Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

ОТЧЕТ
по лабораторной работе

«Выделение границ объектов»

Разработка графических приложений

Работу выполнил студент

гр. 3540901/91502 ____ Дьячков В.В.

Работу принял преподаватель

_____ Абрамов Н.А.

Санкт-Петербург

2019

Содержание

1. Программа работы

Реализовать алгоритмы выделения границ объектов с использованием:

1. оператора Робертса;
2. оператора Собеля;
3. оператора Прюитта.

2. Выполнение работы

2.1. Операция свертки

Свертка является общим методом обработки изображений, изменяющим интенсивность пикселя для отражения интенсивности окружающих пикселей. Используя свертку, можно получить различные эффекты изображений, в том числе и обнаружение границ. Главным элементом свертки является ядро свертки – матрица (чаще всего используется квадратная матрица).

При вычислении нового значения выбранного пикселя изображения, ядро свертки накладывается своим центром на этот пиксель. Соседние пиксели так же накрываются ядром. Затем вычисляется сумма произведений значений пикселей изображения на значения, накрывшего данный пиксель элемента ядра. Полученная сумма и является новым значением выбранного пикселя. На рис. 2.1 приведен пример применения ядра к одному из пикселей изображения.

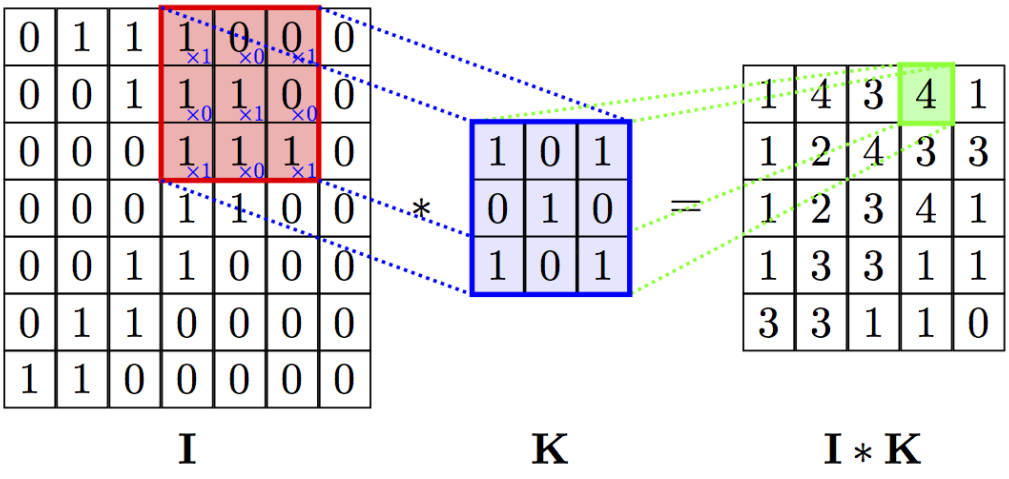

Рис. 2.1: Применение свертки к изображению

Реализуем операцию свертки при помощи языка Python.

```
def apply_kernel(img, kernel):
    K = kernel.shape[0]
    height = img.shape[0] - (K // 2) * 2
    width = img.shape[1] - (K // 2) * 2
    res = np.zeros((height, width))
    for i in np.arange(height):
        for j in np.arange(width):
            m = img[i:i+K, j:j+K] * kernel
            res[i][j] = m.sum()
    return np.clip(res, 0, 255).astype(np.uint8)
```

Убедимся, что разработанная функция свертки изображения работает верно, сравнив результат с функцией `cv.filter2D`, находящейся в библиотеке OpenCV. Для этого подадим функциям на вход одно и то же изображение и ядро. Дополнительно измерим время работы функций при помощи magic-команды Jupyter – `%%timeit`. Строки, начинающиеся с `##` обозначают вывод команды.

```
kernel = np.array([[1, 0], [0, -1]])
spb_apply_kernel = apply_kernel(spb, kernel)
spb_cv_filtered = cv.filter2D(spb, -1, kernel)
np.all(spb_cv_filtered[1:-1, 1:-1] == spb_apply_kernel)
## True

%%timeit
apply_kernel(spb, kernel)
## 1.83 s ± 71.8 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

%%timeit
cv.filter2D(spb, -1, kernel)
## 154 µs ± 6.07 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

Видно, что результаты работы функций идентичны, с точностью до рамок изображения. Дело в том, что разработанная функция возвращает изображение, меньшее, чем оригинальное, на $2 * (K//2)$ пикселей (K – размер ядра, $//$ – целочисленное деление). Библиотечная функция `cv.filter2D` возвращает изображение с таким же размером, что и входное.

Время работы библиотечной функции оказалось на несколько порядков меньше. Скорее всего, это связано с ее более эффективной реализацией внутри библиотеки (например, при помощи распараллеливания применения свертки). В дальнейшем будем использовать библиотечную реализацию свертки.

2.2. Оператор Робертса

Перекрестный оператор Робертса (Roberts) – один из ранних алгоритмов выделения границ, который вычисляет на плоском дискретном изображении сумму квадратов разниц между диагонально смежными пикселами. Это может быть выполнено сверткой изображения с двумя ядрами:

$$h_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \; h_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Итоговая величина перепада G получемого изображения получается по одной из формул:

$$\begin{aligned} G &= \sqrt{(h_1 * I)^2 + (h_2 * I)^2} \\ &= |h_1 * I| + |h_2 * I| \,, \end{aligned}$$

где I – исходное изображение, а $*$ – операция свертки. Первая формула использует Евклидову метрику, а вторая – манхэттенскую.

Реализуем функцию для применения оператора Робертса. Будем использовать манхэттенскую метрику для комбинации получаемых после сверток изображений.

```
h1 = np.array([
    [1, 0],
    [0, -1]
])
h2 = np.array([
    [0, 1],
    [-1, 0]
])
def roberts(img):
    img_h1 = cv.filter2D(img, -1, h1)
    img_h2 = cv.filter2D(img, -1, h2)
    img_roberts = np.abs(img_h1) + np.abs(img_h2)
    return img_roberts, img_h1, img_h2
```

Листинг 1: Фунукция для применения оператора Робертса

Применим оператор Робертса к нескольким изображениям.

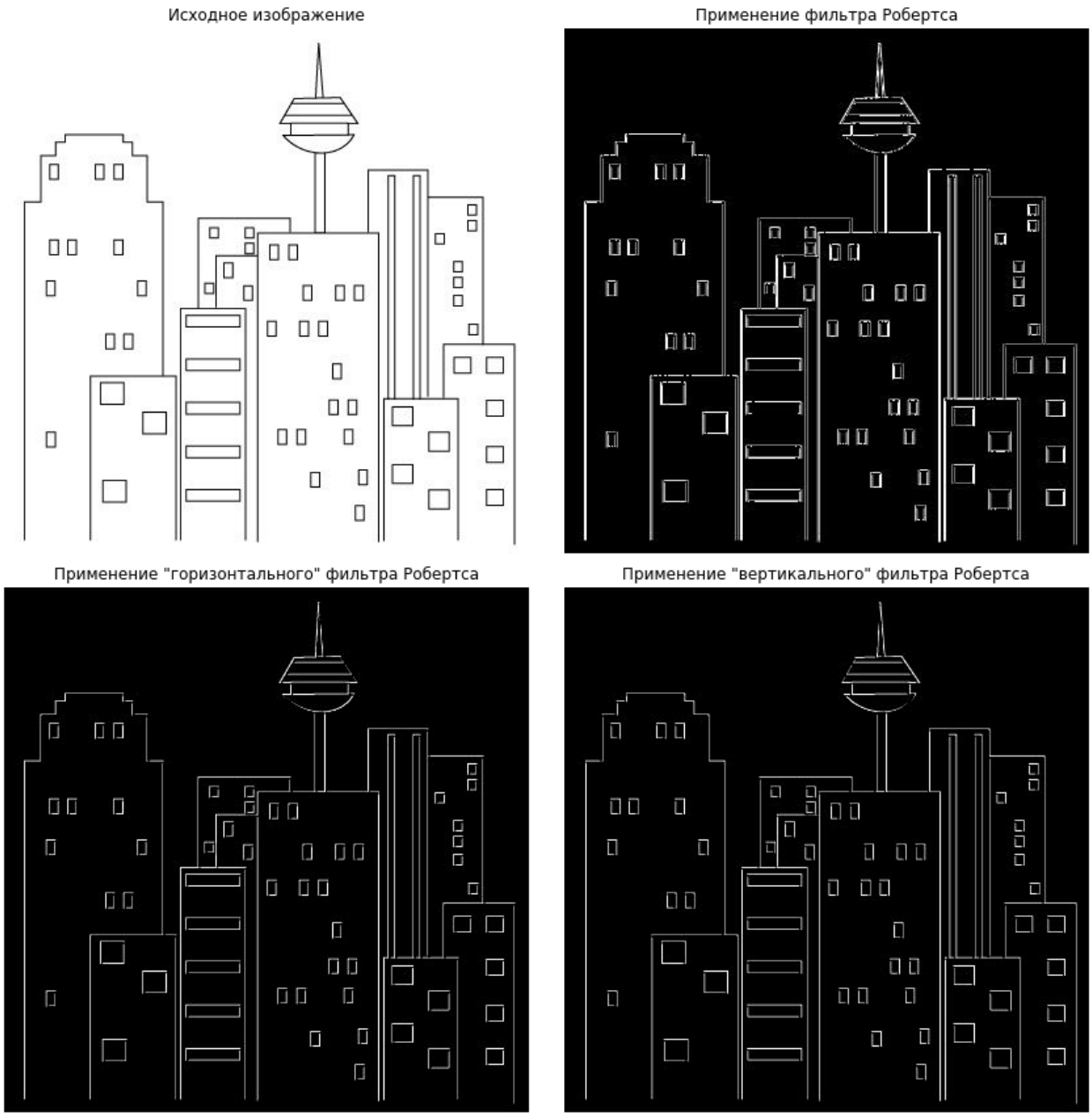

Рис. 2.2: Применение оператора Робертса (1)

Рис. 2.3: Применение оператора Робертса (2)

Рис. 2.4: Применение оператора Робертса (3)

2.3. Оператор Собеля

Применим другой оператор для детектирования границ – оператор Собеля (Sobel). Он основан на свертке изображения небольшими сепарабельными целочисленными фильтрами в вертикальном и горизонтальном направлениях, поэтому его относительно легко вычислять. С другой стороны, используемая им аппроксимация градиента достаточно грубая, особенно это сказывается на высокочастотных колебаниях изображения.

Оператор вычисляет градиент яркости изображения в каждой точке. Так находится направление наибольшего увеличения яркости и величина ее изменения в этом направлении. Результат показывает, насколько «резко» или «плавно» меняется яркость изображения в каждой точке, а значит, вероятность нахождения точки на грани, а также ориентацию границы.

Оператор использует ядра 3×3, с которыми сворачивается исходное изображение для вычисления приближенных значений производных по горизонтали и по вертикали.

$$x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \; y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} = x^T$$

В каждой точке изображения приближенное значение величины и направления градиента можно вычислить путем использования полученных приближенных значений производных:

$$EdgeMagnitude = \sqrt{x^2 + y^2}, \; EdgeDirection = \tan^{-1} \frac{y}{x}$$

Реализуем функцию для применения оператора Собеля.

```
sx = np.array([
    [-1, 0, 1],
    [-2, 0, 2],
    [-1, 0, 1]
])
sy = sx.T
def sobel(img):
    img_x = cv.filter2D(img, -1, sx).astype(np.single)
    img_y = cv.filter2D(img, -1, sy).astype(np.single)
    img_sobel = np.sqrt(img_x ** 2 + img_y ** 2)
    return img_sobel, img_x, img_y
```

Листинг 2: Применение оператора Собеля

Применим оператор Собеля к нескольким изображениям.

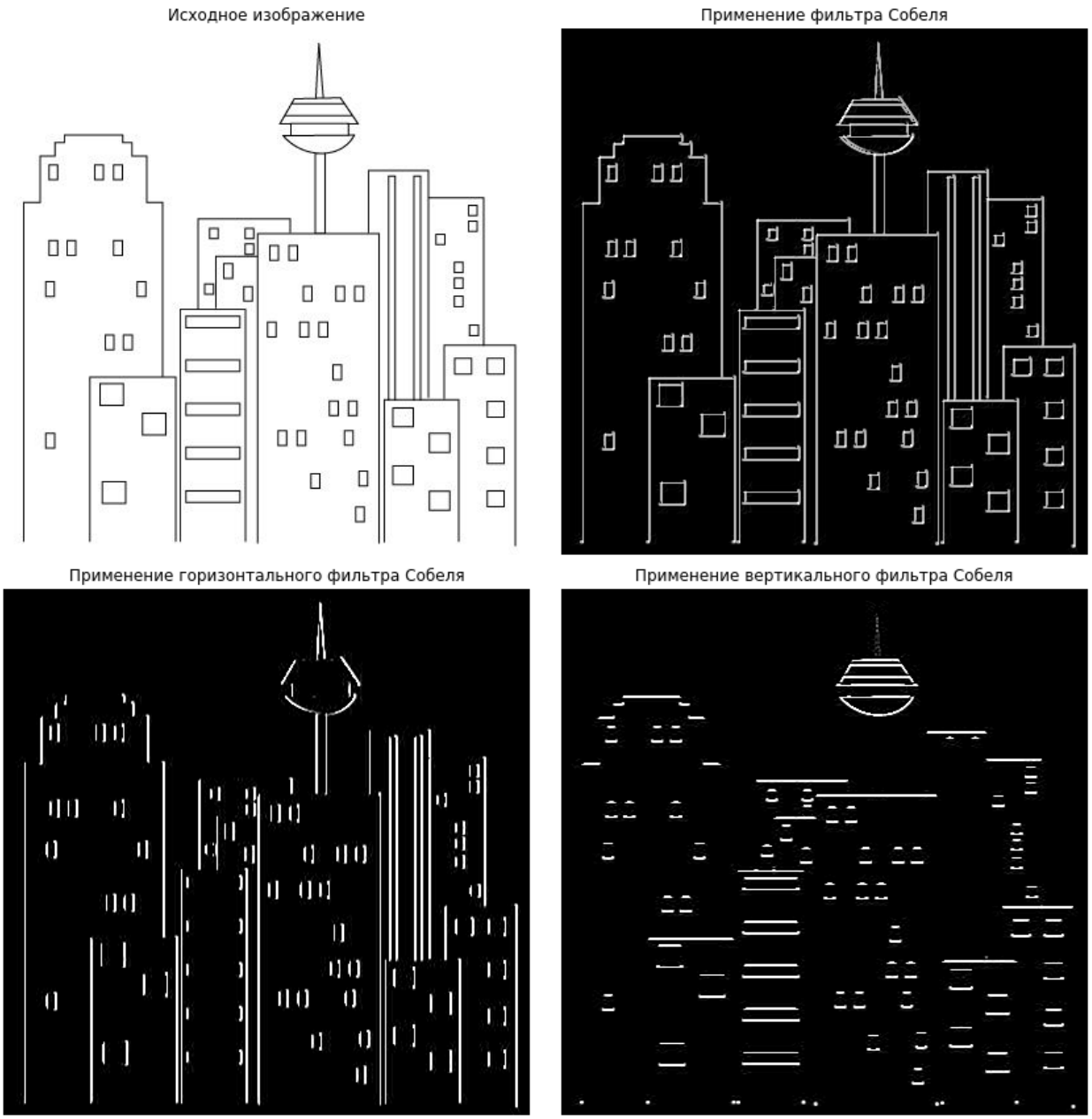

Рис. 2.5: Применение оператора Собеля (1)

Рис. 2.6: Применение оператора Собеля (2)

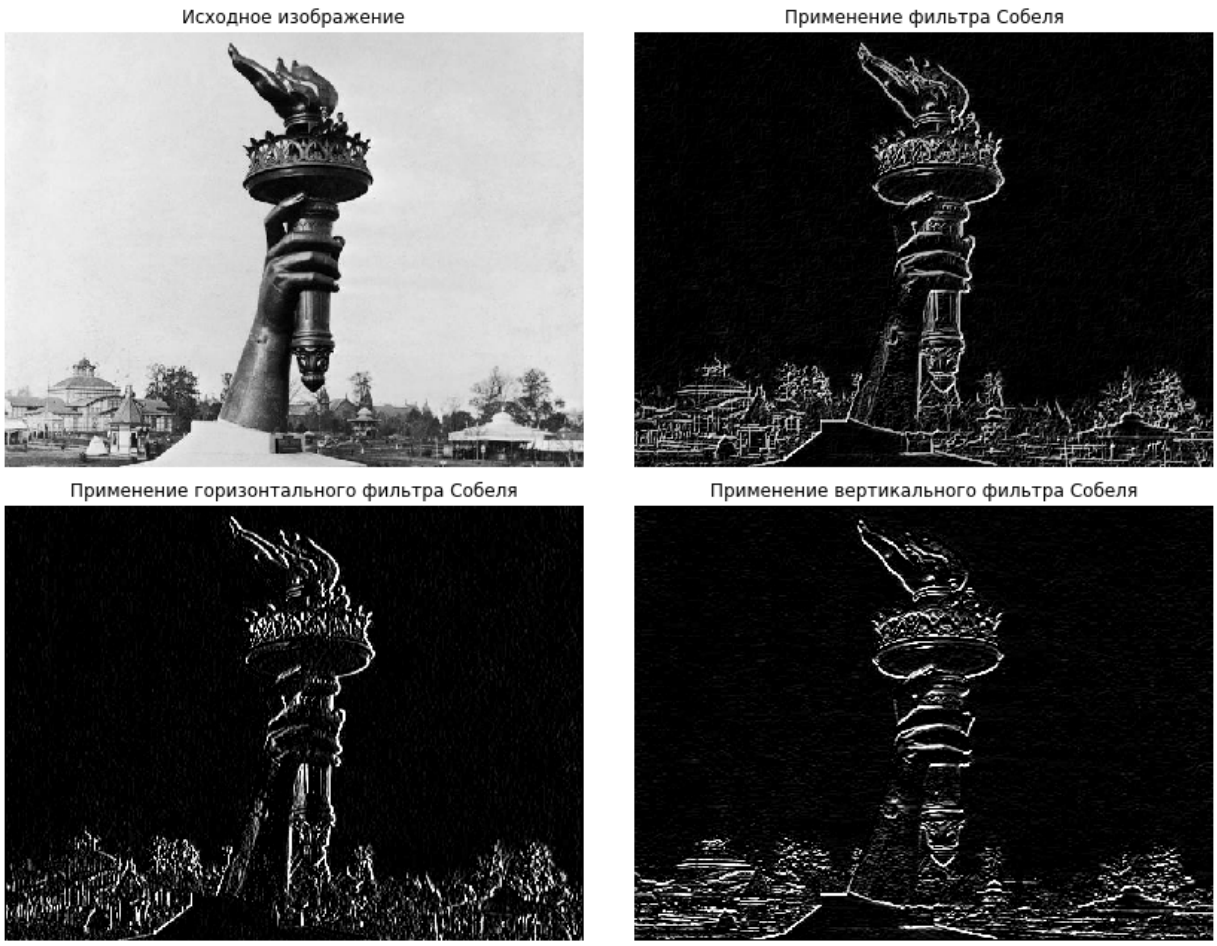

Рис. 2.7: Применение оператора Собеля (3)

2.4. Оператор Прюитта

Воспользуемся еще одним оператором для детектирования границ – оператором Прюитта (Prewitt). Этод метод вычисляет максимальный отклик на ядрах свертки для нахождения локальной ориентации границы в каждом пикселе. Этот оператор совпадает с оператором Собеля, но использует другие ядра:

$$x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \; y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} = x^T$$

Оператор вычисляет градиент яркости изображения в каждой точке. Так находится направление наибольшего увеличения яркости и величина ее изменения в этом направлении. Результат показывает, насколько «резко» или «плавно» меняется яркость изображения в каждой точке, а значит, вероятность нахождения точки на грани, а также ориентацию границы.

Реализуем функцию для применения оператора Прюитта.

```
px = np.array([
    [-1, 0, 1],
    [-1, 0, 1],
    [-1, 0, 1]]
)
py = px.T
def prewitt(img):
    img_x = cv.filter2D(img, -1, px).astype(np.single)
    img_y = cv.filter2D(img, -1, py).astype(np.single)
    img_prewitt = np.sqrt(img_x ** 2 + img_y ** 2)
    return img_prewitt, img_x, img_y
```

Листинг 3: Применение оператора Собеля

Применим оператор Прюитта к нескольким изображениям.

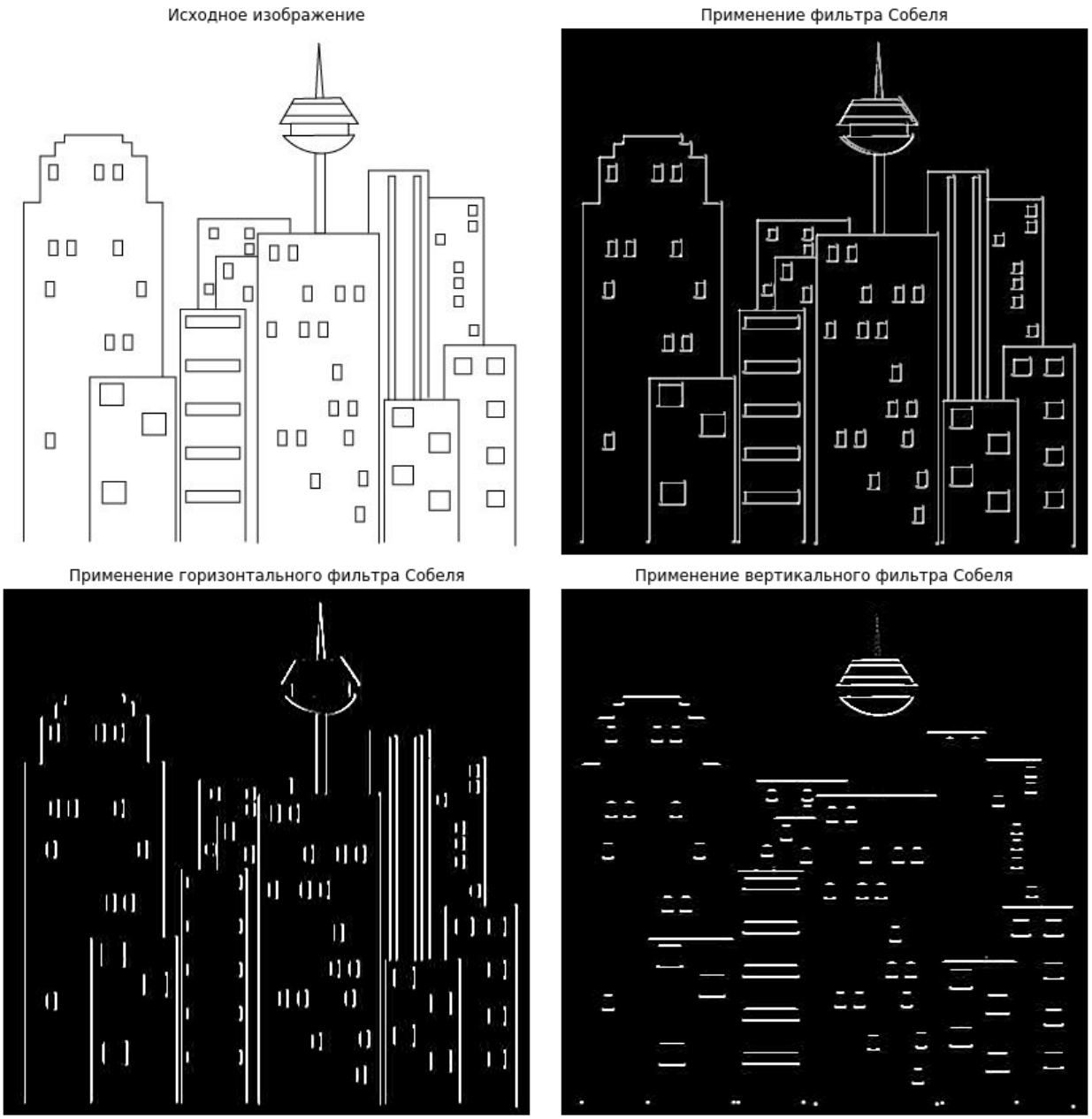

Рис. 2.8: Применение оператора Собеля (1)

Рис. 2.9: Применение оператора Собеля (2)

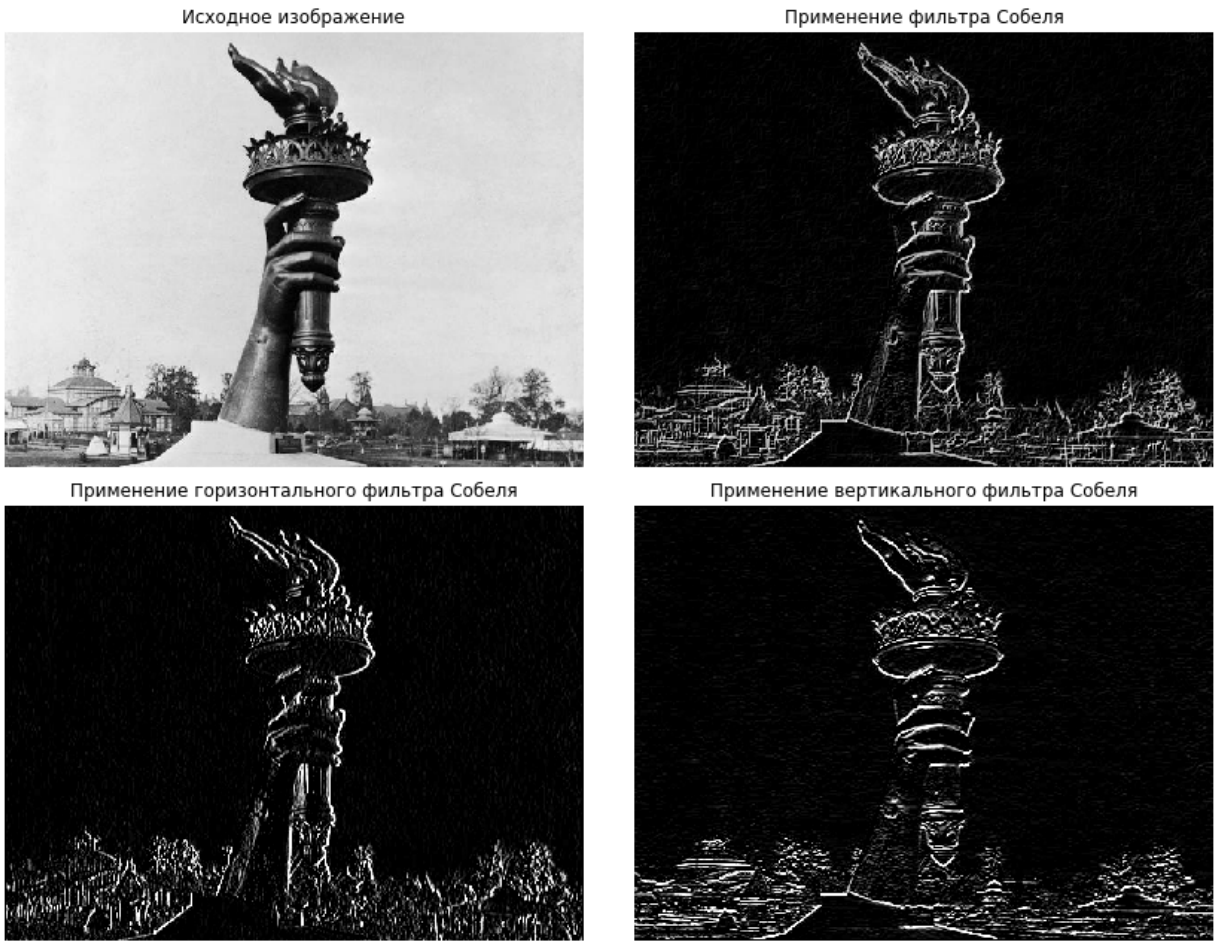

Рис. 2.10: Применение оператора Собеля (3)

3. Сравнение операторов

Сравним результаты применения операторов к одному и тому же изображения. Для удобства выведем их рядом друг с другом.

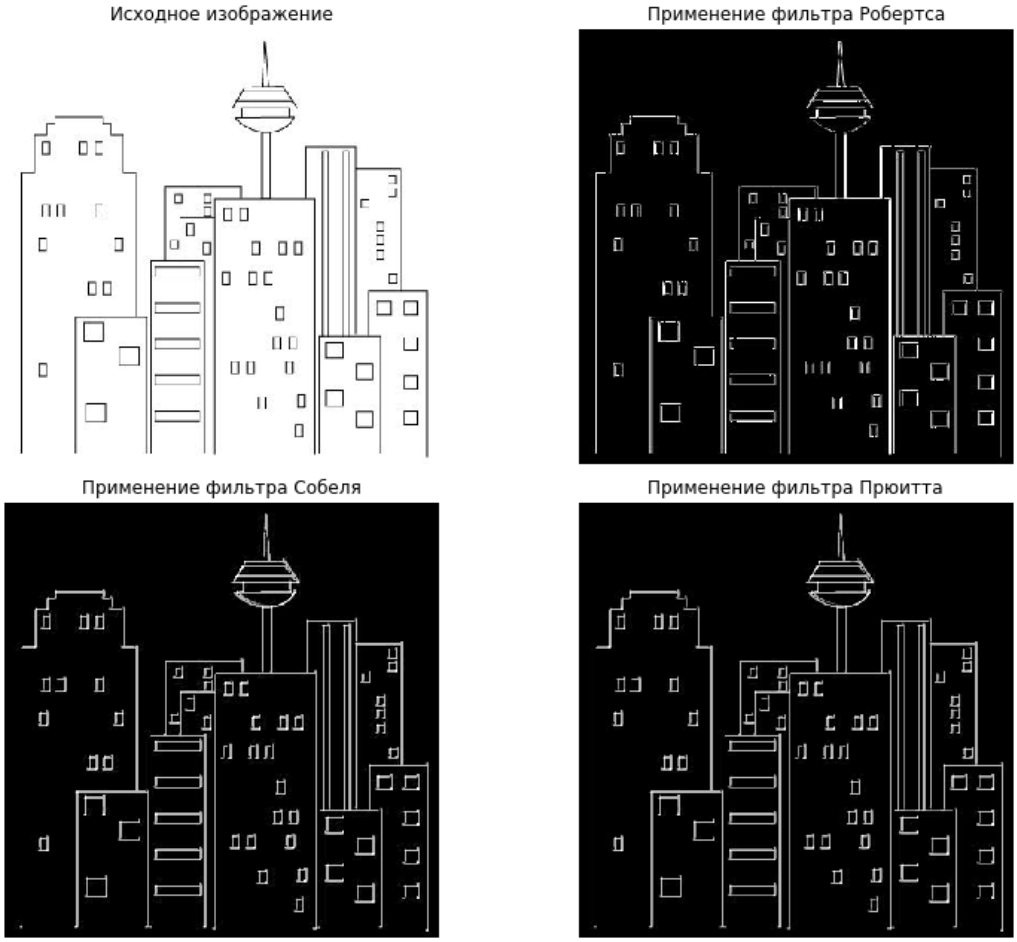

Рис. 3.1: Применение всех операторов (1)

Рис. 3.2: Применение всех операторов (2)

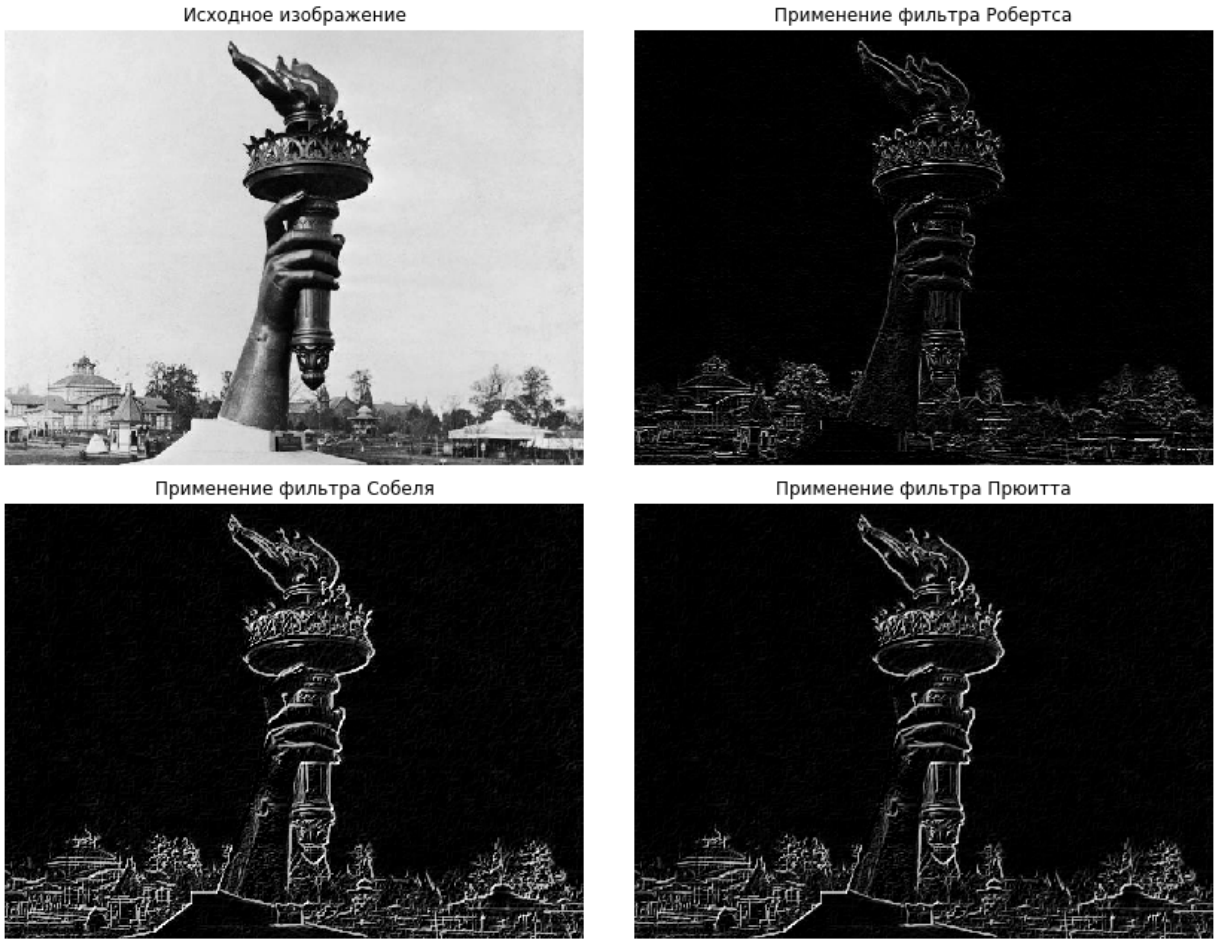

Рис. 3.3: Применение всех операторов (3)

4. Выводы

В данной работе были реализованы три оператора для детектирования границ на изображении:

- оператор Робертса;
- оператор Собеля;
- оператор Прюитта.

Из получившихся изображений видно, что результаты применения операторов Собеля и Прюитта оказались очень схожи. Оба оператора высчитывают аппроксимацию горизонтально и вертикального градиента изображения, позволяя получить хорошую карту границ изображения. В то же время, оператор Робертса использует подход с поиском диагональных градиентов и работает хуже, упуская границы некоторых предметов на изображении.