cv2.findContours()
Функция findContours возвращает сгруппированные наборы точек, которые являются точками контура (или концами отрезков контура, в зависимости от типа аппроксимации)

Вернёт точки (аппроксимация None)
```
cv2.findContours(thresh_img, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
```

Вернёт отрезки (аппроксимация chain_simple)
```
cv2.findContours(thresh_img, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

```
contours, hierarchy = cv2.findContours(thresh_img, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
print(type(contours),type(hierarchy))
```
Получим вывод
<class 'tuple'> <class 'numpy.ndarray'>

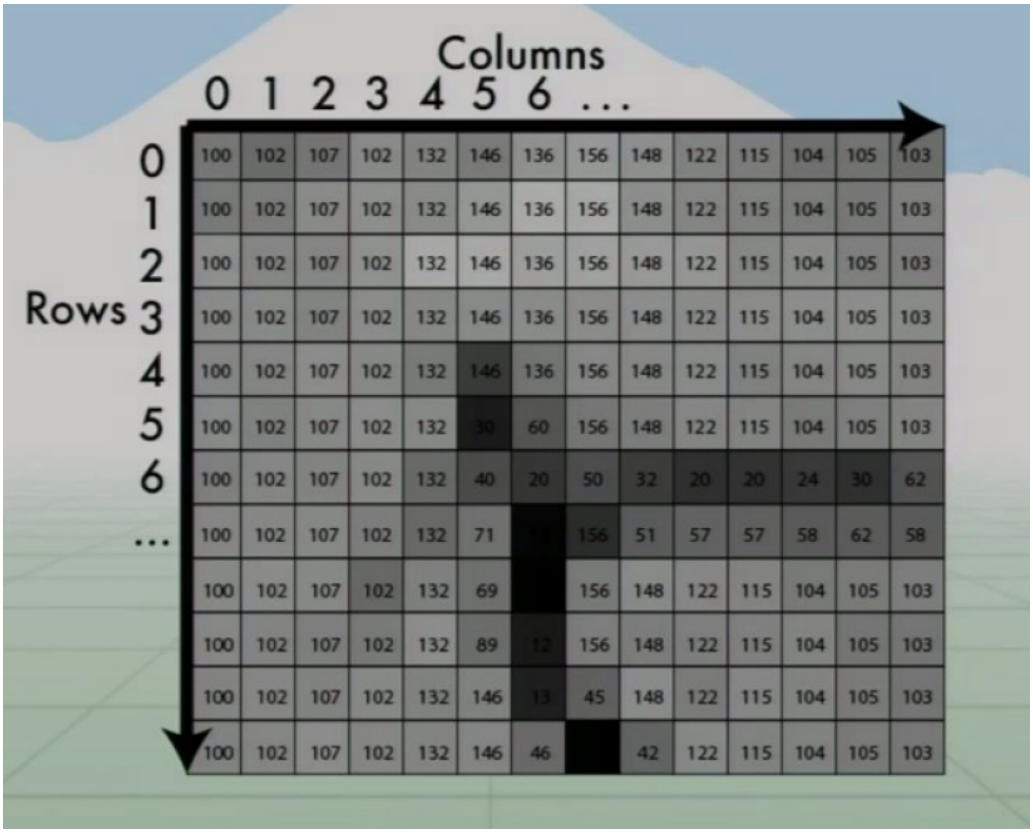
Таким образом, сам контур – это обыкновенный тьюпл, а второе возвращенное значение массив numpy. Элементами тьюпла являются массивы numpy.

Полученные контуры можно нарисовать с помощью функции cv2.drawContours
```
cv2.drawContours(img_contours, [sel_countour], -1, (255,255,255), 1)
```

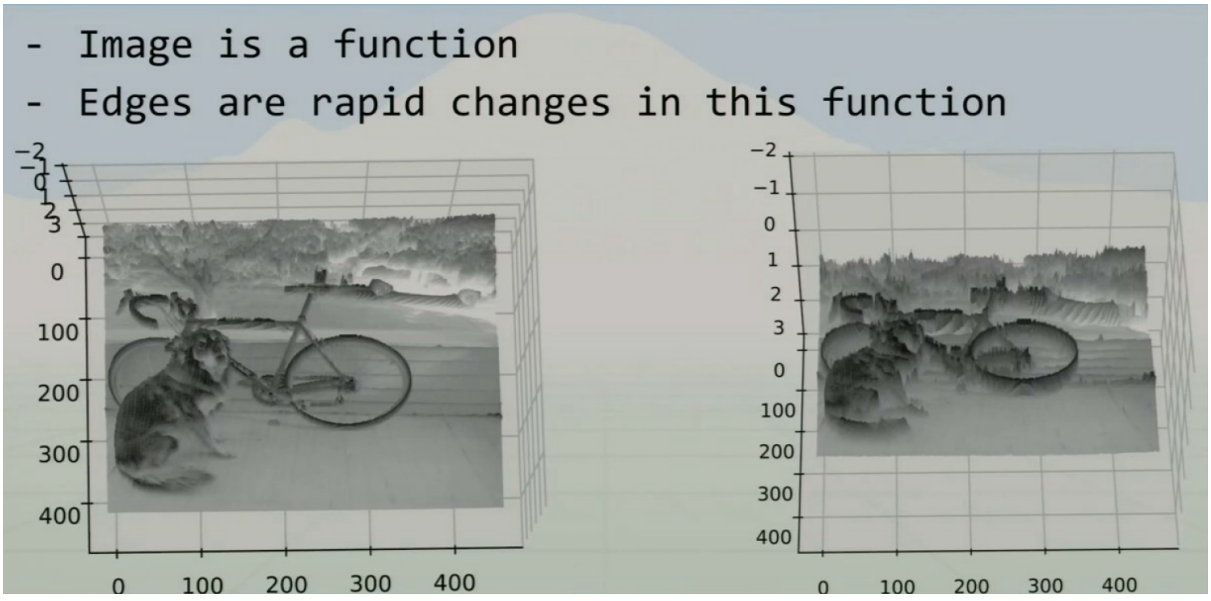
Если включена аппроксимация (CHAIN_APPROX_SIMPLE), том можно нарисовать контур линиями
```
last_point=None
for point in sel_countour:
    curr_point=point[0]
    if not(last_point is None):
        x1=int(last_point[0])
        y1=int(last_point[1])
        x2=int(curr_point[0])
        y2=int(curr_point[1])
        cv2.line(img_contours, (x1, y1), (x2, y2), 255, thickness=1)
    last_point=curr_point
```

Если аппроксимация выключена мы можем нарисовать контур по точкам
```
for point in sel_countour:
    y=int(point[0][1])
    x=int(point[0][0])
    img_contours[y,x]=255
```

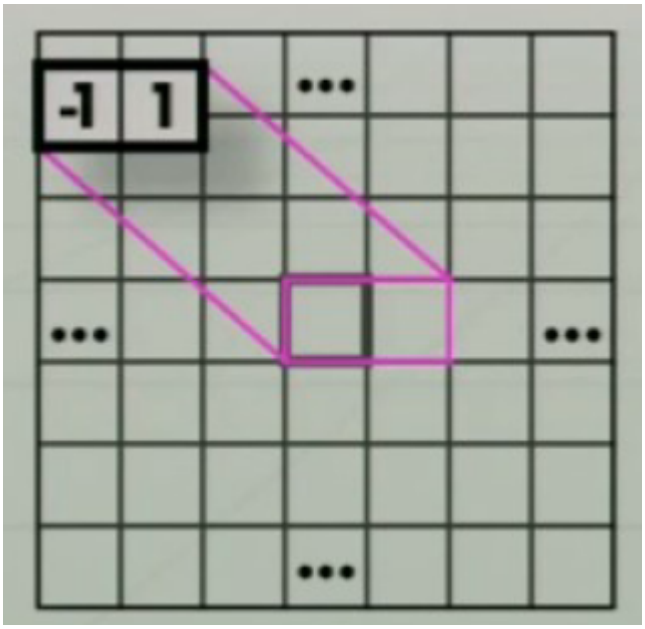
Перед тем как обсудить методы нахождения границ (краев), нужно понять что есть граница на изображении.
Для простоты рассмотрим черно-белое изображение



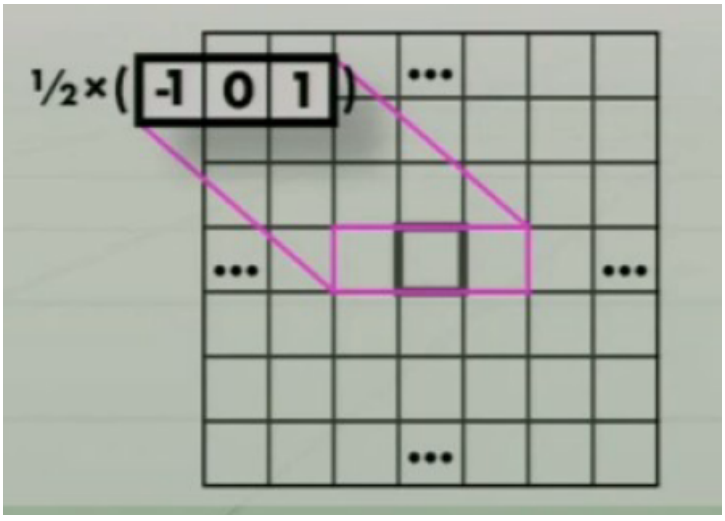
Яркость на изображении задаётся целым числом от 0 до 255
Понятно, что граница объекта на таком изображении это резкий переход от светлого пикселя к тёмному или наоборот.
Если представить черно-белое изображение как функцию I(x,y) и нарисовать её график, то границы станет наглядно видно:



Таким образом, самый простой способ найти границу объекта - это посчитать разность между соседними пикселями (Значение пикселя x1 умножить на один, а значение пикселя x0 умножить на “-1”)

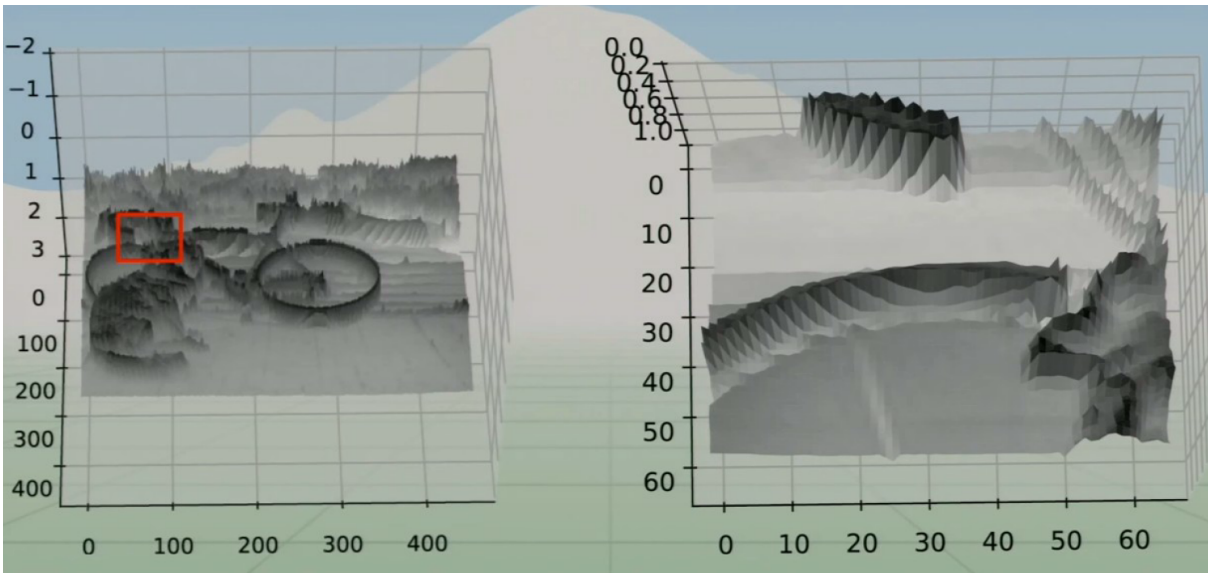


Также можно рассчитать разницу между пикселями, находящимися через один

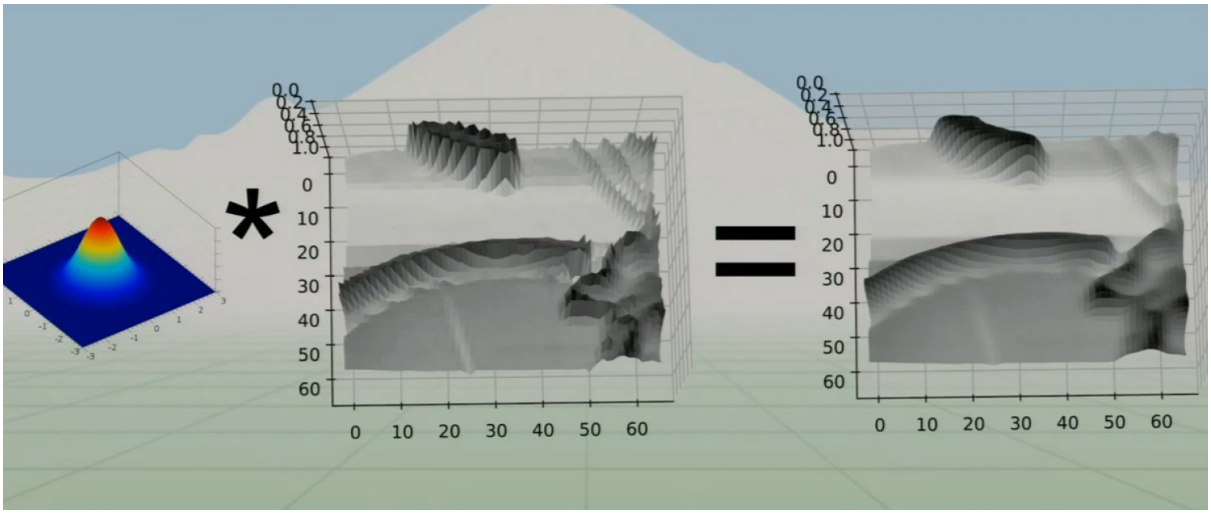


Тем самым мы найдём границы в доль направления “X”, для нахождения границ вдоль направления “Y” нужно сравнивать соседние пиксели не в строке, а в столбцах.

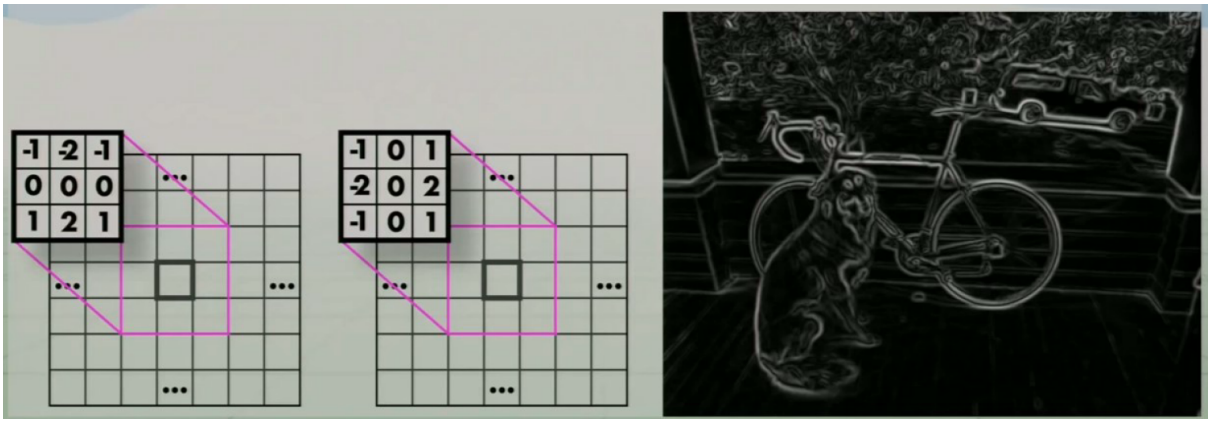
Важно помнить, что изображение часто бывает очень шумным



Для уменьшения влияния шума на нахождение границ применяют сглаживающие фильтры, например фильтр Гаусса:



Фильтр Собеля это последовательное применение к изображению двух фильтров:
Первый - фильтр Гаусса (для сглаживания), второй - 0.5 * [-1| 0 | 1].
Фильтр Собеля может действовать по двум разным направлениям



Метод нахождения границ (opencv):

1) Разность между соседними пикселями
   kernel = np.array([-1, 1])
   kernel_2 = (1/2)*np.array([-1, 0, 1])
   img_out = cv2.filter2D(img_gray, -1, kernel)
2) Фильтр Собеля:
   sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
   sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)
   действует вдоль разных направлений: X и Y
   Также есть возможность объединить результаты:
   sobel_sum=sobelx/2+sobely/2
3) Фильтр Лапласа
   Он более чувствителен к “шуму” на изображении:
   laplacian = cv2.Laplacian(img,cv2.CV_64F)
4) Фильтр Кэнни (Canny)
   В основе он также использует фильтр Собеля, но после этого дополнительно применяет к результату алгоритм NMS (Non-MaximumSuppression) - “подавление не максимумов”. Тем самым Canny оставляет на изображении только те пиксели, значение которых максимально в локальной области:
   edges = cv2.Canny(img,100,200)