Тема: Контурный анализ

Какая информация в изображении является значимой, а какая нет?

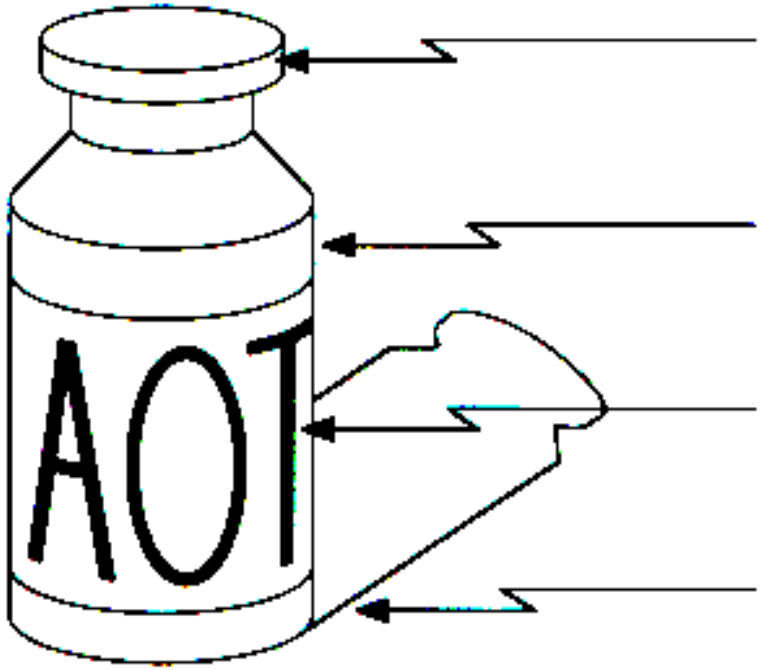
- Интуитивно понятно, что основная информация в картинке содержится как раз в границах (краях)

- **Задача:** Выделить не все контуры, а только резкие изменения (разрывы) изображения
- **Идеал:** рисунок художника

Source: D. Lowe

Откуда берутся границы

Существует множество причин формирования границ на изображении



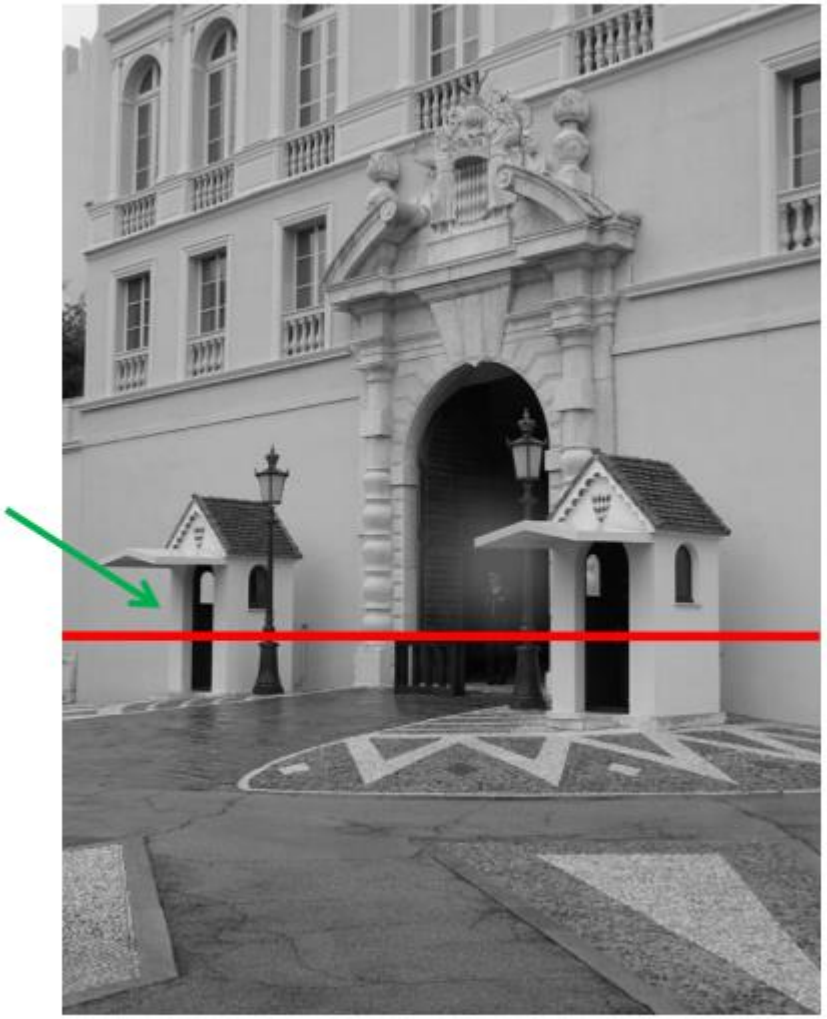
Резкое изменение нормали поверхности

Резкое изменение глубины

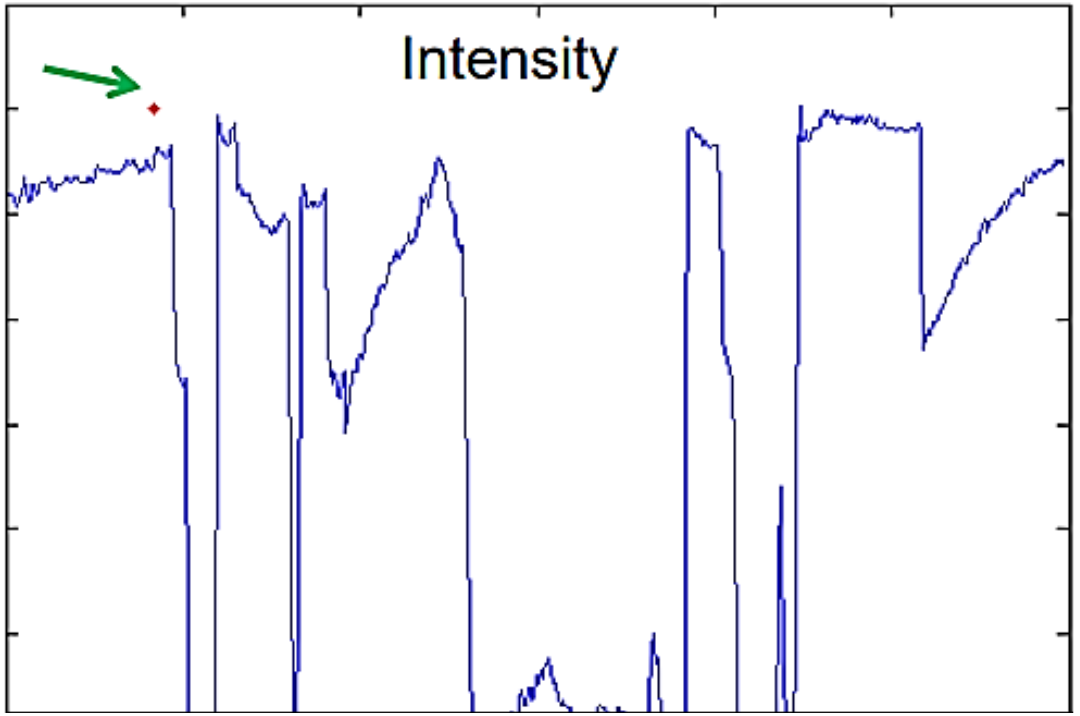
Резкое изменение цвета поверхности

Резкое изменение освещенности (мнимый контур)

Source: Steve Seitz



Функция интенсивности в одной строке матрицы изображения



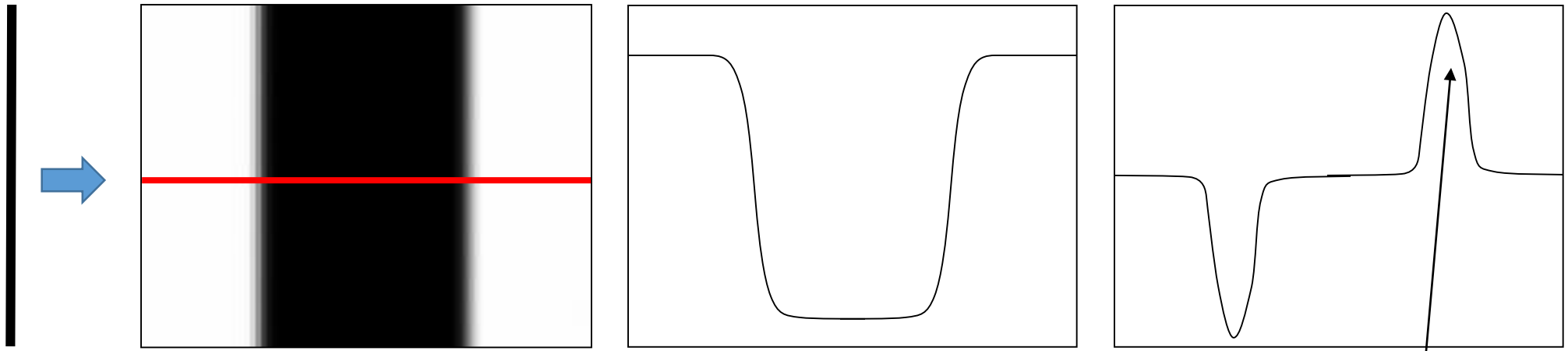
Описание «края»

Край – это точка резкого изменения значений функции интенсивности изображения

изображение

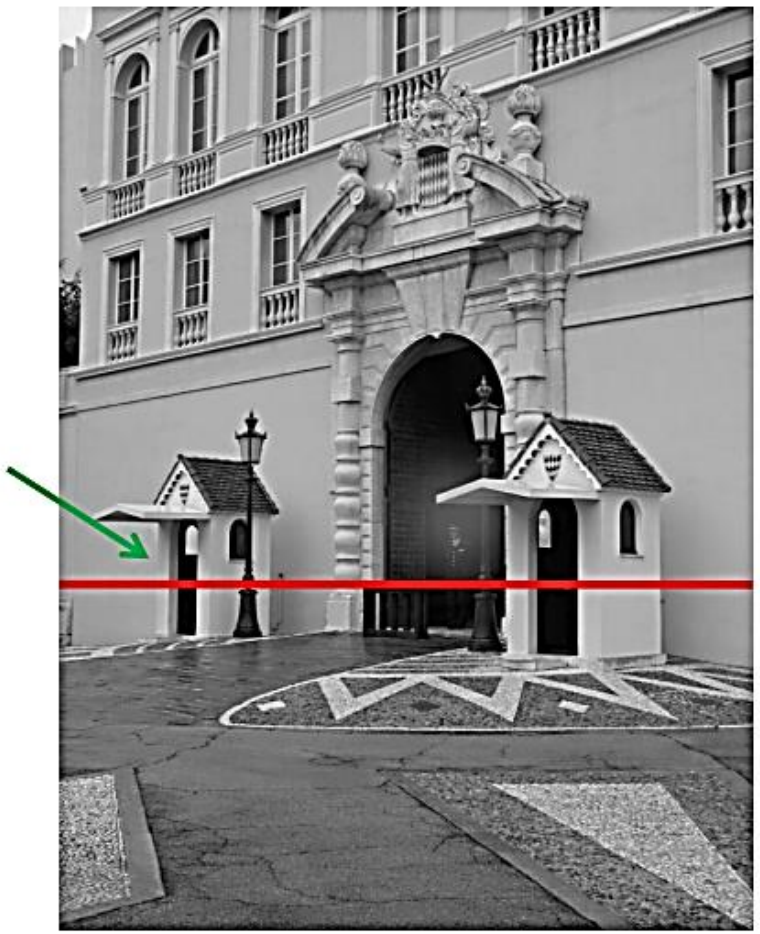
Функция интенсивности (строка изображения)

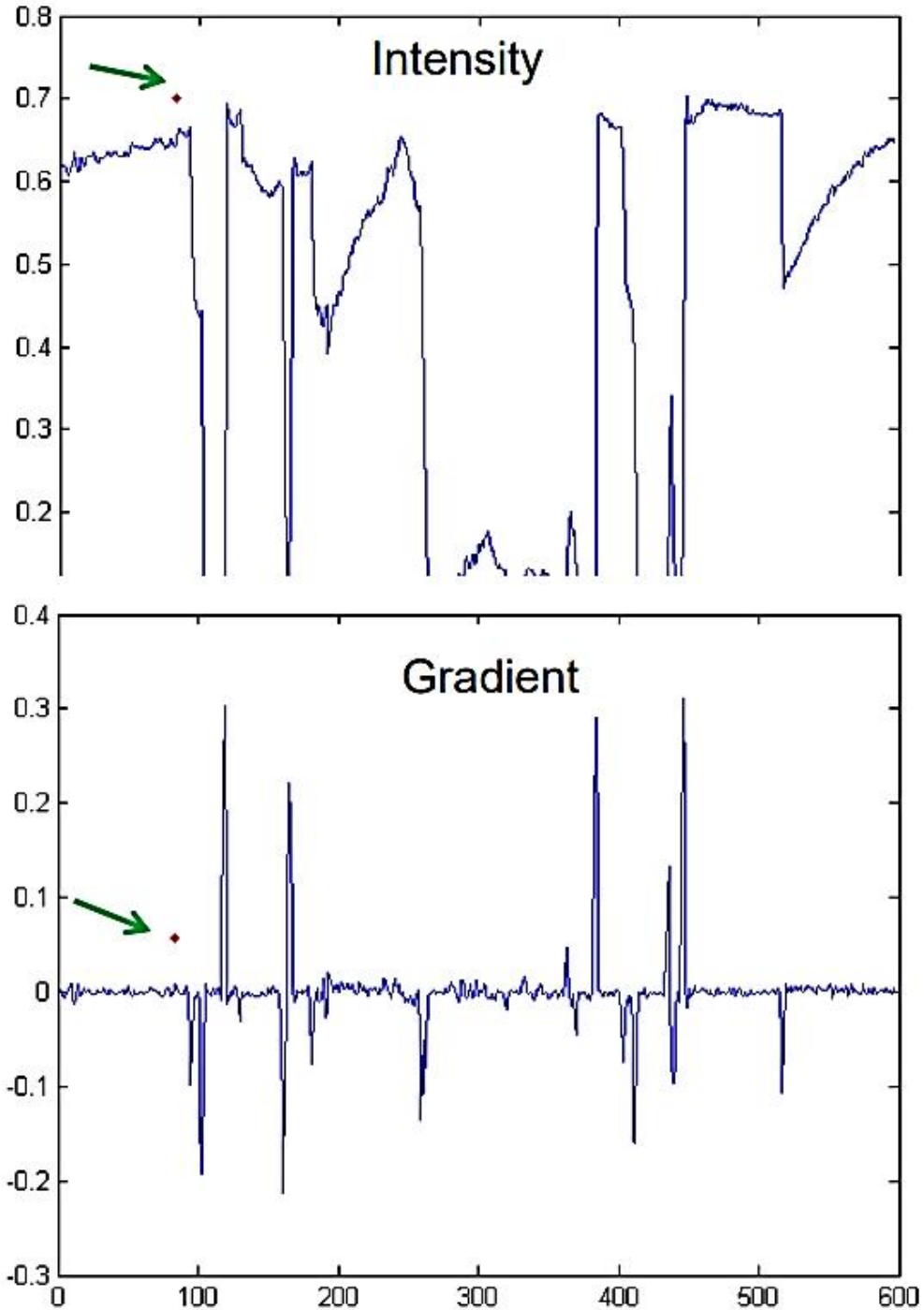
1ая производная



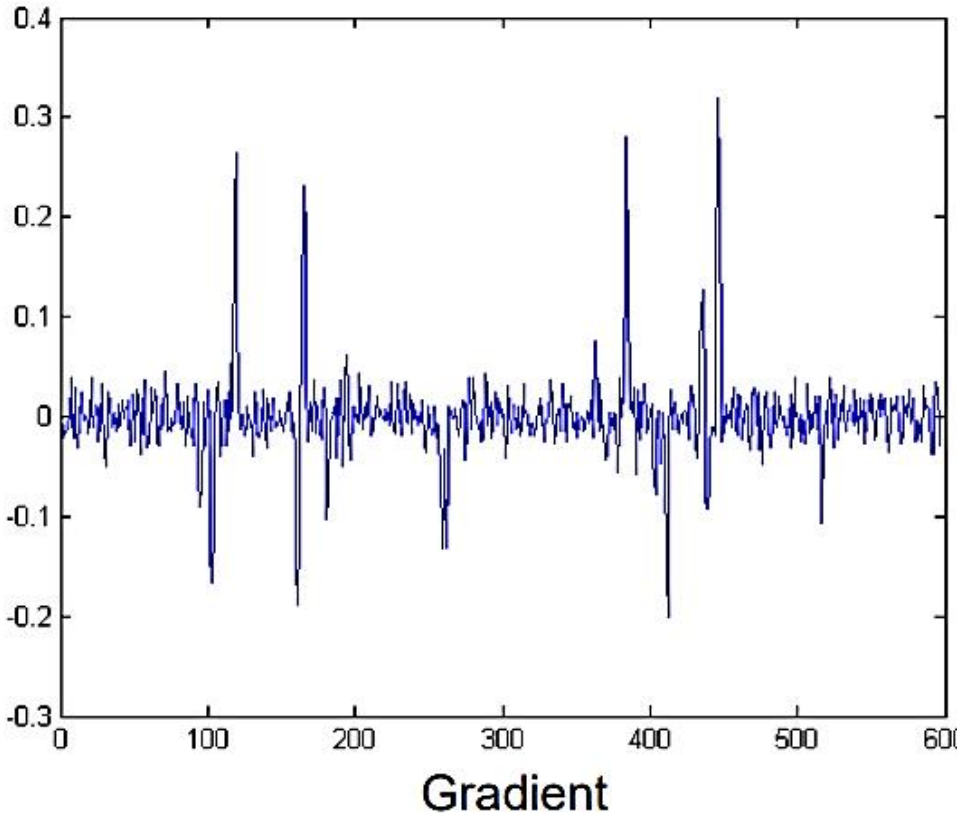
Края соответствуют экстремумам производной

Функция интенсивности

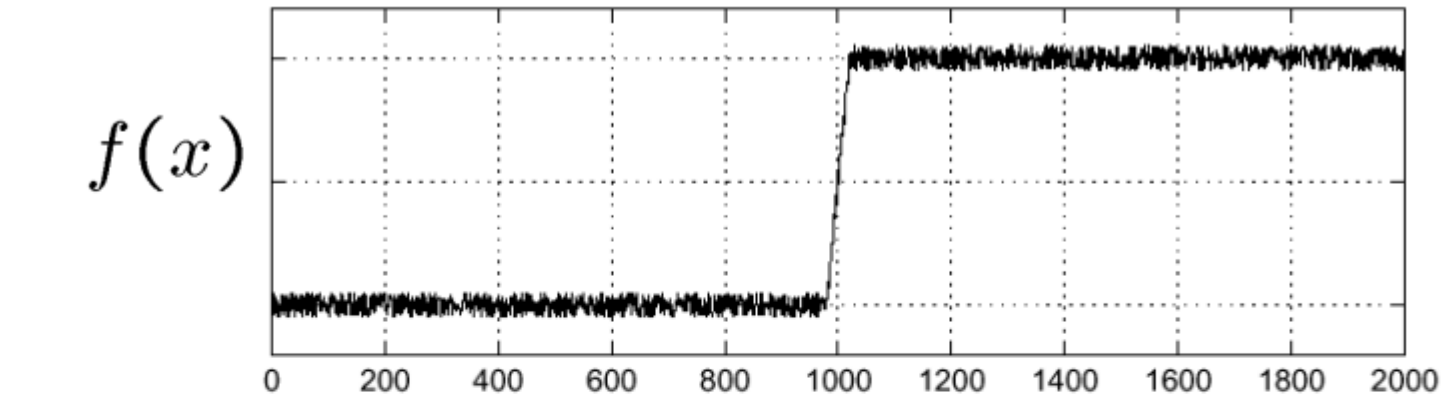


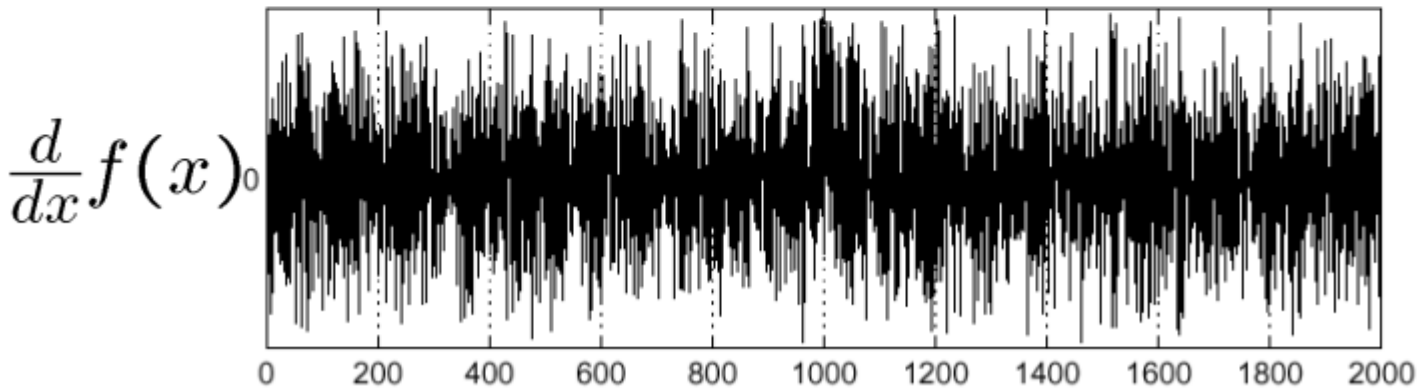


На реальных снимках присутствует шум



Производная зашумленной функции интенсивности



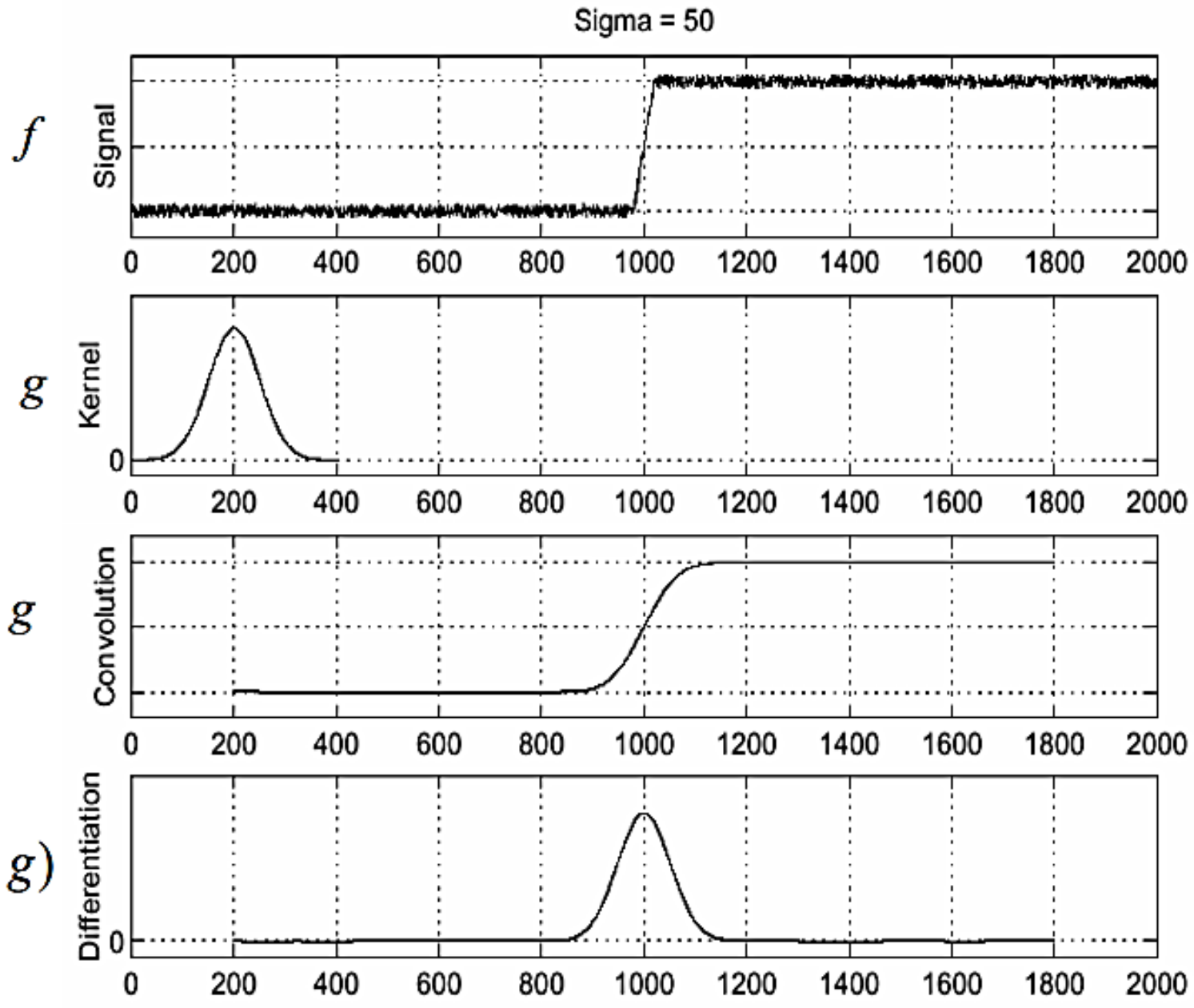


И где же здесь углы?

Что делать?

Решение: сглаживание изображения перед нахождением производной

Фильтр Гауса g



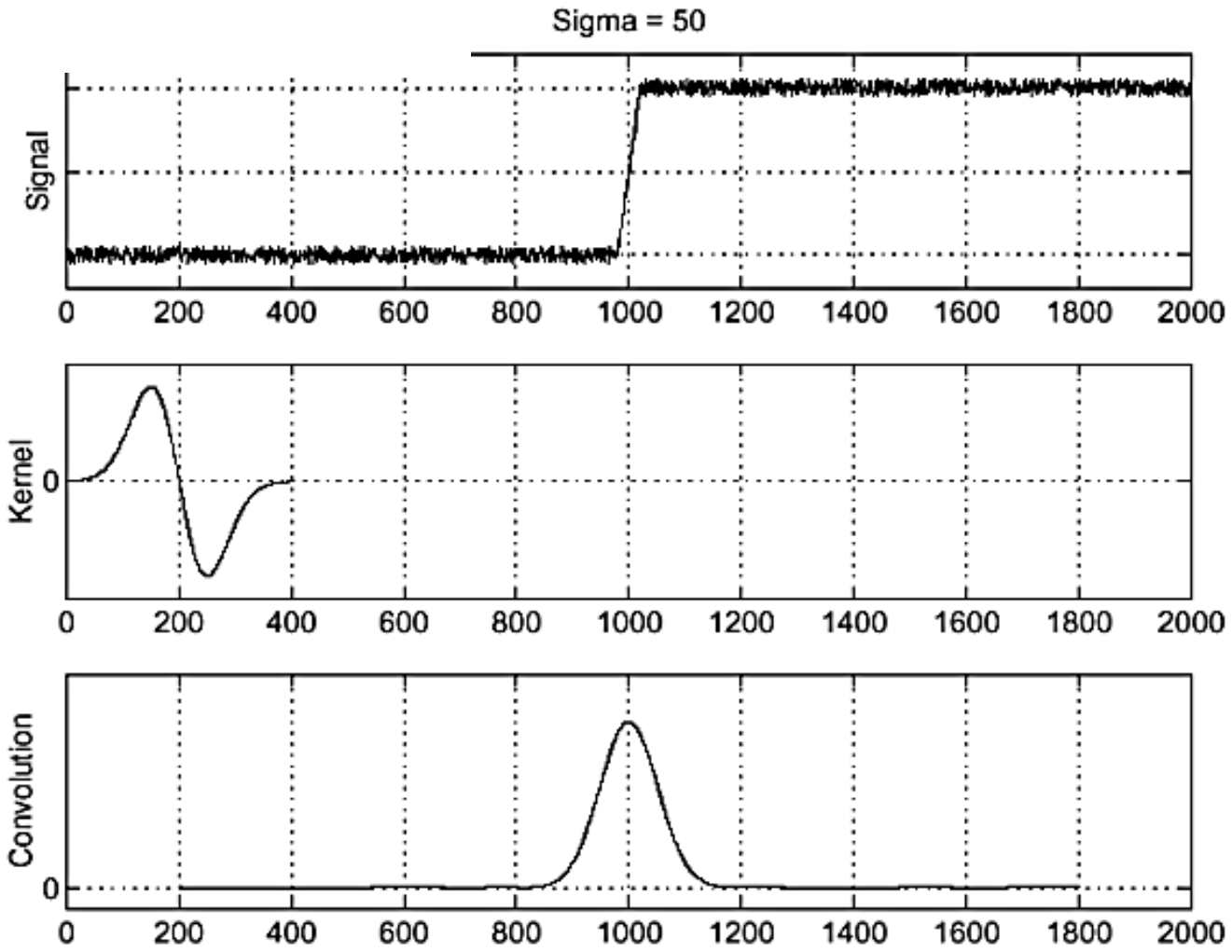
- Differentiation is convolution, and convolution is associative: $\frac{d}{dx}(f * g) = f * \frac{d}{dx} g$

- This saves us one operation:

f

$\frac{d}{dx} g$

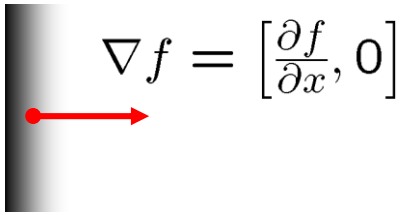
$f * \frac{d}{dx} g$

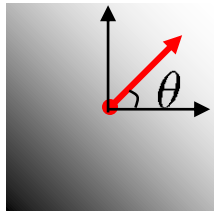


В рассматриваемом далее детекторе краёв Canny как раз используется фильтр на основе первой производной от гауссианы

Градиент изображения

- Градиент изображения: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$





Градиент направлен в сторону наибольшего изменения интенсивности

Направления градиента задается как: $\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$

Сила края задается величиной (нормой) градиента:

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Source: Steve Seitz

Дифференцирование и свёртка

•Для функции 2х переменных, f(x,y):

$$\frac{\partial f}{\partial x} = \lim_{\varepsilon \to 0} \left(\frac{f(x+\varepsilon, y)}{\varepsilon} - \frac{f(x,y)}{\varepsilon} \right)$$

• Разностная производная:

$$\frac{\partial f}{\partial x} \approx \frac{f(x_{n+1}, y) - f(x_n, y)}{\Delta x}$$

•Линейная и инвариантная к переносу, поэтому м.б. Результатом свертки

• Свёртка!

-1	1

Source: D. Forsyth, D. Lowe

Вычисление градиента

Семейство методов основано на приближенном вычисление градиента, анализе его направления и абсолютной величины.

Свертка по функциям:

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Робертса

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Превитт

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Собеля

Фильтры Собеля позволяют получить более устойчивые к шуму градиенты

Примеры карты силы краев

Примеры:

Робертса

Превитт

Собеля

Выделение краев

Подробнее оператор Собеля
https://robocraft.ru/computervision/460

По оси Y

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Собеля

По оси X

Результат сложения

В *OpenCV* оператор Собеля реализуется функцией ***Sobel***():

```
void Sobel(
    InputArray src,         // входное изображение
    OutputArray dst,        // выходное изображение
    int ddepth,             // глубина выходного изображения
    int xorder,             // порядок частной производной по x
    int yorder,             // порядок частной производной по y
    Size ksize = 3,         // размер ядра
    double scale = 1,       // коэффициент масштабирования
    double delta = 0,                   // смещение
    int borderType = BORDER_DEFAULT   // экстраполяция границы
    );
```

Вторая производная может также использоваться для обнаружения краев. Для этих целей служит **оператор Лапласа**, который позволяет вычислить так называемый лапласиан изображения суммирование производных второго порядка

$$\text{Laplase}(f) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Оператор Лапласа реализован в *OpenCV* функцией *Laplacian*().

Поскольку оператор Лапласа определяется в терминах вторых производных, можно предположить, что дискретная реализация работает примерно так же, как производная Собеля второго порядка. В реализации оператора Лапласа в *OpenCV* напрямую используются операторы Собеля:

```
void Laplacian(
    InputArray src,         // входное изображение
    OutputArray dst,        // выходное изображение
    int ddepth,             //глубина выходного изображения
    int ksize = 3,          // размер ядра
    double scale = 1,       // масштабный коэффициент
    double delta = 0,       // смещение
    int borderType = BORDER_DEFAULT   // экстраполяция границы
    );
```

Выделение краев

Исходное изображение

Карта силы краев

Что можно здесь улучшить?

- Сделать края тоньше
- Добавить информацию о связности

Разработка детектора краев

- Критерии качества детектора:
  - **Надежность:** оптимальный детектор должен редко ошибаться (ложные края и пропущенные края)
  - **Точная локализация:** найденный край должен быть как можно ближе к истинному краю
  - **Единственный отклик на одну границу:** детектор должен выдавать одну точку для одной точки истинного края, т.е. локальных максимум вокруг края должно быть как можно меньше
  - **Связанность:** хотим знать, какие пиксели принадлежат одной линии края

В 1986 году Джоном Кэнни (англ. John F. Canny) был разработан детектор границ Кэнни, который использует многоступенчатый алгоритм для обнаружения широкого спектра границ в изображениях.
Детектор границ Канни до сих пор является одним из лучших детекторов. Кроме особенных частных случаев трудно найти детектор, который бы работал существенно лучше, чем детектор Канни.

Source: L. Fei-Fei

Детектор Canny (1986)

Подробнее

Алгоритм состоит из пяти отдельных шагов:

- **Сглаживание**. Размытие изображения для удаления шума.
- **Поиск градиентов**. Границы отмечаются там, где градиент изображения приобретает максимальное значение.
- **Подавление не-максимумов**. Только локальные максимумы отмечаются как границы.
- **Двойная пороговая фильтрация**. Потенциальные границы определяются порогами.
- **Трассировка области неоднозначности**. Итоговые границы определяются путём подавления всех краёв, несвязанных с определенными (сильными) границами.

Перед применением детектора, преобразуем изображение в оттенки серого, чтобы уменьшить вычислительные затраты.

Отличие детектора Кэнни от рассмотренных ранее методов

- Утоньшение полос в несколько пикселей до одного пикселя
- Связывание краев и обрезание по порогу

Рассмотренные ранее **градиентные методы выделения границ** можно использовать для нахождения пикселей, расположенных на границе между различными участками изображения, но они ничего не говорят о границах как самостоятельных сущностях.

Контур – это список точек, представляющий кривую в изображении. Существует много способов представить кривую, все зависит от обстоятельств. В OpenCV контуры представляются STL-вектором vector<>, каждый элемент которого содержит информацию об одной точке на кривой. Например, в цепочке Фримена каждая точка представлена «шагом» в направлении от предыдущей точки.

```
1-1-1-2-3-3-4-4-4-4-5-5-6-6-6-7-0-0-0-0
```

Таким образом, зная описание контура, можно производить различные вычисления: например, найти длину контура, площадь фигуры и т.д.

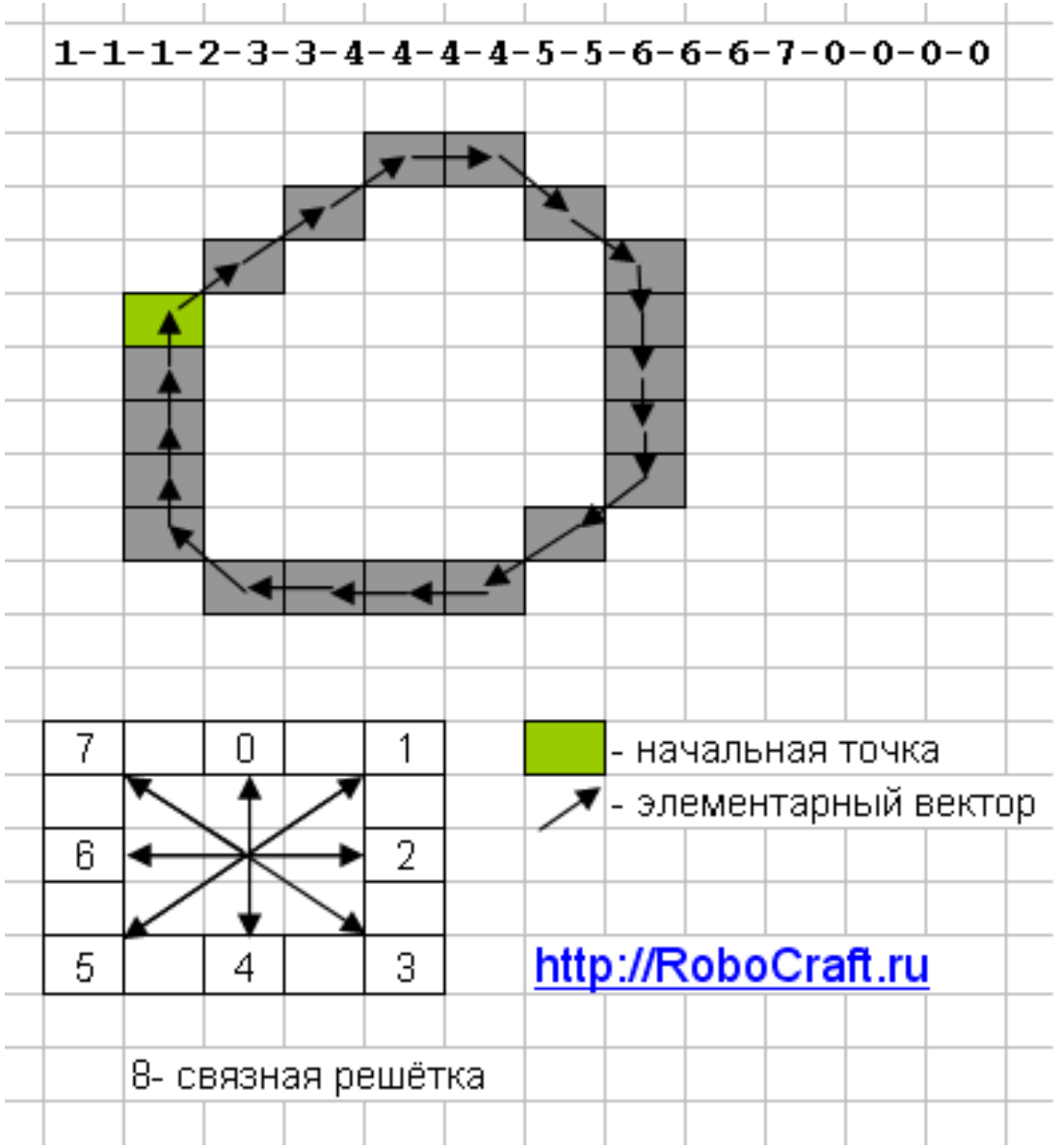
Контурный анализ является совокупностью методов выделения, описания и анализа контуров с целью получения информации об изображении.

Цепной код Фримена (Фридмана)

(Freeman Chain Code)

Детекторы границ позволят обнаружить границы, которые затем желательно записать в виде вектора для последующей работы. Один из вариантов:

Цепные коды применяются для представления границы в виде последовательности отрезков прямых линий определённой длины и направления. В основе этого представления лежит 4- или 8- связная решётка. Длина каждого отрезка определяется разрешением решётки, а направления задаются выбранным кодом.



http://robocraft.ru/blog/computervision/640.html
https://robotclass.ru/tutorials/opencv-python-find-contours/

Нахождение контуров и операции с ними

Для поиска контуров используется функция **FindContours()**:

```
findContours( кадр, режим_группировки, метод_аппроксимации [, контуры[, иерархия[, сдвиг]]])
```

кадр —подготовленное для анализа изображение. Это должно быть 8-битное изображение. Поиск контуров использует для работы монохромное изображение, так что все пиксели картинки с ненулевым цветом будут интерпретироваться как 1, а все нулевые останутся нулями.

режим_группировки может принимать следующие значения:

```
CV_RETR_EXTERNAL 0 // найти только крайние внешние контуры
CV_RETR_LIST     1 // найти все контуры и разместить их списком
CV_RETR_CCOMP    2 // найти все контуры и разместить их в виде 2-уровневой иерархии
CV_RETR_TREE     3 // найти все контуры и разместить их в иерархии вложенных контуров
```

контуры — список всех найденных контуров, представленных в виде векторов;

иерархия — информация о топологии контуров. Каждый элемент иерархии представляет собой сборку из четырех индексов, которая соответствует контуру[i]:

иерархия[i][0] — индекс следующего контура на текущем слое;
иерархия[i][1] — индекс предыдущего контура на текущем слое:
иерархия[i][2] — индекс первого контура на вложенном слое;
иерархия[i][3] — индекс родительского контура.

сдвиг — величина смещения точек контура.

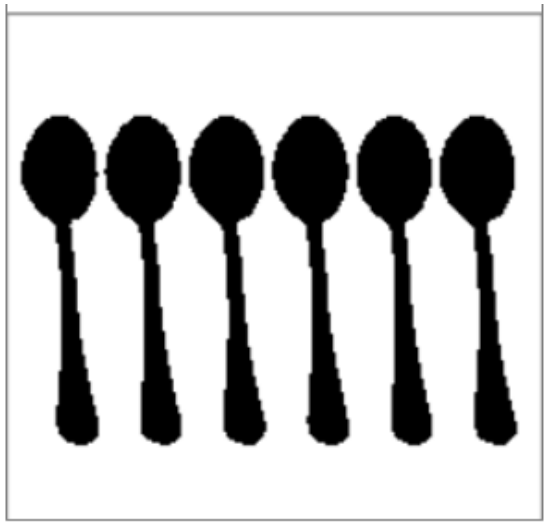
Иерархия отношений между контурами **hierarchy**, для каждого контура contours[i] в hierarchy[i][0], hierarchy[i][1], hierarchy[i][2], hierarchy[i][3] хранятся индексы следующего и предыдущего контуров того же уровня иерархии, индексы первого дочернего и родительского контуров соответственно.

Контуры, найденные на одноканальном изображении

Задача: подсчитать количество предметов

?

Здесь функция вернула множество внутренних контуров и если задача состоит в подсчете предметов по контурам, то эти внутренние контуры приведут к неверному результат. Эту задачу можно решить различными способами. Например, можно в параметре ***method*** указать ***RETR_EXTERNAL*** (возвращает только внешние контуры) или перед поиском контуров преобразовать изображение в бинарное, чтобы исключить влияние теней или текстуры предметов.



Контуры, найденные на бинарном изображении

Если в рассмотренном выше коде в качестве входного изображения функции ***findContours***() использовать бинарное изображение, то результат будет гораздо лучше

Нахождение контуров и операции с ними

Последовательность действий при распознавании объектов методом контурного анализа:

1. предварительная обработка изображения (сглаживание, фильтрация помех, увеличение контраста);
2. бинаризация изображения;
3. операции математической морфологии;
4. выделение контуров объектов;
5. первичная фильтрация контуров (по периметру, площади и т.п.);
6. анализ контуров в соответствии с поставленной задачей.

Свойства контуров

cv.contourArea(contour[, oriented]) -> retval

oriented = `false`

Calculates a contour area.

cv.arcLength(curve, closed) -> retval

closed = 0 — кривая полагается открытой
closed > 0 — кривая полагается закрытой

Calculates a contour perimeter or a curve length.

Example:

```
vector<Point> contour;
contour.push_back(Point2f(0, 0));
contour.push_back(Point2f(10, 0));
contour.push_back(Point2f(10, 10));
contour.push_back(Point2f(5, 4));

double area0 = contourArea(contour);
vector<Point> approx;
approxPolyDP(contour, approx, 5, true);
double area1 = contourArea(approx);

cout << "area0 =" << area0 << endl <<
        "area1 =" << area1 << endl <<
        "approx poly vertices" << approx.size() << endl;
```

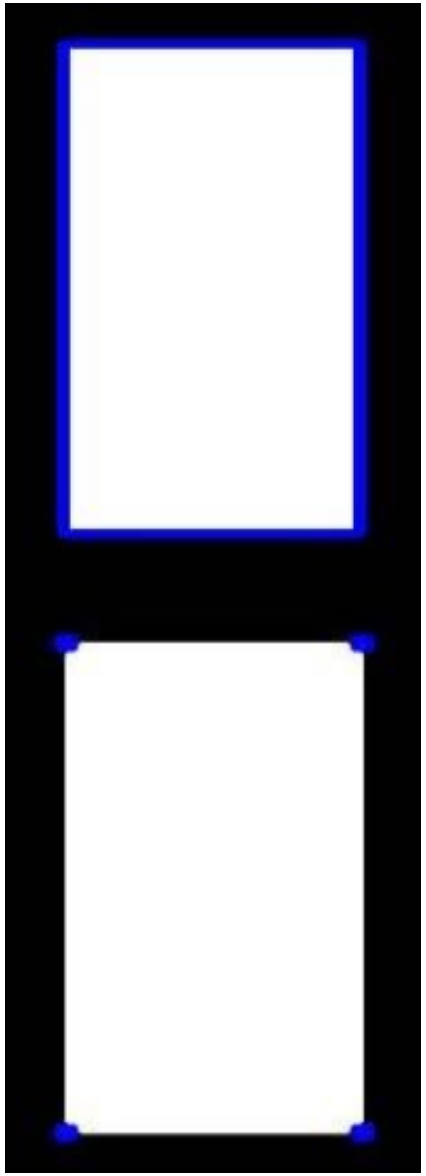
Метод контурной аппроксимации

Это третий аргумент `cv2.findContours` функции. Что оно обозначает на самом деле?

Выше мы сказали, что контуры — это границы формы с одинаковой интенсивностью. Он хранит (x, y) координаты границы формы. Но хранит ли он все координаты? Это определяется этим методом аппроксимации контура.

Если вы пройдете `cv2.CHAIN_APPROX_NONE`, все граничные точки будут сохранены. Но на самом деле нужны ли нам все точки? Например, вы нашли контур прямой линии. Вам нужны все точки на линии, чтобы представить эту линию? Нет, нам нужны только две конечные точки этой линии. Вот что `cv2.CHAIN_APPROX_SIMPLE` делает. Он убирает все лишние точки и сжимает контур, тем самым экономя память.

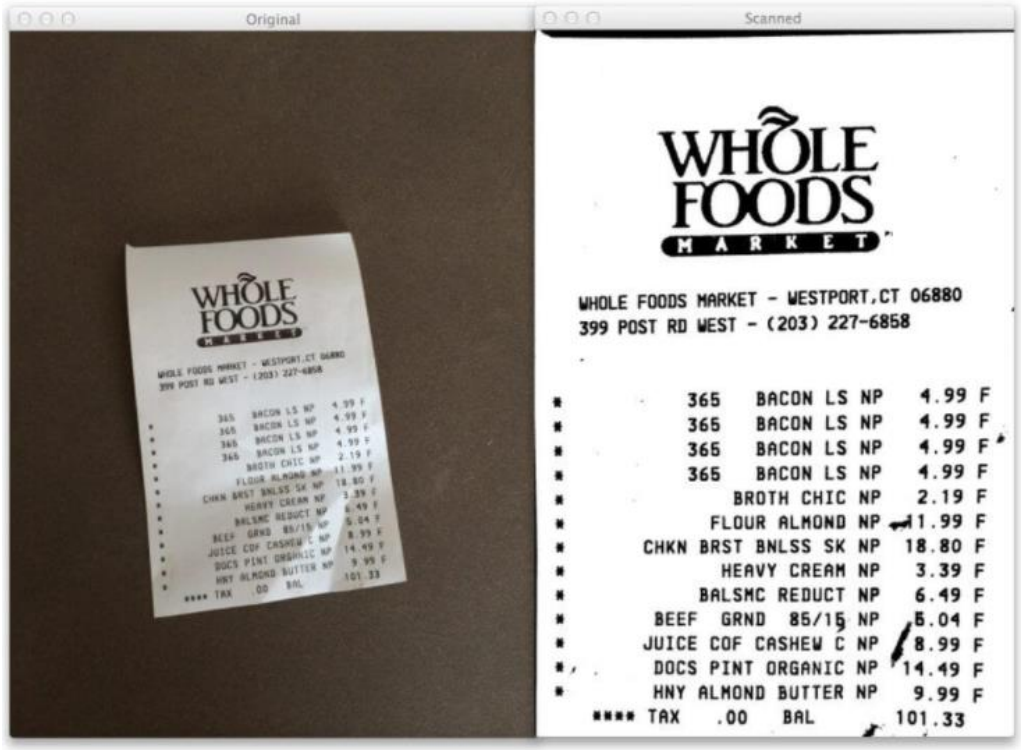
Ниже изображение прямоугольника демонстрирует эту технику. Просто нарисуйте круг по всем координатам в массиве контуров (нарисованы синим цветом). На первом изображении показаны баллы, которые я получил `cv2.CHAIN_APPROX_NONE` (734 балла), а на втором изображении показан тот, в котором я получил `cv2.CHAIN_APPROX_SIMPLE` (всего 4 балла). Посмотрите, сколько памяти это экономит!!!



Оригинал статьи
https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contours_begin/py_contours_begin.html#contours-getting-started

https://www.pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/?_ga=2.81015831.1637974644.1638339468-1528956013.1637982400

https://www.pyimagesearch.com/2014/09/01/build-kick-ass-mobile-document-scanner-just-5-minutes/



Моменты

Моменты изображения помогают рассчитать некоторые функции, такие как центр масс объекта, площадь объекта и т. Д.

Функция **cv2.moments()** предоставляет словарь всех рассчитанных значений моментов:

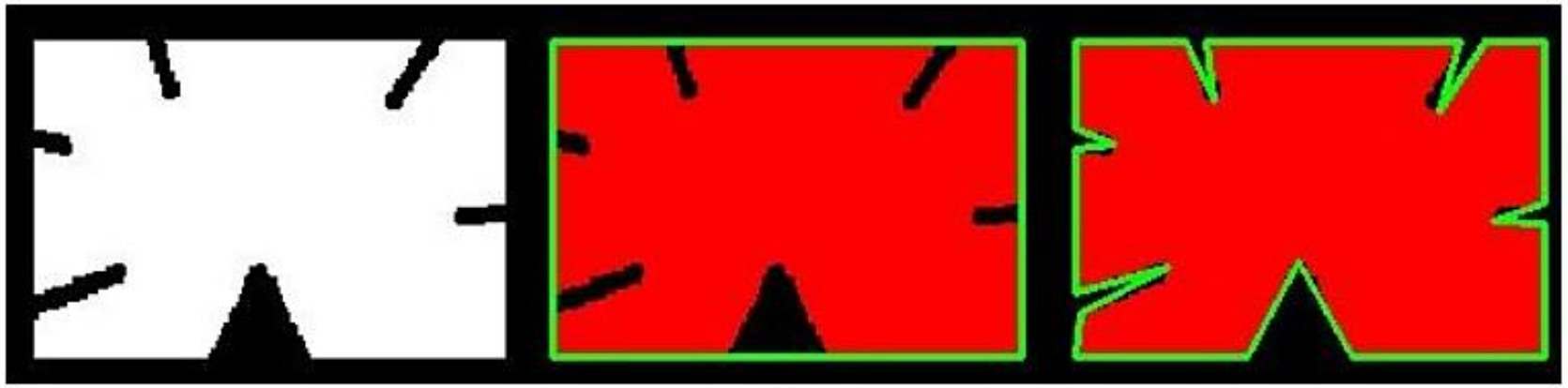
```
import cv2
import numpy as np

img = cv2.imread('star.jpg',0)
ret,thresh = cv2.threshold(img,127,255,0)
contours,hierarchy = cv2.findContours(thresh, 1, 2)

cnt = contours[0]
M = cv2.moments(cnt)
print M
```

Контурная аппроксимация приближает форму контура к другой форме с меньшим количеством вершин в зависимости от указанной нами точности **epsilon**.

```
epsilon = 0.1*cv2.arcLength(cnt,True)
approx = cv2.approxPolyDP(cnt,epsilon,True)
```

Below, in second image, green line shows the approximated curve for `epsilon = 10% of arc length`. Third image shows the same for `epsilon = 1% of the arc length`. Third argument specifies whether curve is closed or not.

epsilon=10% **epsilon=1%**



https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contour_features/py_contour_features.html

Выпуклая оболочка будет похожа на контурную аппроксимацию, но это не так (в некоторых случаях оба метода могут давать одинаковые результаты). Здесь функция **cv2.convexHull()** проверяет кривую на дефекты выпуклости и исправляет ее. Вообще говоря, выпуклые кривые — это кривые, которые всегда выпуклые или, по крайней мере, плоские. А если он вогнут внутрь, то это называется дефектом выпуклости.

Например, посмотрите на изображение руки ниже. Красная линия показывает выпуклый корпус руки. Двусторонними стрелками показаны дефекты выпуклости, которые являются локальными максимальными отклонениями корпуса от обводов.

Ограничивающий прямоугольник

Существует два типа ограничивающих прямоугольников.

- **Прямой ограничивающий прямоугольник** *(зеленый на рисунке)*

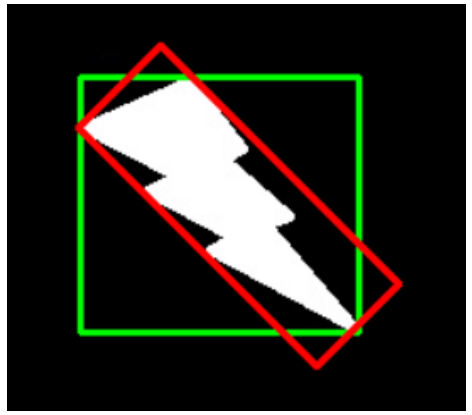
Это прямой прямоугольник, он не учитывает вращение объекта. Таким образом, площадь ограничивающего прямоугольника не будет минимальной. Его находит функция **cv2.boundingRect()** .

- **Повернутый прямоугольник** *(красный на рисунке)*

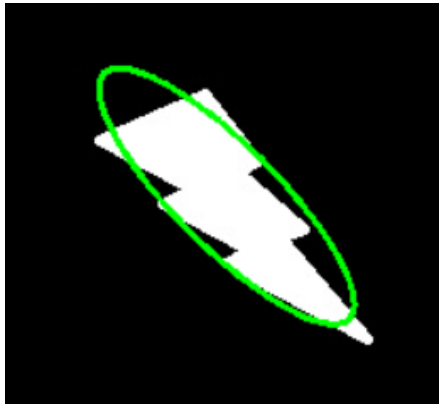
Здесь ограничивающий прямоугольник нарисован с минимальной площадью, поэтому он также учитывает вращение. Используется функция **cv2.minAreaRect()** .

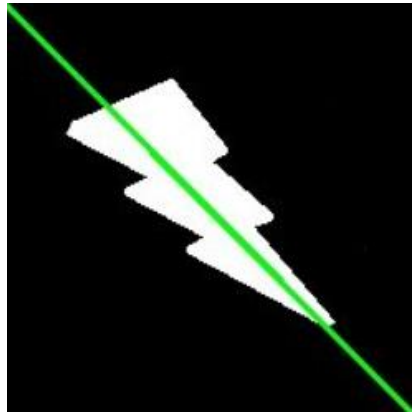
Минимальный ограничивающий круг

Находим описанную окружность объекта с помощью функции **cv2.minEnclosingCircle()** . Это круг, который полностью покрывает объект с минимальной площадью.

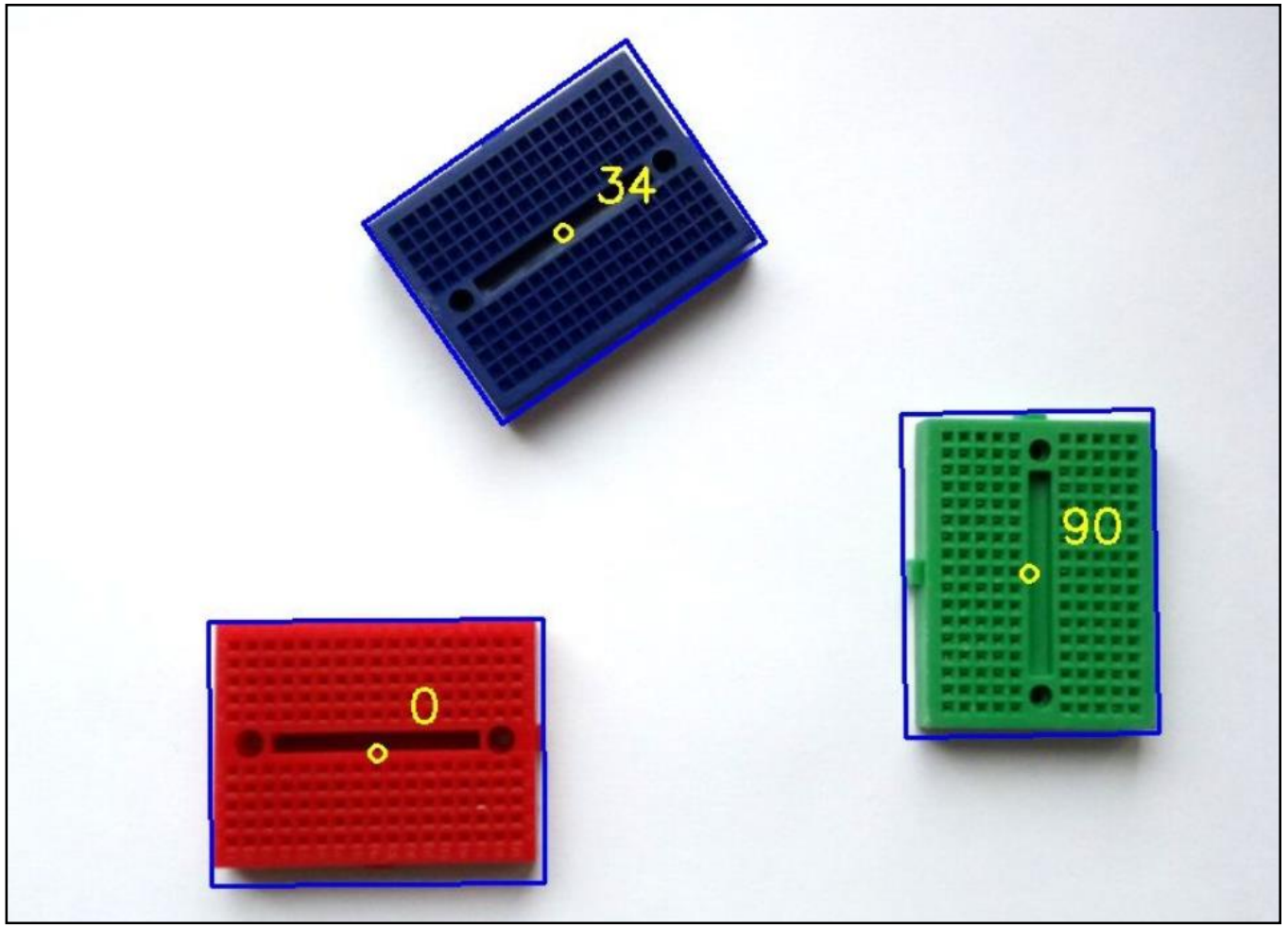


ещё





пример применения контурного анализа для поиска прямоугольников определения их наклона относительно горизонта (для роботов)



https://robotclass.ru/tutorials/opencv-detect-rectangle-angle/

minAreaRect() Функция вычисляет и возвращает ограничивающий прямоугольник минимальной площади (возможно, повернутый) для указанного набора точек.

minEnclosingCircle()
Находит окружность минимальной площади, охватывающую набор 2D-точек.

Экстремальные точки — это самая верхняя, самая нижняя, самая правая и самая левая точки объекта.

Контурный анализ — это метод описания, хранения, распознавания, сравнения и поиска графических объектов по их контурам.

Переход к рассмотрению только контуров объектов позволяет уйти от пространства изображения – к пространству контуров, что **существенно снижает сложность алгоритмов и вычислений**.

Ограничения на область применения контурного анализа:

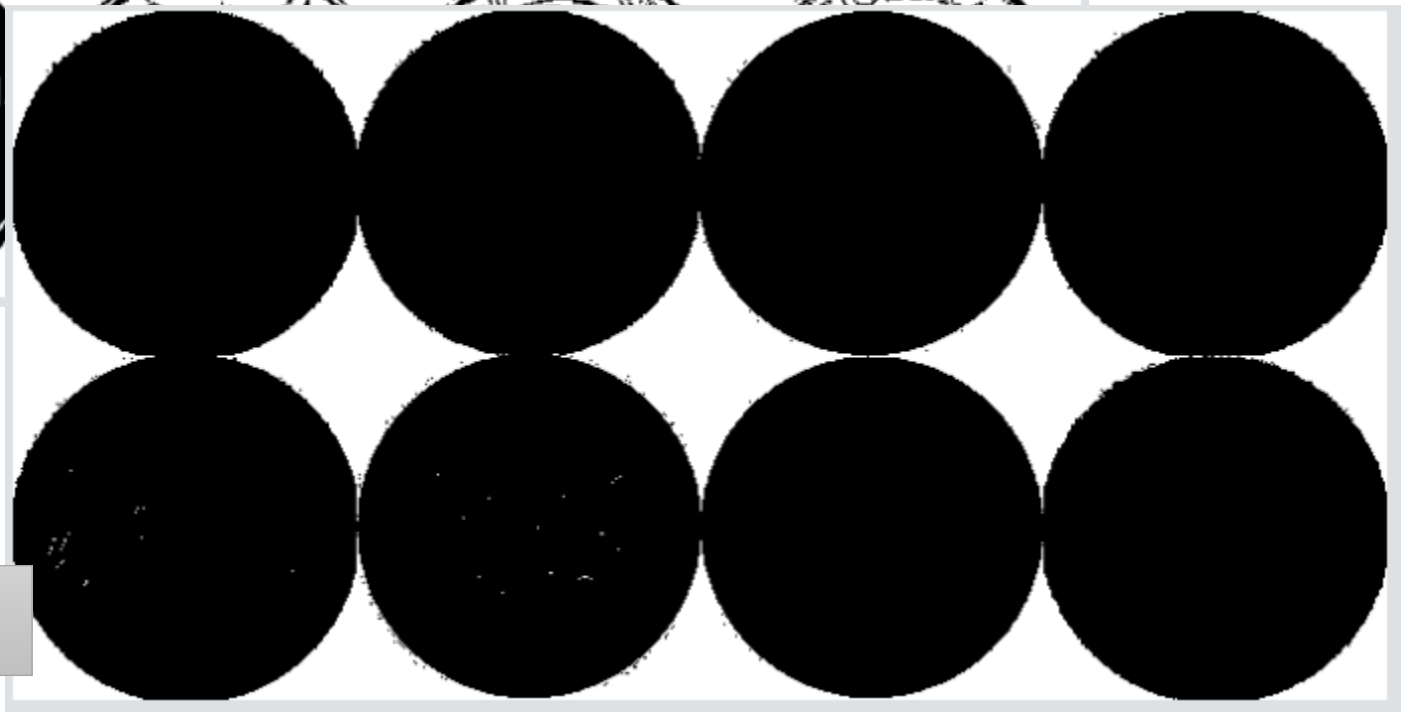
- ✓ объект может не иметь чёткой границы, или может быть зашумлён помехами, что приводит к невозможности выделения контура;
- ✓ перекрытие объектов или их группировка приводит к тому, что контур выделяется неправильно и не соответствует границе объекта.

Вывод:

- контурный анализ простой и быстрый;
- контурный анализ хорошо работает при чётко выраженном объекте на контрастном фоне и отсутствии помех;
- контурный анализ имеет довольно **слабую устойчивость к помехам**.

Задача: подсчитать количество

Threshold = 170



Threshold = 250

Что можно сделать еще?

Проблема:

При автоматизированном анализе цифровых изображений часто возникает проблема идентификации простых фигур, таких как прямые, круги или эллипсы. из-за зашумлённости изображения, либо из-за несовершенства алгоритма обнаружения границ, могут появиться «потерянные» точки на кривой, также как и небольшие отклонения от идеальной формы прямой, круга или эллипса.
По этим причинам **часто довольно сложно приписать найденные границы соответствующим прямым, кругам и эллипсам в изображении.**

Решение - применение преобразования Хафа

Преобразование Хафа

Преобразование Хафа (Hough Transform) — это метод для поиска линий, кругов и других простых форм на изображении.

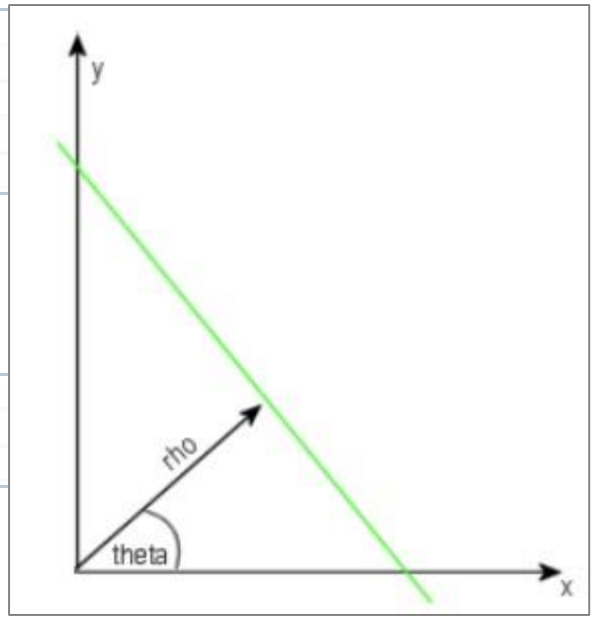
Преобразование Хафа основывается на представлении искомого объекта в виде **параметрического уравнения**. Параметры этого уравнения представляют фазовое пространство (пространство Хафа).

Берётся двоичное изображение (например, результат работы детектора границ Кенни). Перебираются все точки границ и делается предположение, что точка принадлежит линии искомого объекта — т.о. для каждой точки изображения рассчитывается нужное уравнение и получаются необходимые параметры, которые сохраняются в пространстве Хафа.
Финальным шагом является обход пространства Хафа и выбор максимальных значений, за которые «проголосовало» больше всего пикселей картинки, что и даёт нам параметры для уравнений искомого объекта.

В основе теории преобразования Хафа лежит утверждение, что любая точка двоичного изображения может быть частью некоторого набора возможных линий.

формула линии:

```
y = a*x+b                  // в декартовых координатак
p = x*cos(f)+y*sin(f)      // в полярных координатах
```



Прямую на плоскости можно представить:

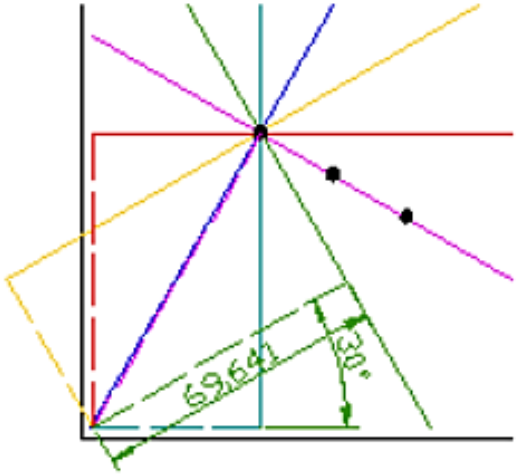
```
x*cos(f) + y*sin(f) = R
```

, где

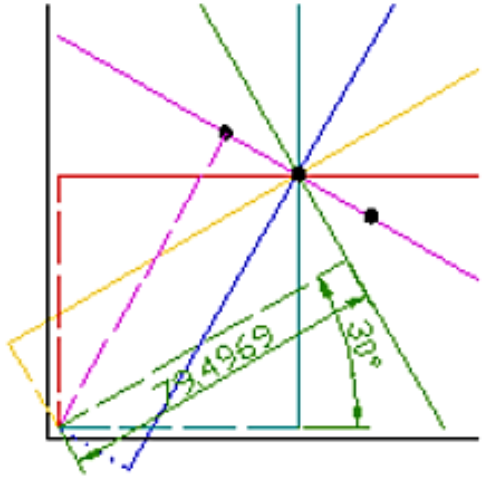
R – длина перпендикуляра опущенного на прямую из начала координат,

f — угол между перпендикуляром к прямой и осью OX.

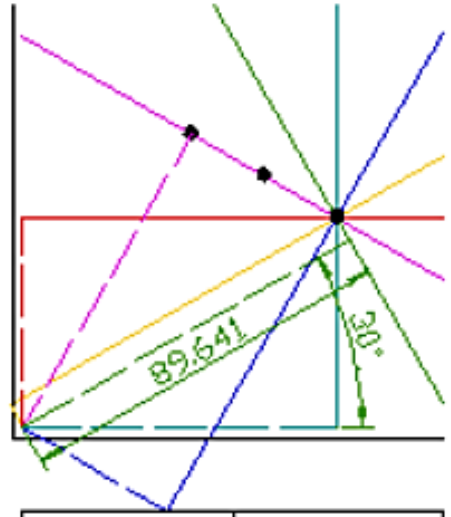
Рассмотрим исходное тестовое изображение из трех черных точек. Проверим, расположены ли точки на прямой линии.



Angle	Dist.
0	40
30	69.6
60	81.2
90	70
120	40.6
150	0.4



Angle	Dist.
0	57.1
30	79.5
60	80.5
90	60
120	23.4
150	−19.5



Angle	Dist.
0	74.6
30	89.6
60	80.6
90	50
120	6.0
150	−39.6

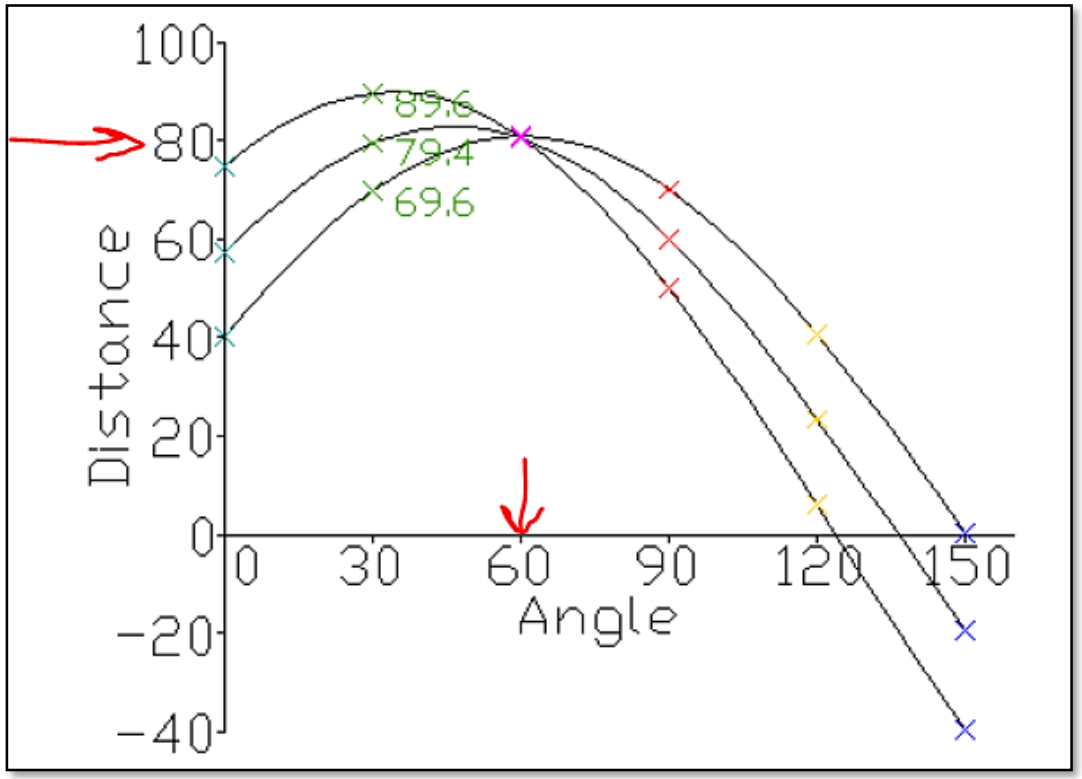
- Через каждую точку проведено (для наглядности) только по шесть прямых, имеющих разный угол.
- К каждой прямой из начала координат построен перпендикуляр.
- Для всех прямых длина соответствующего перпендикуляра и его угол с осью абсцисс сведены в таблицу.
- Данные таблицы являются результатом преобразования Хафа и могут служить основой для графического представления в "пространстве Хафа".

Прямую на плоскости можно представить:

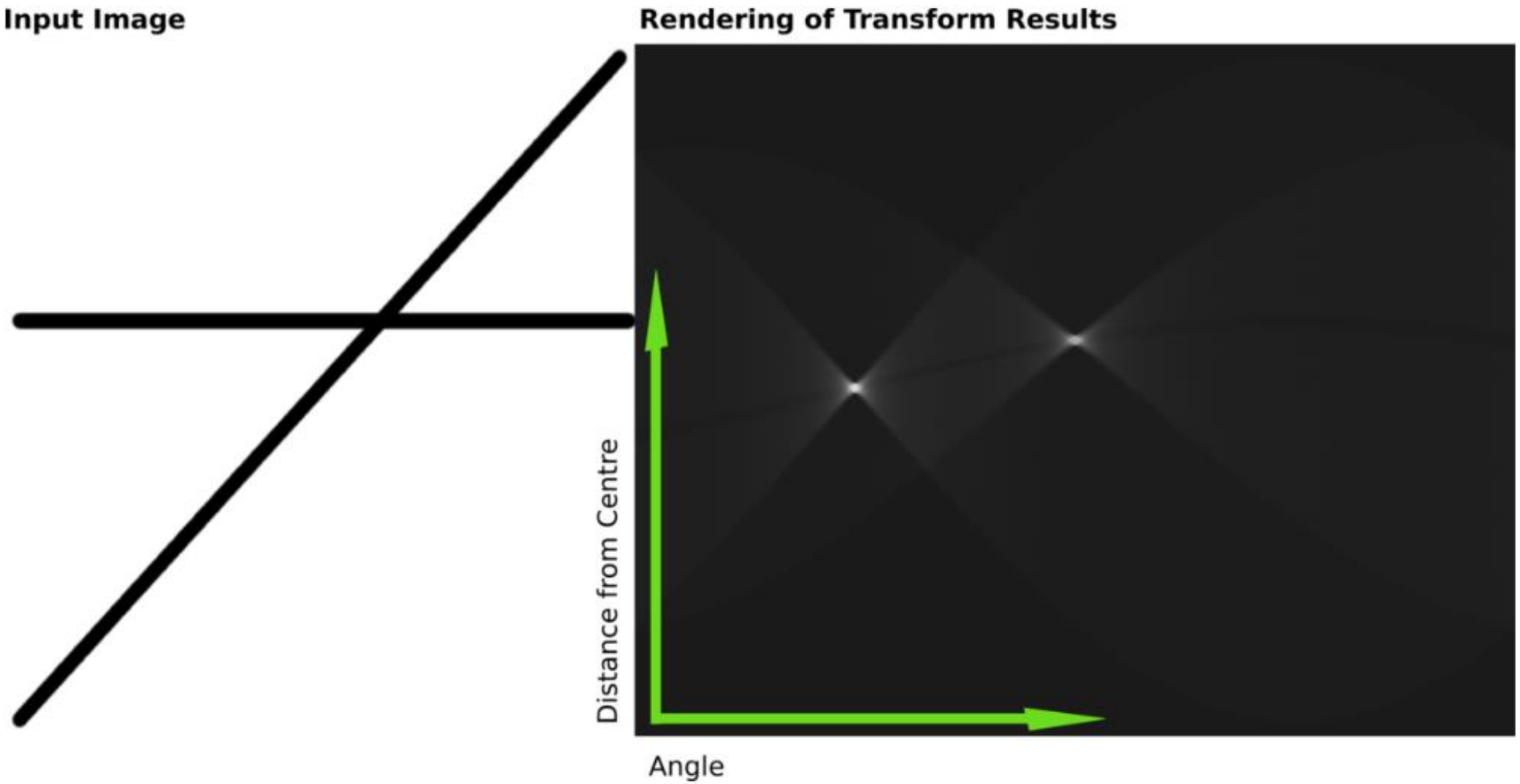
```
x*cos(f) + y*sin(f) = R
```

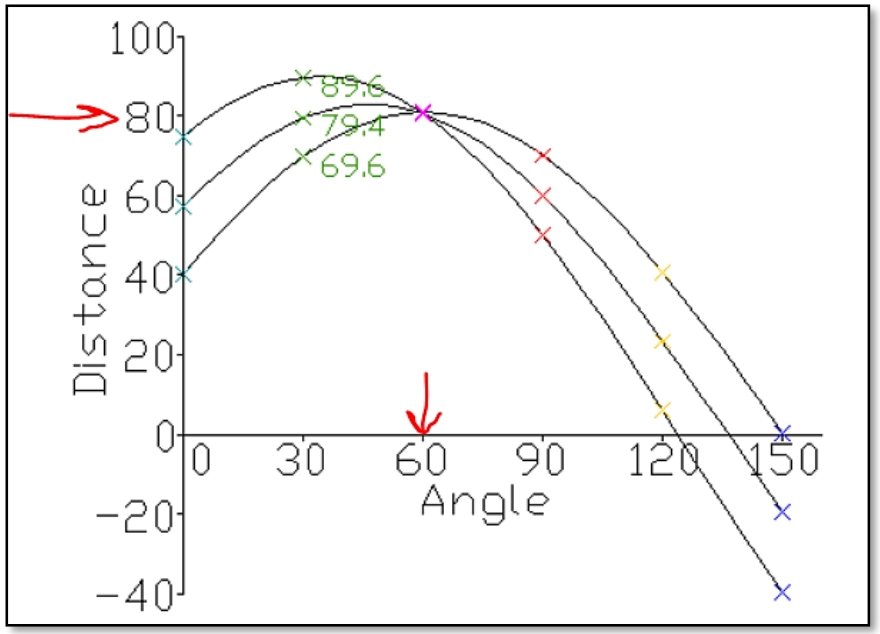
Через каждую точку **(x,y)** изображения можно провести несколько прямых с разными R и f, то есть каждой точке (x, y) изображения соответствует набор точек в фазовом пространстве **(R, f)**, образующий синусоиду.

Координаты точки пересечения синусоид определяют параметры прямой, общей для проверяемых точек на исходном изображении.

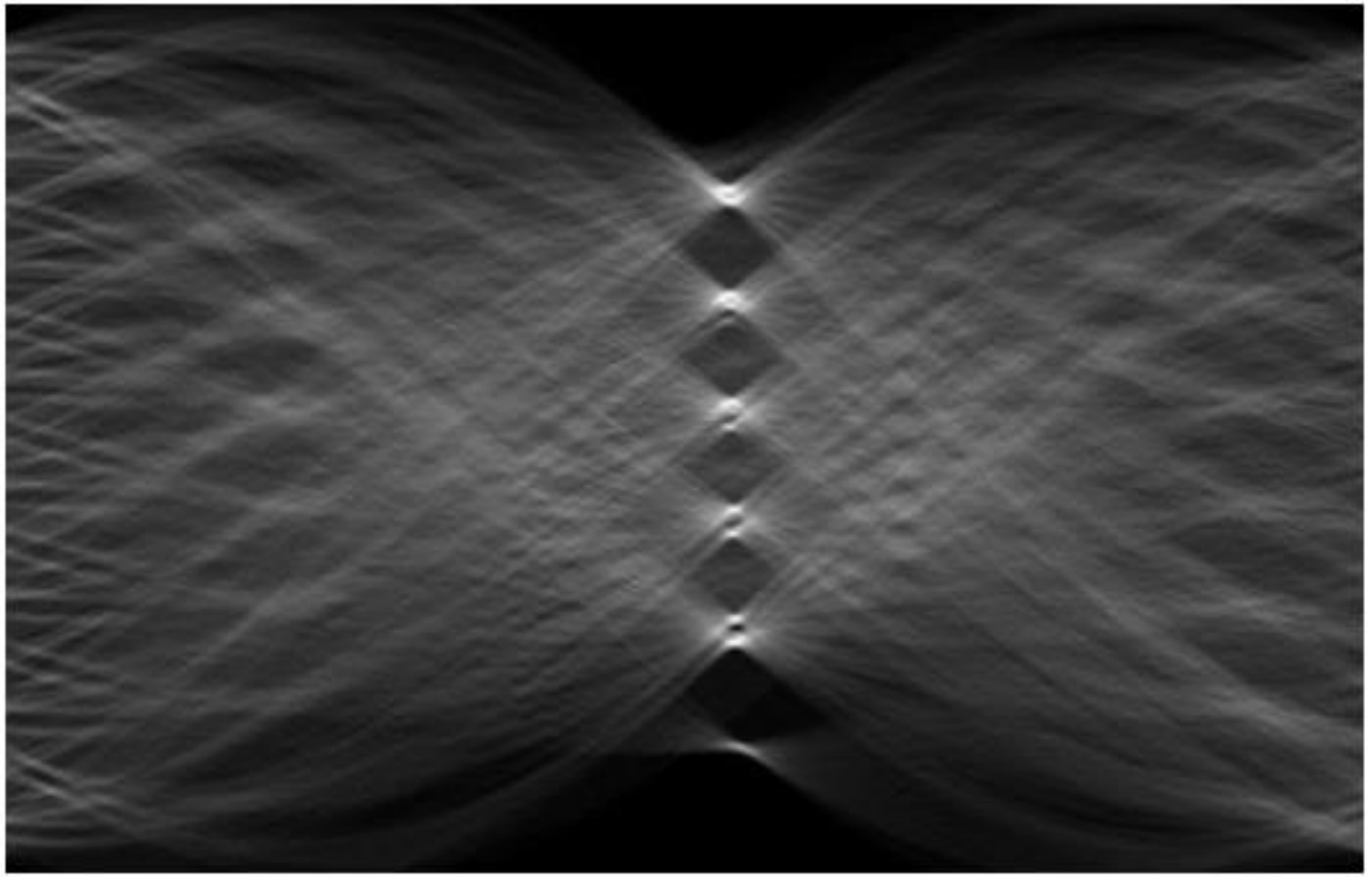


Отображение в пространстве Хафа двух прямых

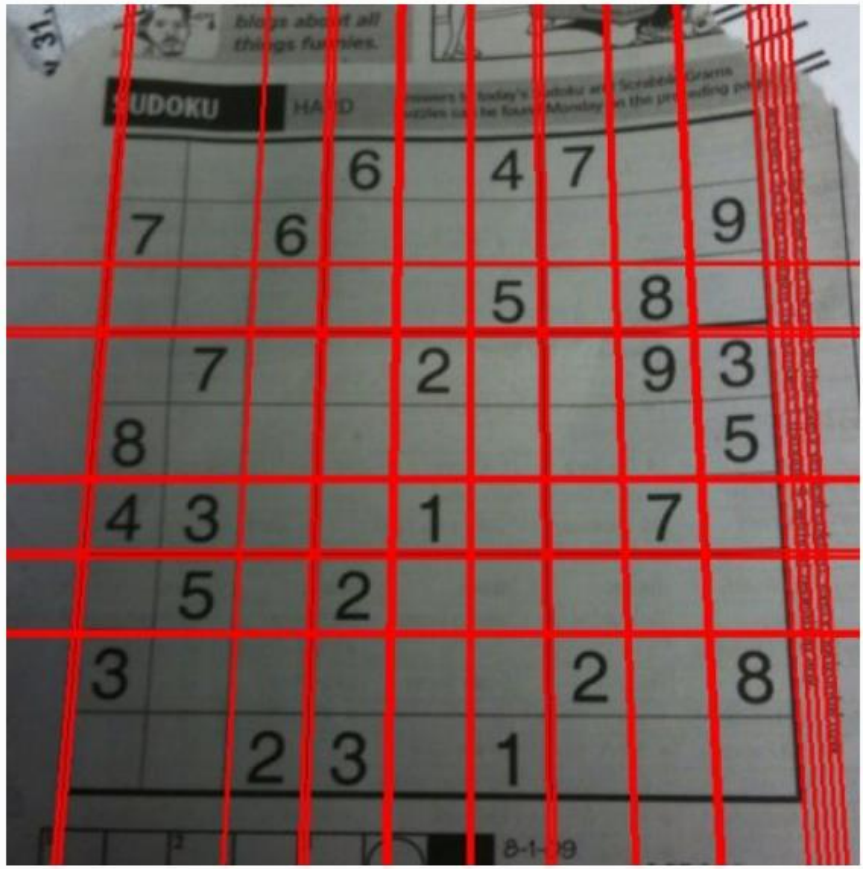




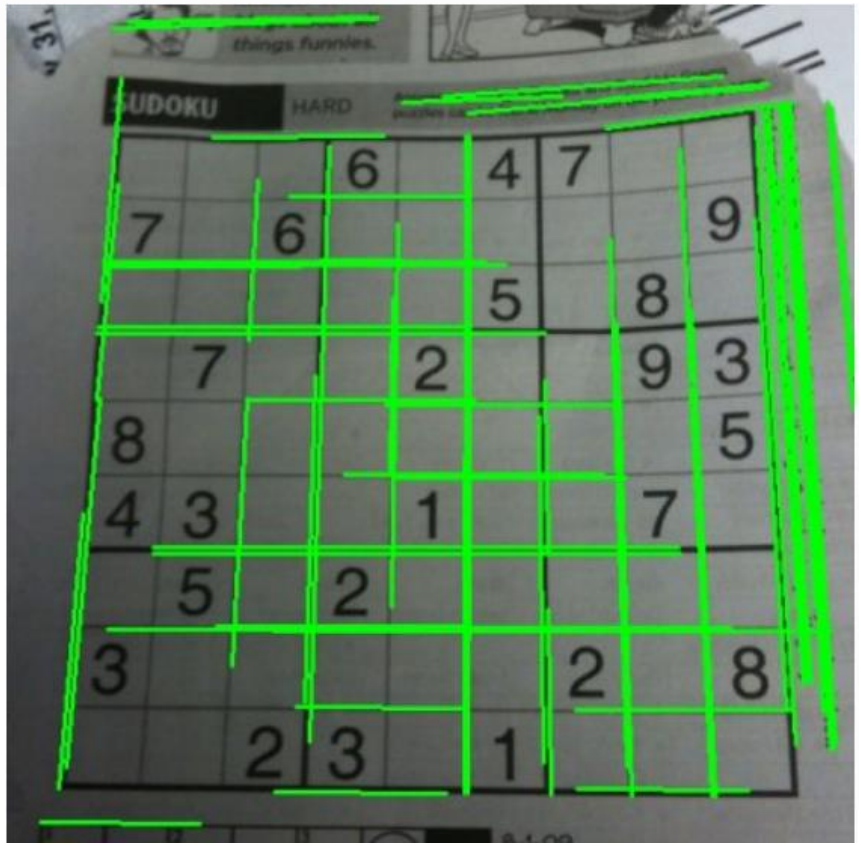
Что здесь изображено?



```
img = cv2.imread('dave.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,50,150,apertureSize = 3)

lines = cv2.HoughLines(edges,1,np.pi/180,200)
for rho,theta in lines[0]:
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))

    cv2.line(img,(x1,y1),(x2,y2),(0,0,255),2)

cv2.imwrite('houghlines3.jpg',img)
```

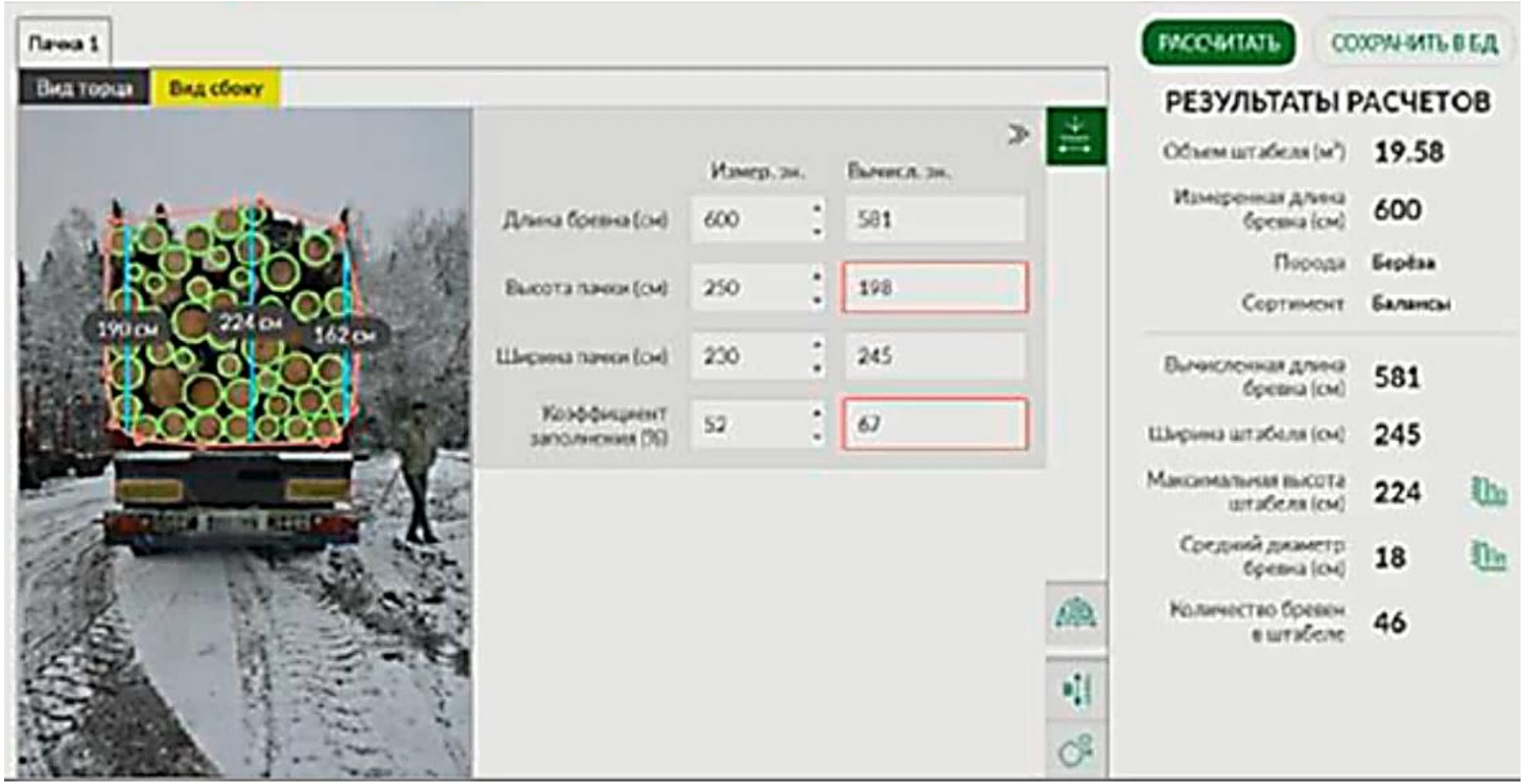


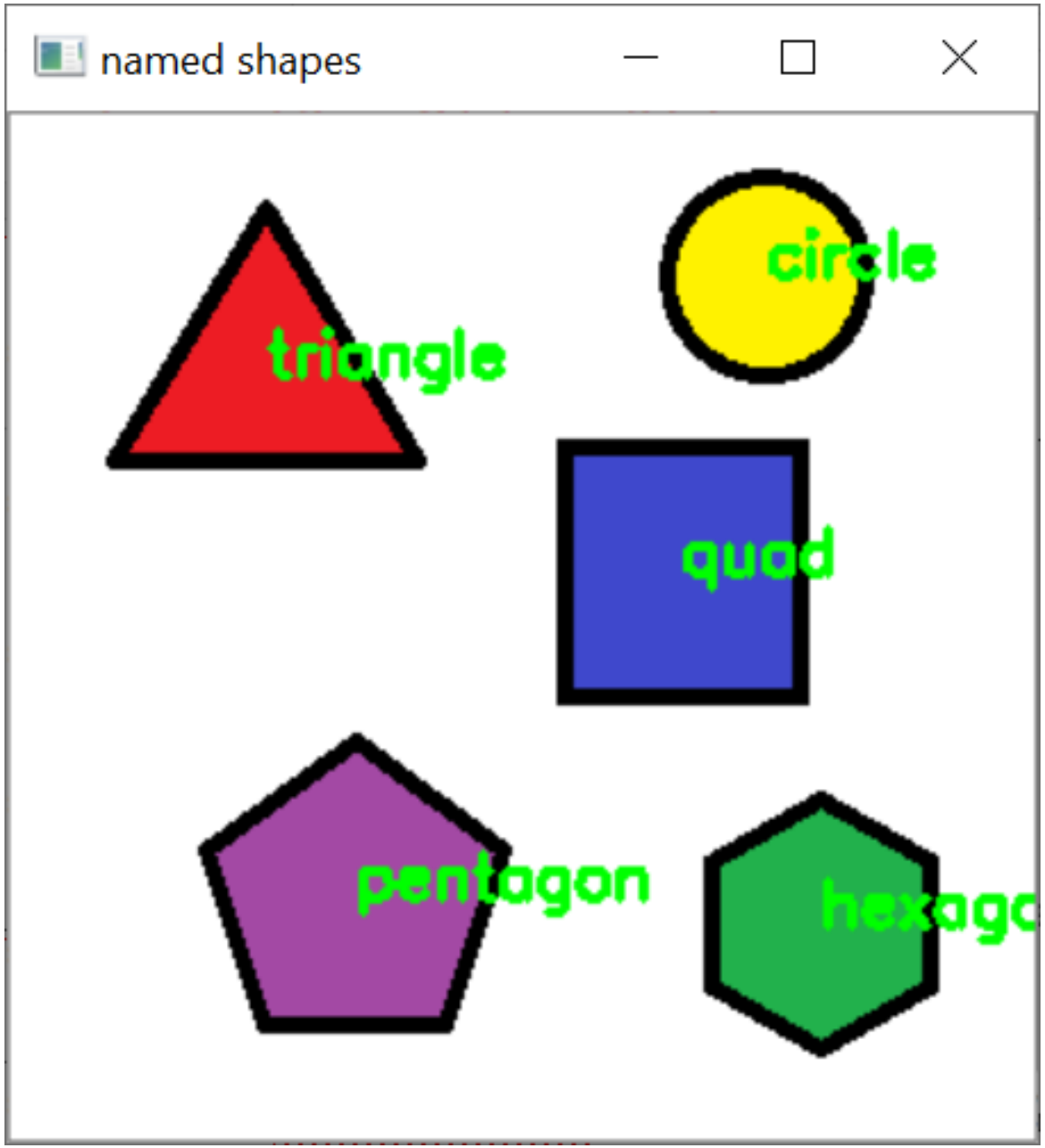
```
img = cv2.imread('dave.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,50,150,apertureSize = 3)
minLineLength = 100
maxLineGap = 10
lines = cv2.HoughLinesP(edges,1,np.pi/180,100,minLineLength,maxLineGap)
for x1,y1,x2,y2 in lines[0]:
    cv2.line(img,(x1,y1),(x2,y2),(0,255,0),2)

cv2.imwrite('houghlines5.jpg',img)
```



```
img = cv2.imread('opencv_logo.png',0)
img = cv2.medianBlur(img,5)
cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)

circles = cv2.HoughCircles(img,cv2.HOUGH_GRADIENT,1,20,
                           param1=50,param2=30,minRadius=0,maxRadius=0)

circles = np.uint16(np.around(circles))
for i in circles[0,:]:
    # draw the outer circle
    cv2.circle(cimg,(i[0],i[1]),i[2],(0,255,0),2)
    # draw the center of the circle
    cv2.circle(cimg,(i[0],i[1]),2,(0,0,255),3)

cv2.imshow('detected circles',cimg)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Пример приложения, подсчитывающего бревна методом HoughCircles ()





Вопрос:
при помощи каких функций, рассмотренных на лекции можно распознать данные фигуры?