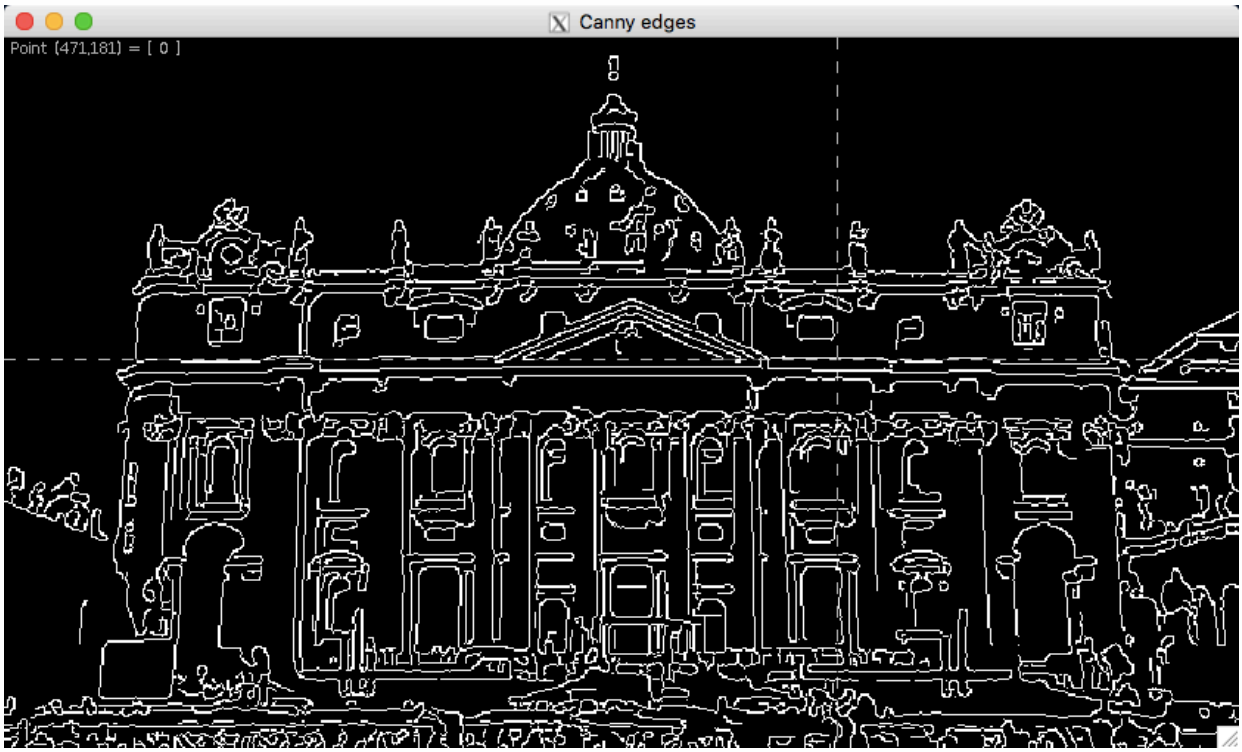
twows



bigben



stpietro



分析

从上面几张图片对比可以分析出各参数影响如下：

1. 高斯模糊的 sigma 越大，对于范围更大的图像边缘更有可能检测出来，而小的 sigma对于图像细节的边缘检测效果更好。
2. 高斯模糊的核大小因为高斯分布的性质而在 sigma 较小时对效果影响较小，在 sigma 足够大时核大小的增大可以使得大范围的边缘检测效果更好，但是核大小的增大也会增加高斯模糊运行的时间。
3. 双阈值过大可能会导致处理后剩下的边缘数量较少，过小可能会造成处理后剩下许多的伪边缘，因此需要找到较为合适的值才能使效果较优。