



Кафедра молекулярных процессов и экстремальных  
состояний вещества

# Математические основы методов анализа результатов физического эксперимента

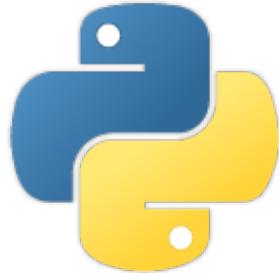
10. Задача распознавания образов. Прикладные  
задачи анализа изображений.

Коротеева Екатерина Юрьевна, ст. преп.  
Дорощенко Игорь Александрович, ст. н. с.

# Алгоритмы компьютерного зрения

# Алгоритмы компьютерного зрения

## используемое ПО



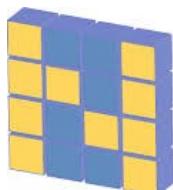
<https://opencv.org/>



Image Processing Toolbox  
<https://www.mathworks.com/>



scikit-image  
image processing in python



NumPy

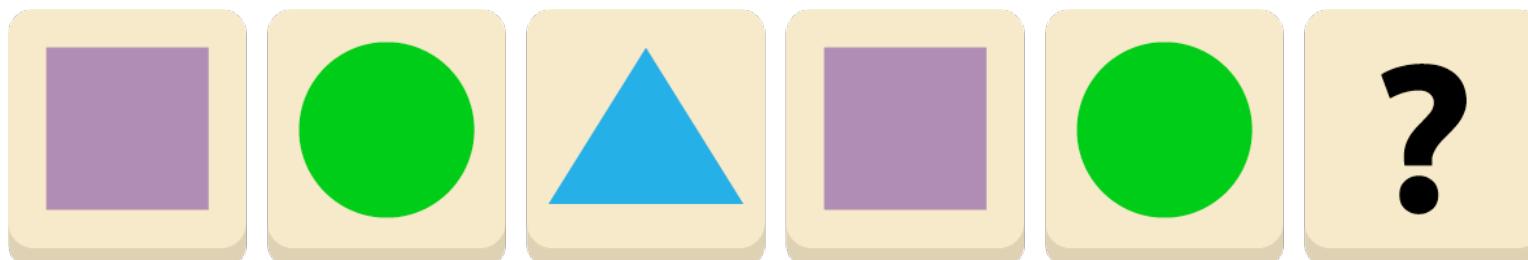


# Распознавание образов

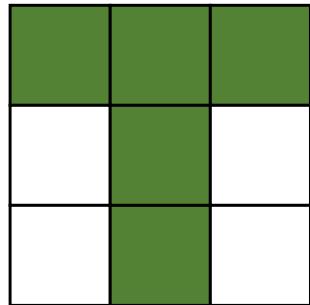
**Теория распознавания образа** — раздел информатики и смежных дисциплин, развивающий основы и методы классификации и идентификации предметов, явлений, процессов, сигналов, ситуаций и т. п. объектов, которые характеризуются конечным набором некоторых свойств и признаков.

Распознавание образов - повседневная неотъемлемая составляющая деятельности человеческого мозга.

**Образ** - это описание объекта или процесса, позволяющее выделять его из окружающей среды и группировать с другими объектами или процессами для принятия необходимых решений.



# Алгоритмы компьютерного зрения


$$\begin{bmatrix} [1,1,1] \\ [0,1,0] \\ [0,1,0] \end{bmatrix}$$

Изображение

Двумерный массив,  
содержащий значения пикселей  
(интенсивность)

Изображения представляются в виде числовых массивов (матриц), с которыми работают алгоритмы компьютерного зрения

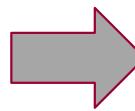
# Работа с пикселями

Изображения представляются в виде числовых значений:

- *Монохромные изображения:*  
каждый пиксель, как правило, задается **8 битами (1 байт)** информации. Всего **256** возможных оттенка
- *Цветные изображения:*  
каждый пиксель задается **24 битами (3 байта)** – RGB компоненты. Всего около **16 миллионов** цветов.
- *В программах обработки:*  
+ альфа канал (отвечает за прозрачность пикселя)

# Работа с пикселями

- Как правило, перед обработкой изображения переводятся из цветных в монохромные, что дает возможность рассматривать изображения как функции на плоскости.



Чтобы преобразовать цветное изображение в черно-белое, нужно найти среднее арифметическое значение  $R$ ,  $G$  и  $B$  каналов пикселя и затем это одно значение присвоить  $RGB$  каналам этого же пикселя

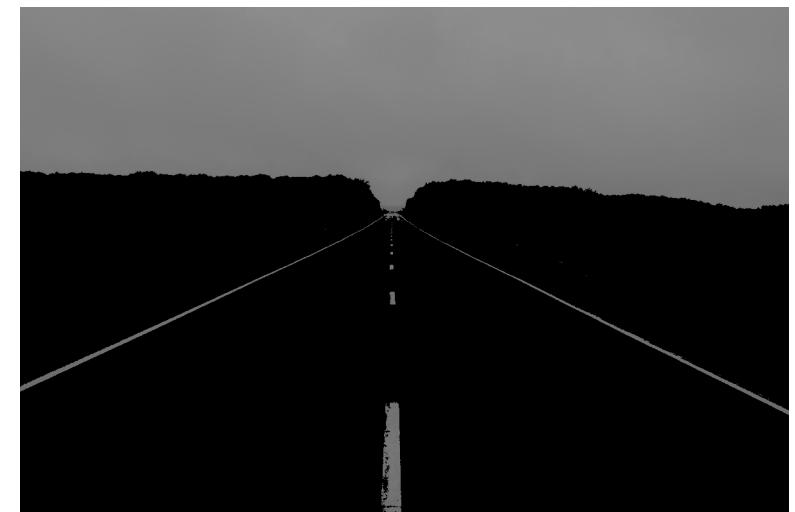
# Работа с пикселями

- Далее можно работать со значениями пикселей, меняя их. Например, оставить только достаточно яркие пиксели ( $I > 100$ ), чтобы выделить дорожную разметку:

```
import matplotlib.image as mpimg
import cv2

# Convert to grayscale
image_color = mpimg.imread('road.jpg')
image_gray = cv2.cvtColor(image_color, cv2.COLOR_BGR2GRAY)
cv2.imshow('Gray image', image_gray)
```

```
# Pixel intensity filtering
image_gray[(image_gray[:, :] < 100)] = 0
cv2.imshow('Only bright pixels', image_gray)
```



# Работа с пикселями: пример из газодинамики

## Пример:

Определение фронта ударной волны на теневых кадрах по распределению интенсивности свечения

## Теневой метод визуализации течений

Связь между плотностью однородного газа ( $\rho$ ) и ее показателем преломления ( $n$ ) можно выразить **соотношением Гладстона-Дейла**:

$$\frac{n-1}{\rho} = G \quad G - \text{постоянная Гладстона-Дейла}$$

(для воздуха 0,226 см<sup>3</sup>/г)

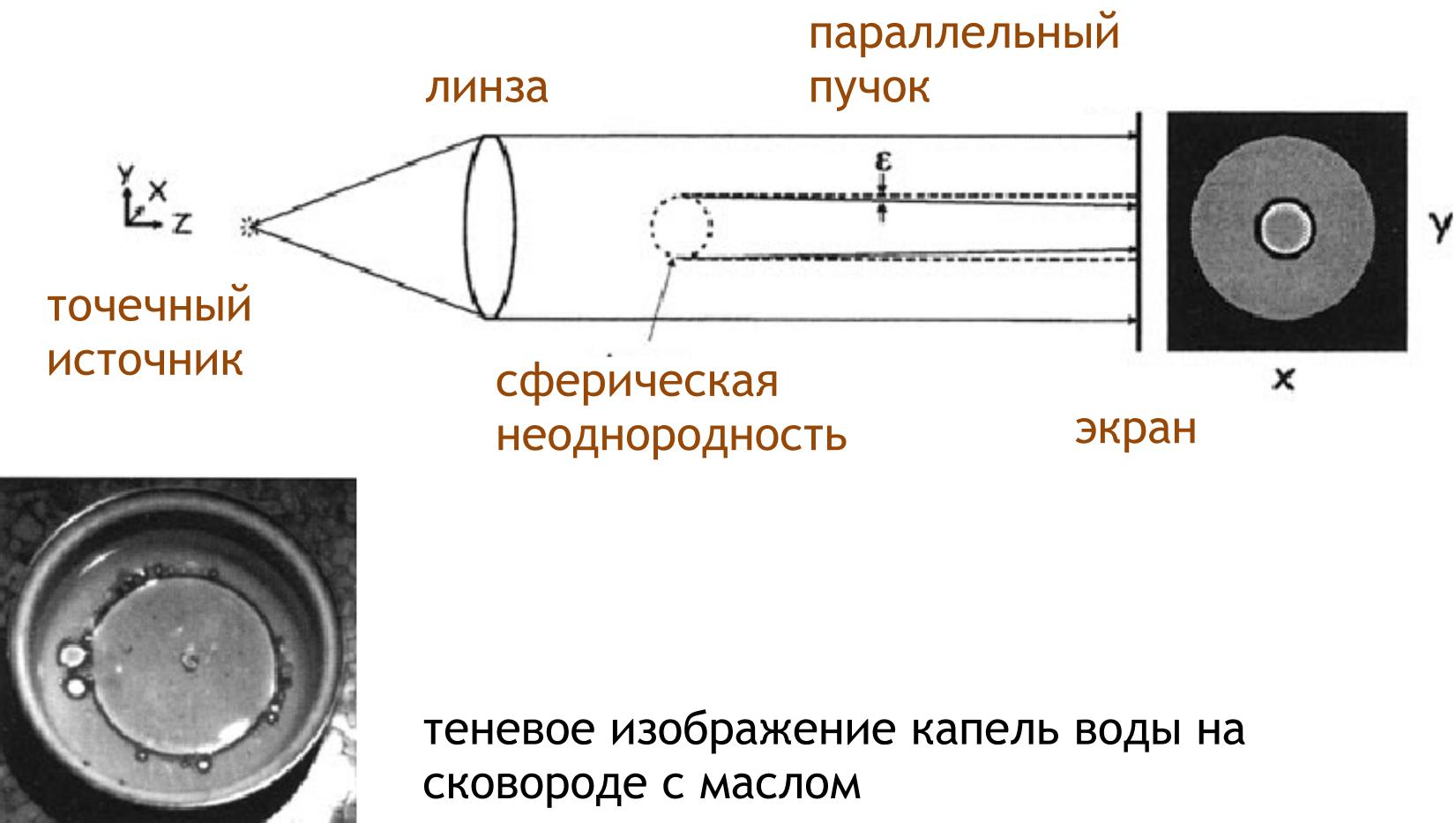
Отклонение луча:

при изменении на 1 К в воде отклонение 47"  
при изменении на 10 К в воздухе отклонение 3.6".

$$\alpha \cong \int_0^L \frac{\partial}{\partial x} \ln n(x, y, z) dz$$

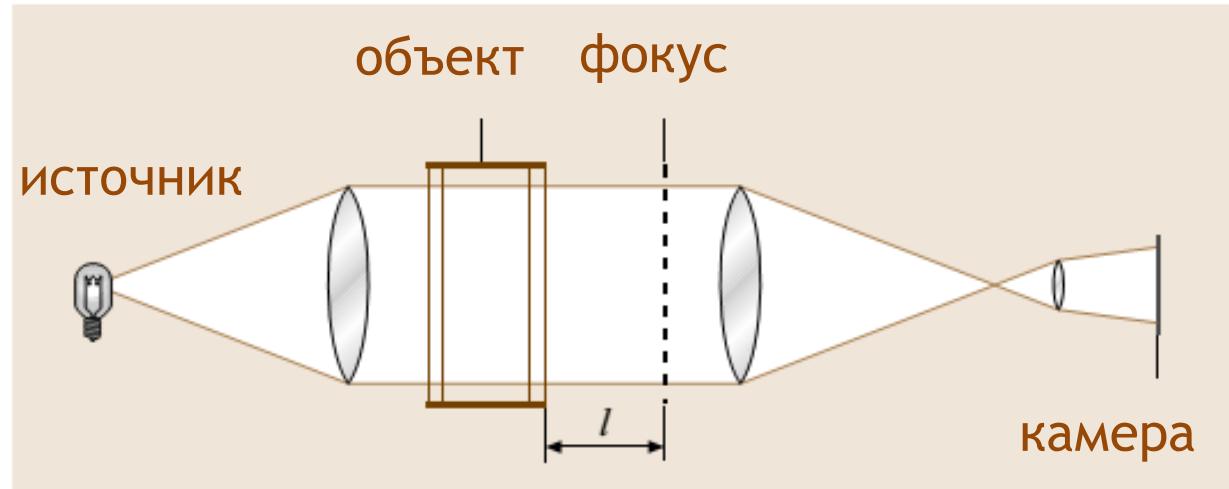
# Работа с пикселями: пример из газодинамики

- Прямой теневой метод

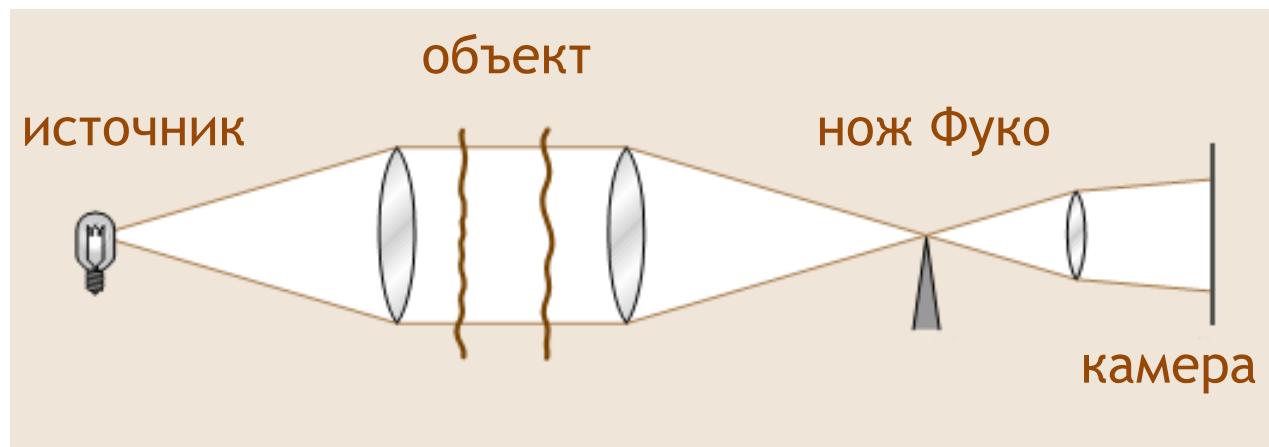


# Работа с пикселями: пример из газодинамики

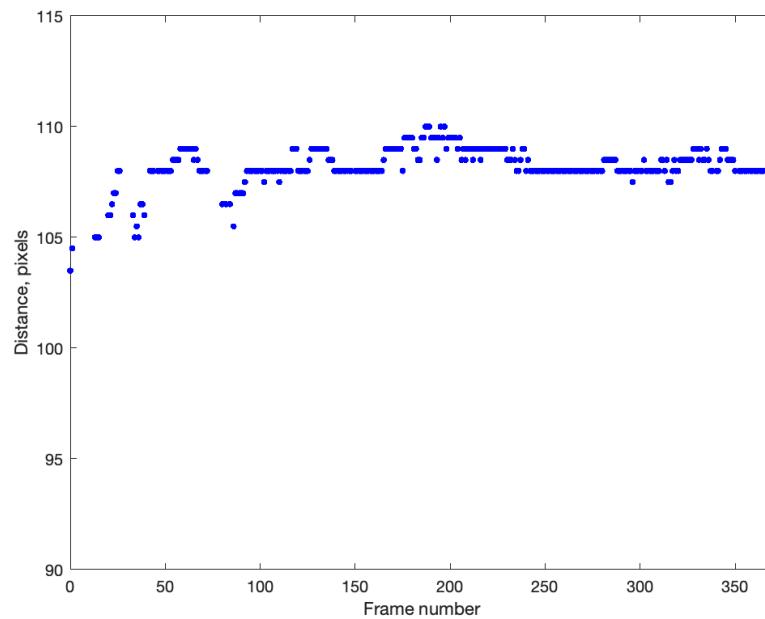
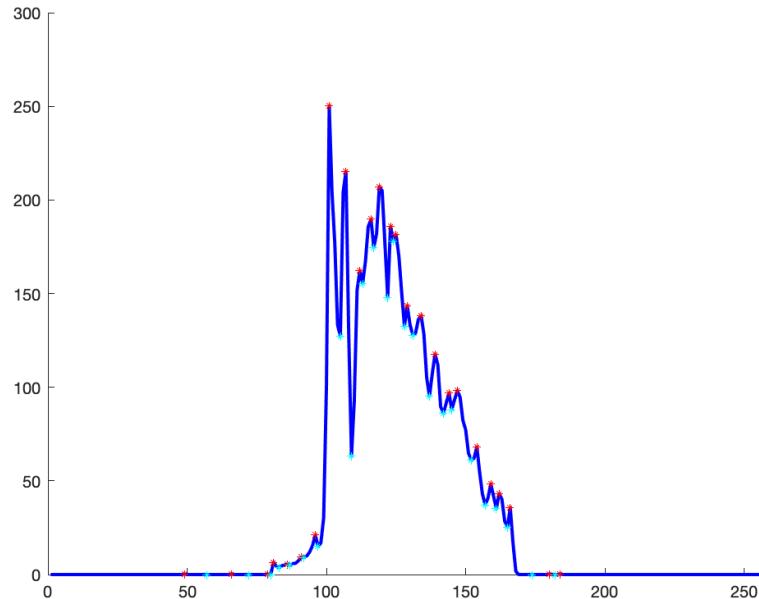
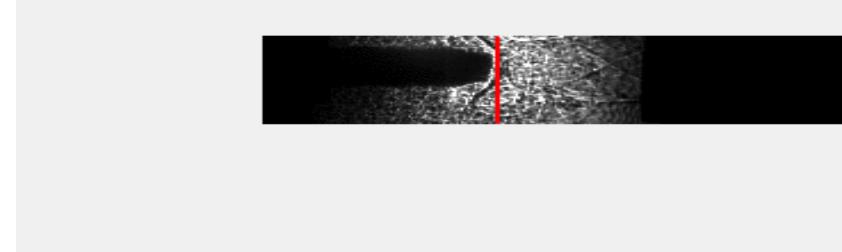
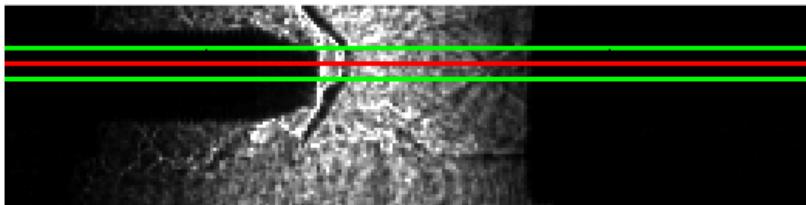
- Теневой метод регистрируется изменение градиента плотности  $d^2n/dx^2$



- Шлирен-метод (метод Тэплера) регистрируется изменение плотности  $dn/dx$



# Работа с пикселями: пример из газодинамики

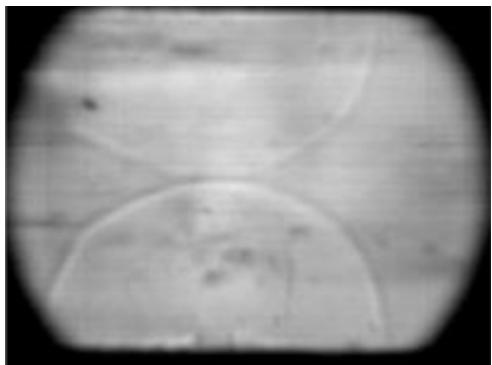


Поиск локальных экстремумов программными методами и нахождение паттерна, соответствующего головной ударной волне для каждого изображения на видео

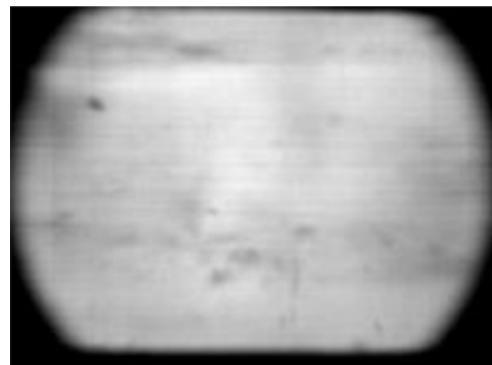
Автоматическое определение расстояния между головной ударной волной и моделью

# Работа с пикселями: пример из газодинамики

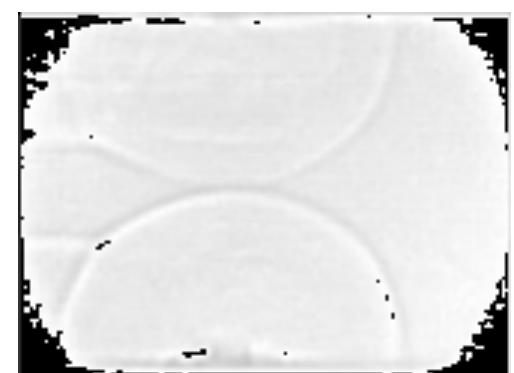
- Попиксельное вычитание фона



-



=

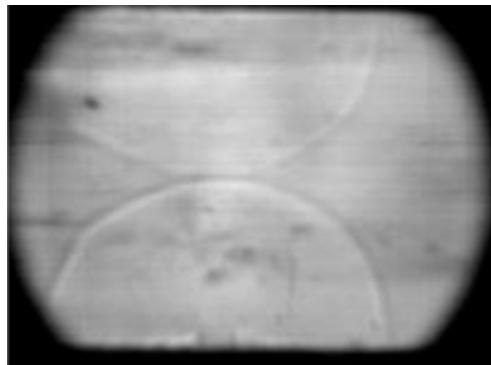


```
# Background image subtraction
source_image = cv2.imread('explosion.png', cv2.IMREAD_COLOR)
source_image = cv2.cvtColor(source_image, cv2.COLOR_BGR2GRAY)
bg_image = cv2.imread('bg_explosion.png', cv2.IMREAD_COLOR)
bg_image = cv2.cvtColor(bg_image, cv2.COLOR_BGR2GRAY)
subtracted_image = source_image - bg_image

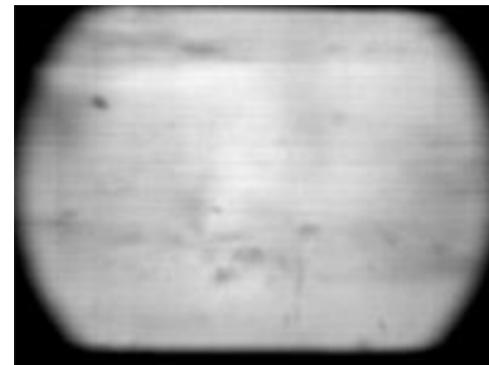
cv2.imshow('Background subtraction', subtracted_image)
```

# Работа с пикселями: пример из газодинамики

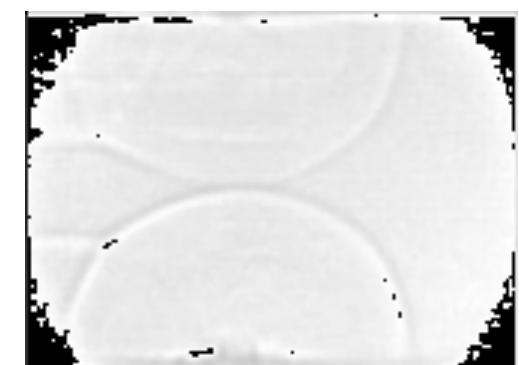
- Попиксельное вычитание фона



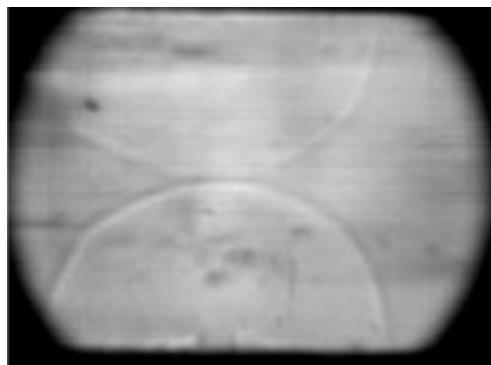
-



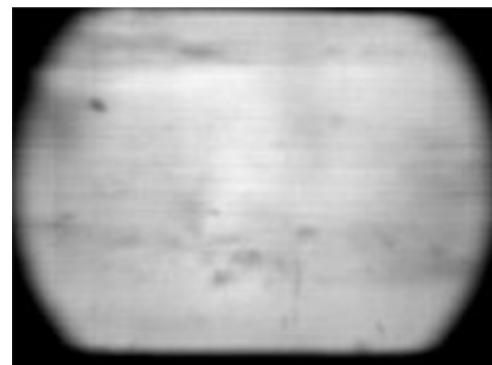
=



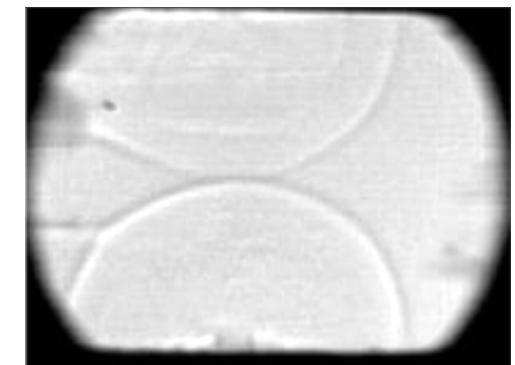
- Более продвинутый метод – с выравниванием интенсивности фона



-



=



# Алгоритмы компьютерного зрения: свертка

# Алгоритмы компьютерного зрения: свертка

**Операция свертки** (convolutional operation) – процесс сложения каждого элемента изображения со своими соседними элементами (пикселями), умноженными на определенные коэффициенты (веса).

Одна из самых важных операций в CV.

Применяется для выделения характерных структур изображений, например границ, углов.

$$g(x,y) \bullet f(x,y) = \sum_{n=0}^{\text{cols}} \sum_{m=0}^{\text{rows}} g(n, m) * f(x - n, y - m)$$

$f$  – исходное изображение,  $g$  – ядро (kernel) – матрицы, содержащие дискретные значения пикселей.  
Поэтому для изображений свертка – матричная операция.

# Алгоритмы компьютерного зрения: свертка

	0	0
0	1	0
0	0	0
1	0	1
0	0	0

Изображение ( $f(x, y)$ )

1	0	0
1	0	1
0	1	1

Ядро ( $g(x, y)$ )

$$0*0 + 0*1 + 0*1 + 1*1$$

=

1					

Результат

Операция повторяется для каждого пикселя на изображении

# Алгоритмы компьютерного зрения: свертка

0	0	1	0	0
0	1	0	0	0
0	0	0	1	0
1	0	1	1	0
0	0	0	1	0



1	0	0
1	0	1
0	1	1

=

1	2	0	1	0
1	0	2	2	0
1	1	4	1	

Изображение ( $f(x, y)$ )

Ядро ( $g(x, y)$ )

Результат

0	0	1	0	0
0	1	0	0	0
0	0	0	1	0
1	0	1	1	0
0	0	0	1	0




1	0	0
1	0	1
0	1	1

=

1	2	0	1	0
1	0	2	2	0
1	1	4	1	1
0	2	2	2	2
0	1	1	1	2

Изображение ( $f(x, y)$ )

Ядро ( $g(x, y)$ )

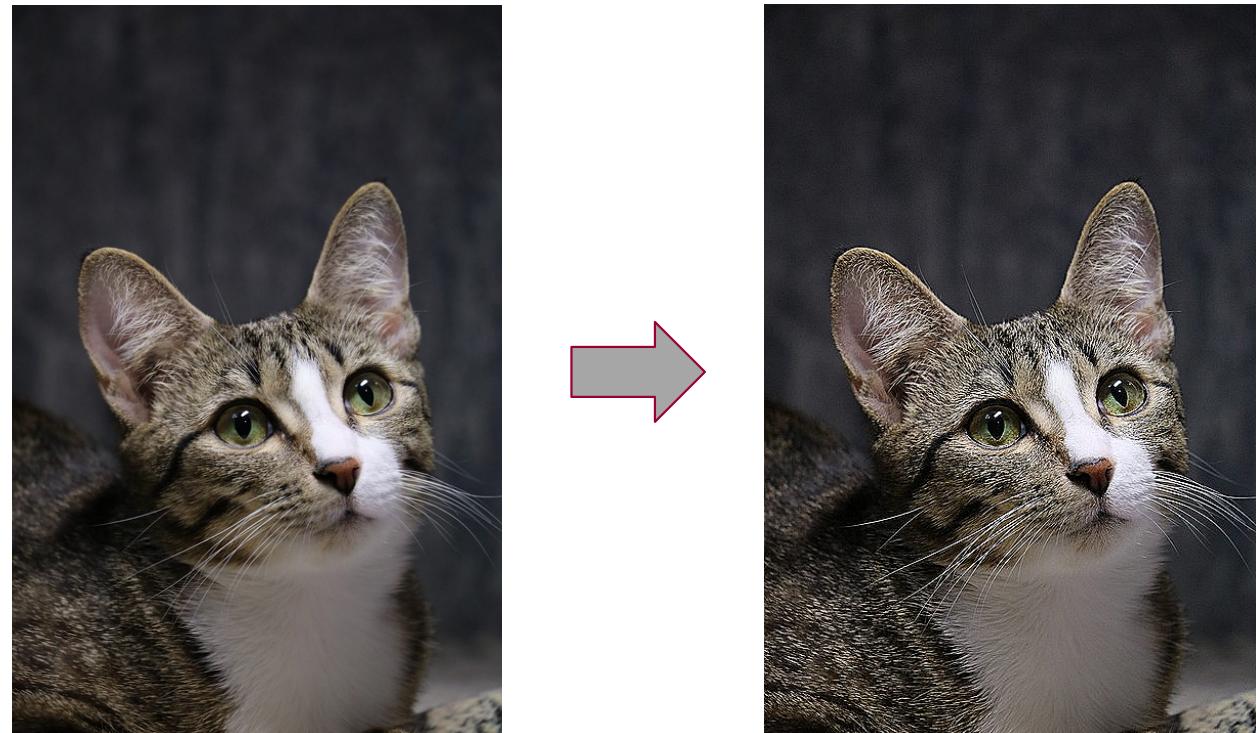
Результат

# Алгоритмы компьютерного зрения: свертка

0	-1	0
-1	5	-1
0	-1	0

Такой выбор ядра увеличивает **резкость**. Для каждого пикселя:

- увеличивается интенсивность данного пикселя
- уменьшается интенсивность соседних пикселей



# Алгоритмы компьютерного зрения: свертка

0	-1	0
-1	5	-1
0	-1	0

Такой выбор ядра увеличивает **резкость**. Для каждого пикселя:

- увеличивается интенсивность данного пикселя
- уменьшается интенсивность соседних пикселей

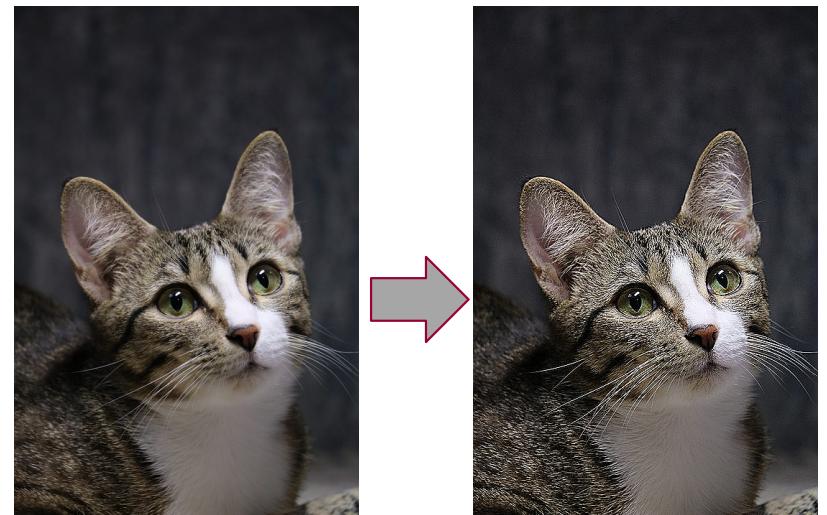
```
# SHARPEN KERNEL
original_image = cv2.imread('cat.jpeg', cv2.IMREAD_COLOR)

kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])

sharpen_image = cv2.filter2D(original_image, -1, kernel)

cv2.imshow('Original Image', original_image)
cv2.imshow('Sharpen Image', sharpen_image)

cv2.filter2D(src, ddepth, kernel)
```



# Алгоритмы компьютерного зрения: свертка

1/25

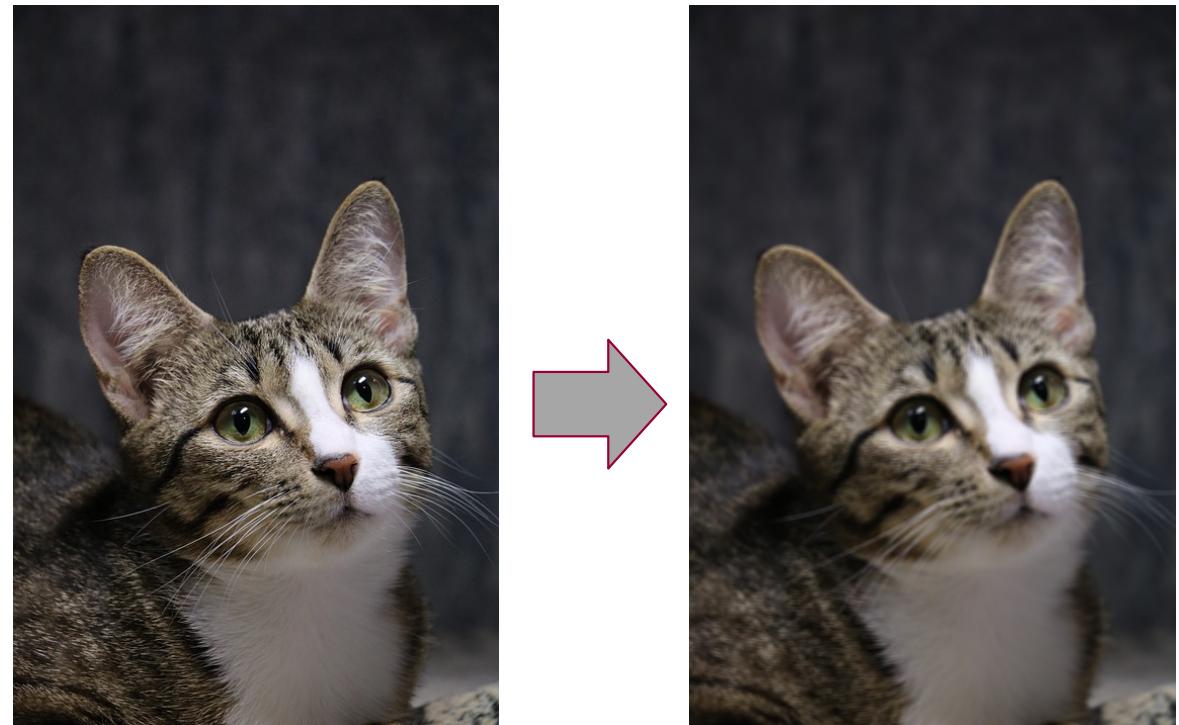
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

```
kernel = np.ones((5, 5)) / 25
```

! Сумма значений в ядре должна быть равна 1, иначе будет меняться яркость изображения.

Такой выбор ядра увеличивает **размытость**. Для каждого пикселя:

- не изменяется интенсивность данного пикселя
- учитывается интенсивность соседних пикселей

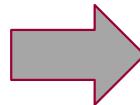


# Алгоритмы компьютерного зрения: свертка

0	1	0
1	-4	1
0	1	0

Такой выбор ядра позволяет выделить границы. Для каждого пикселя:

- уменьшается интенсивность данного пикселя
- не меняется интенсивность соседних пикселей

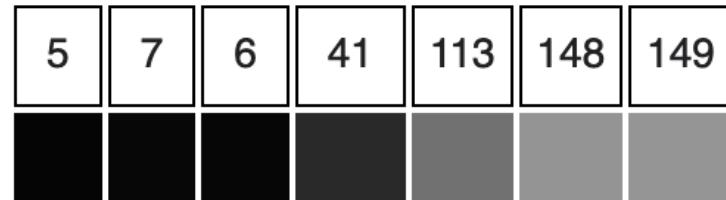
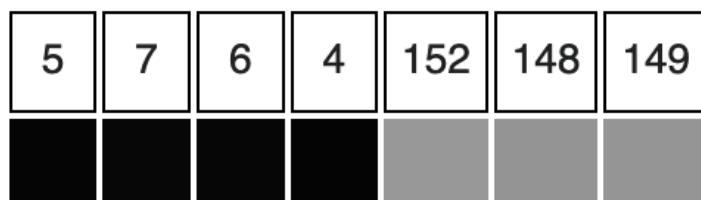
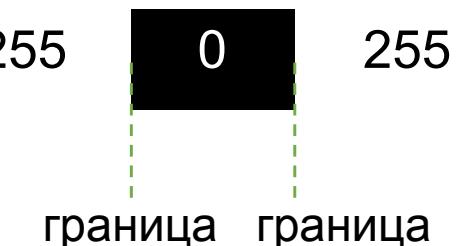


## Выделение границ (Edge detection)

# Выделение границ (Edge detection)

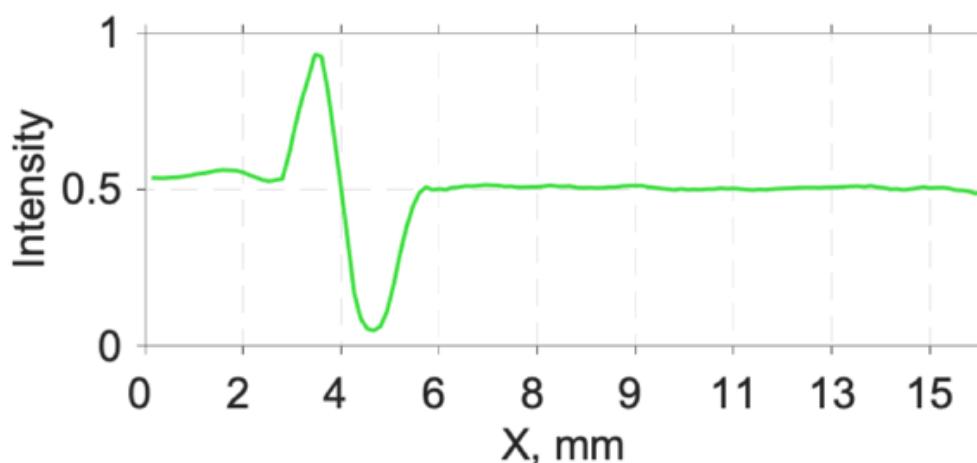
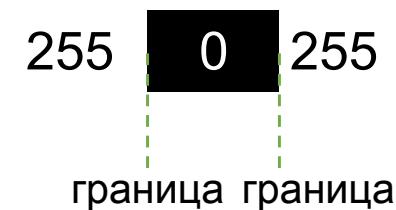
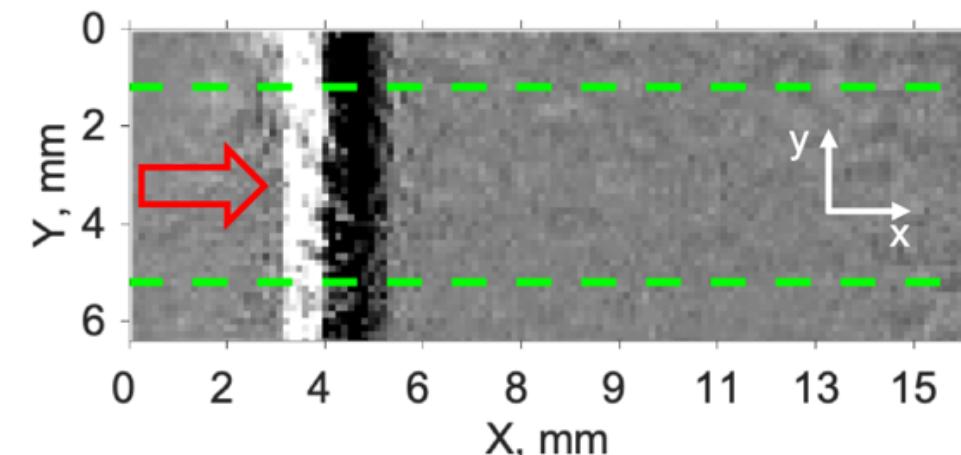
- **Выделение границ** позволяет находить точки на изображении, в которых яркость значительно меняется или содержит разрывы
- **Выделение границ** - важный инструмент в CV, на его основе часто делается выявление признаков (feature detection), которые используются, например, для классификации изображений и обучения нейронных сетей.

Простейший способ поиска границ – вычисление разности между соседними пикселями:



# Выделение границ (Edge detection)

Простейший способ поиска границ – вычисление разности между соседними пикселями:



# Выделение границ (Edge detection)

## ➤ метод Собеля

- основан на вычислении градиентов яркости – производных первого порядка
- производные вычисляются независимо по осям X и Y
- оператор Собеля – матрица размером 3x3, свертка которой с матрицей изображения позволяет вычислить градиенты (1968 г)

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

A – исходное изображение,  $G_x$ ,  $G_y$  – градиенты.

# Выделение границ (Edge detection)

## ➤ метод Собеля

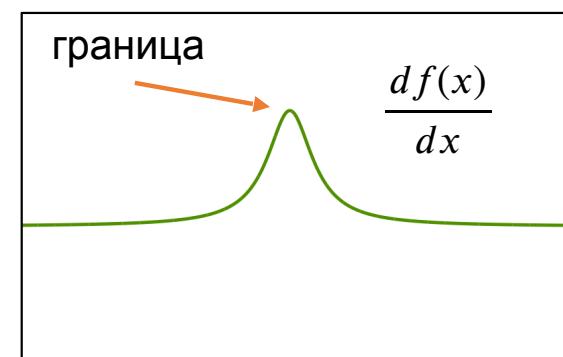
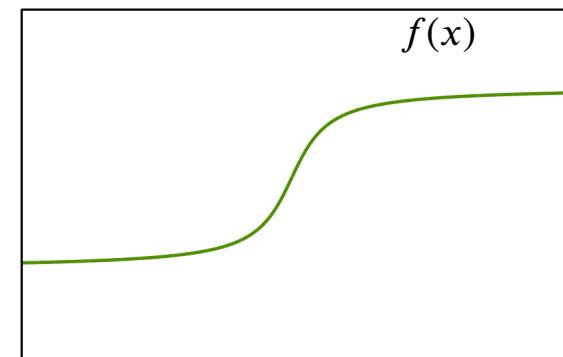
$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A$$

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

A – исходное изображение,  
 $G_x, G_y$  – градиенты.

Выделение границ на основе  
вычисления градиента  
(производная 1 порядка)

Аналогия с функцией 1 аргумента:

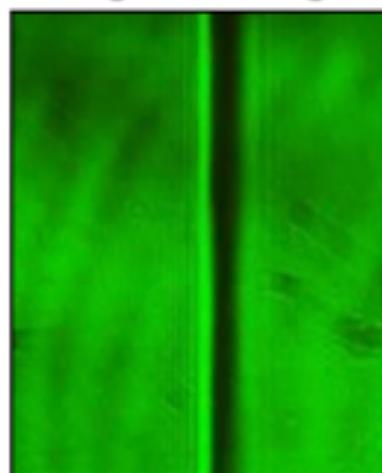


# Выделение границ (Edge detection)

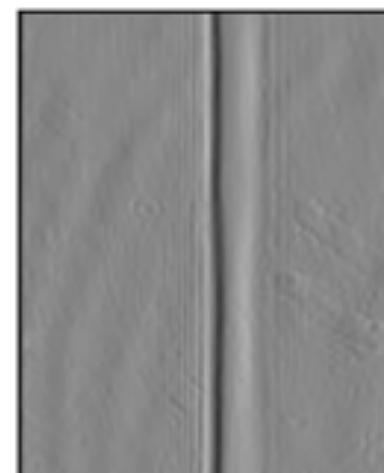
## ➤ метод Собеля

```
# Sobel Calculations
img = cv2.imread('light_diffraction_on_shock.jpg', cv2.IMREAD_COLOR)
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
x_sobel = cv2.Sobel(gray_img, cv2.CV_64F, 1, 0, ksize = 7)
y_sobel = cv2.Sobel(gray_img, cv2.CV_64F, 0, 1, ksize = 7)
```

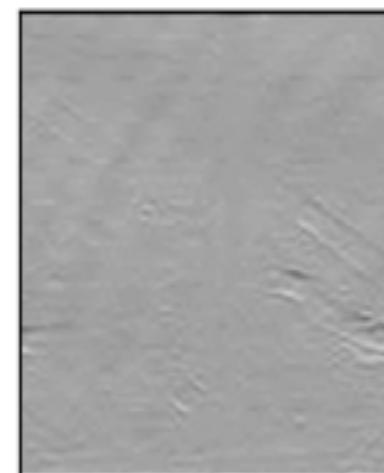
Original Image



Sobel - X direction



Sobel - Y direction



# Выделение границ (Edge detection)

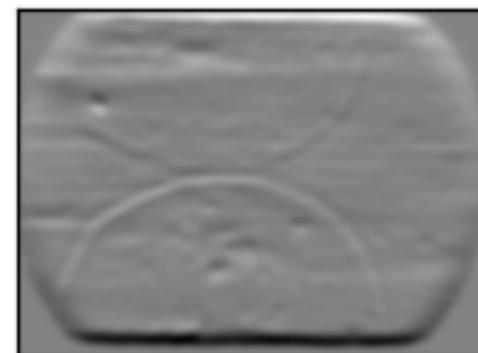
## ➤ метод Собеля

```
# Sobel Calculations
img = cv2.imread('light_diffraction_on_shock.jpg', cv2.IMREAD_COLOR)
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
x_sobel = cv2.Sobel(gray_img, cv2.CV_64F, 1, 0, ksize = 7)
y_sobel = cv2.Sobel(gray_img, cv2.CV_64F, 0, 1, ksize = 7)
```

Original Image



Sobel - X direction



Sobel - Y direction



# Выделение границ (Edge detection)

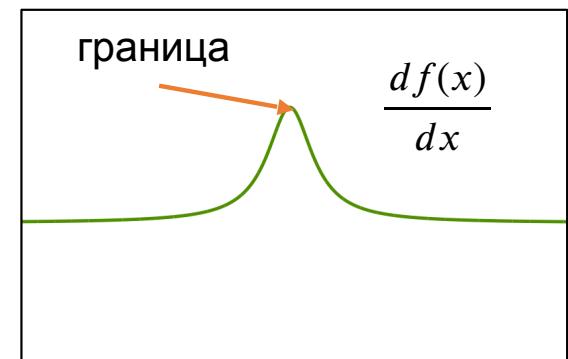
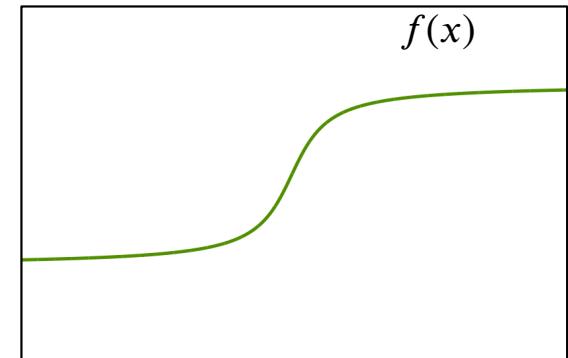
## ➤ метод Лапласа

- Вычисляется производная второго порядка за один проход по изображению и определяются точки пересечение производной с нулем

0	-1	0
-1	4	-1
0	-1	0

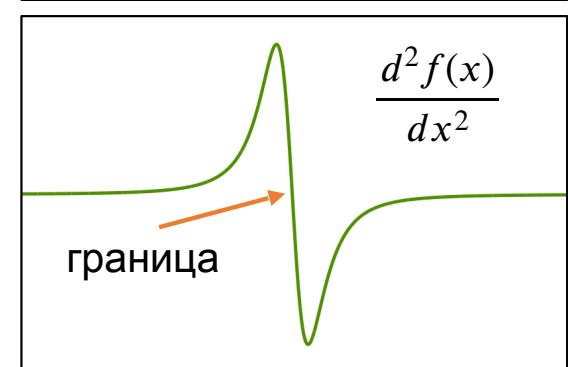
Ядро фильтра Лапласа

Аналогия с функцией 1 аргумента:



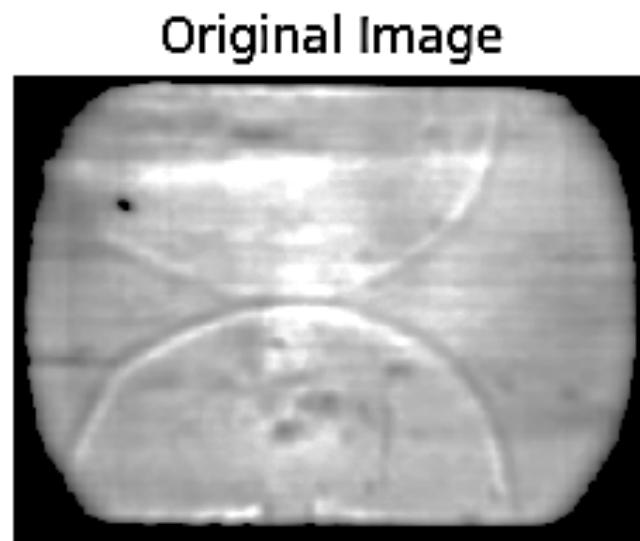
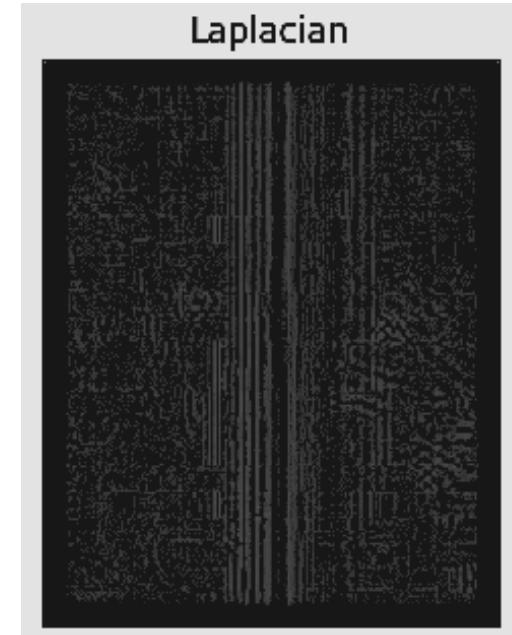
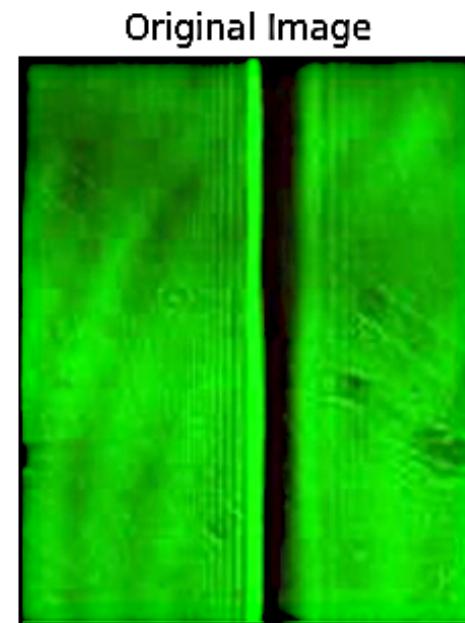
Выделение границ на основе  
вычисления градиента  
(производная 1 порядка)

Выделение границ на основе  
**оператора Лапласа**  
(производная 2 порядка)



# Выделение границ (Edge detection)

## ➤ метод Лапласа



# Выделение границ (Edge detection)

## ➤ метод Кэнни (Canny)

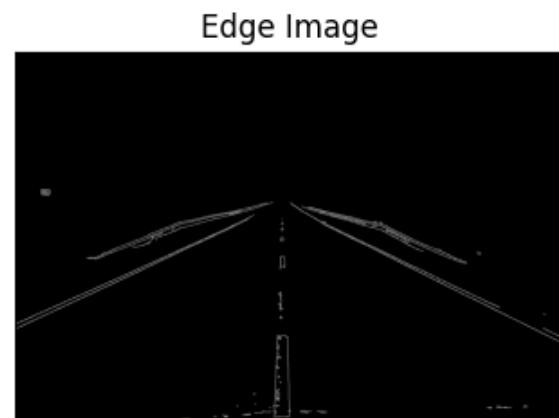
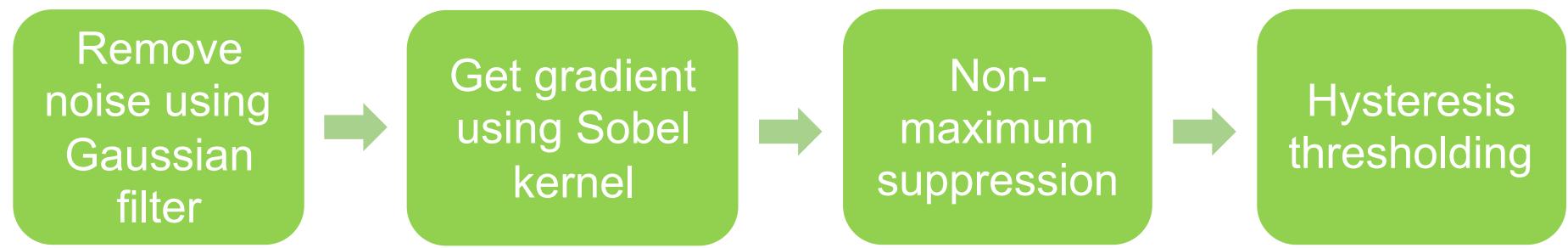
- наиболее продвинутый и часто используемый на практике метод
- J. Canny, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vols. PAMI-8, no. 6, pp. 679-698, 1986.

Основные шаги алгоритма:

1. **Сглаживание** изображения с помощью *фильтра Гаусса* с интенсивностью  $\sigma$ .
2. **Вычисление градиента** с помощью *метода Собеля*: вычисление величины и направления градиентов
3. **Non-maximum suppression**: обнуление пикселей, которые не соответствуют максимумам вдоль направления градиента
4. **Thresholding**: фильтрация недостоверных границ

# Выделение границ (Edge detection)

## ➤ метод Кэнни (Canny)



```
cv2.Canny(image, T_lower, T_upper, aperture_size, L2Gradient)
```

[https://docs.opencv.org/3.4/da/d22/tutorial\\_py\\_canny.html](https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html)

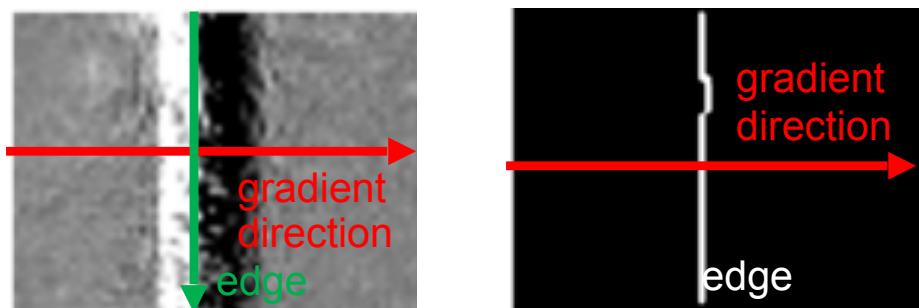
# Выделение границ (Edge detection)

## ➤ метод Кэнни (Canny)

- 1) Удаление шумов фильтром Гаусса ( $5 \times 5$ )

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- 3) Подавление не-максимумов

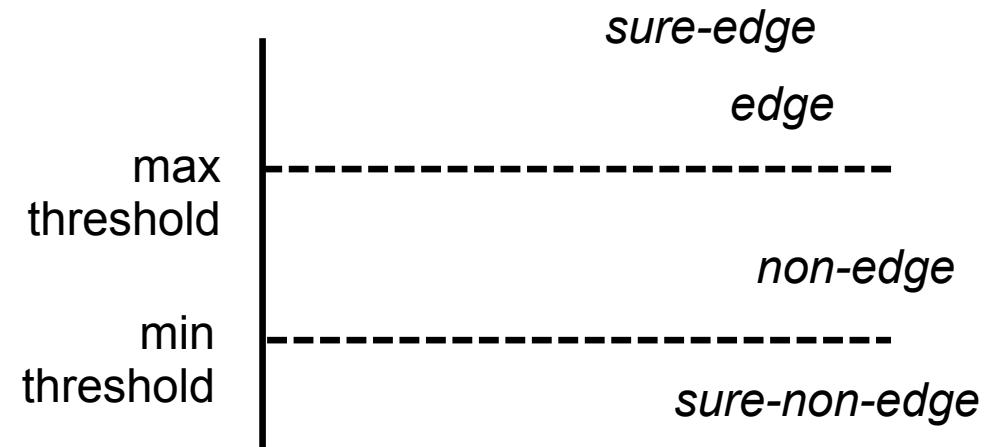


The result is a binary image with "thin edges" formed by the maximum local gradient values

- 2) Поиск градиентов

$$I_x = I * \frac{\partial G}{\partial x}, \quad I_y = I * \frac{\partial G}{\partial y}$$

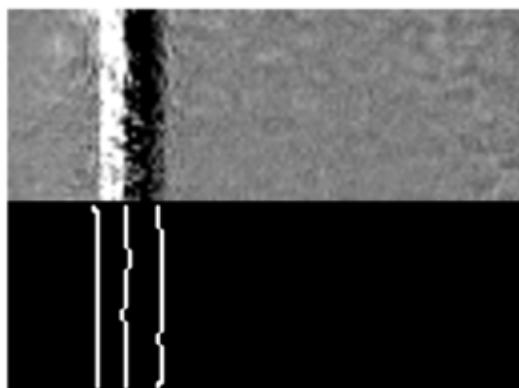
- 4) Двойная пороговая фильтрация



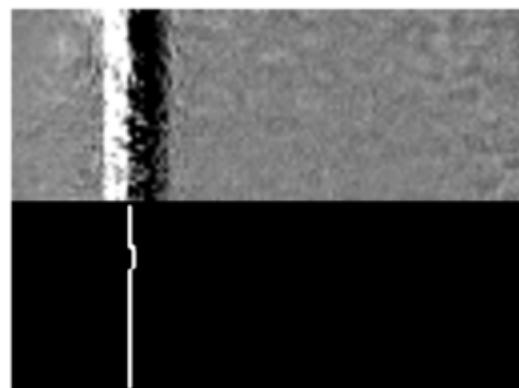
# Выделение границ (Edge detection)

## ➤ метод Кэнни (Canny)

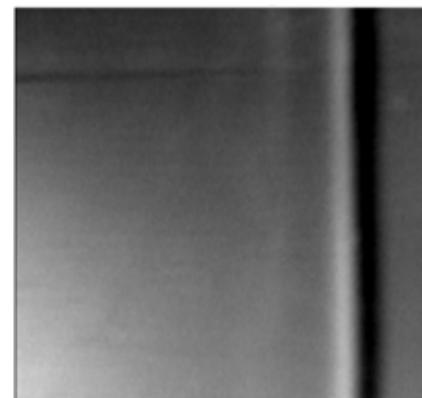
Пример 1: выделения ударных волн



maxThreshold = 0.26  
 $\sigma = 2.1$



maxThreshold = 0.71  
 $\sigma = 2.6$



maxThreshold = 0.25  
 $\sigma = 3.2$

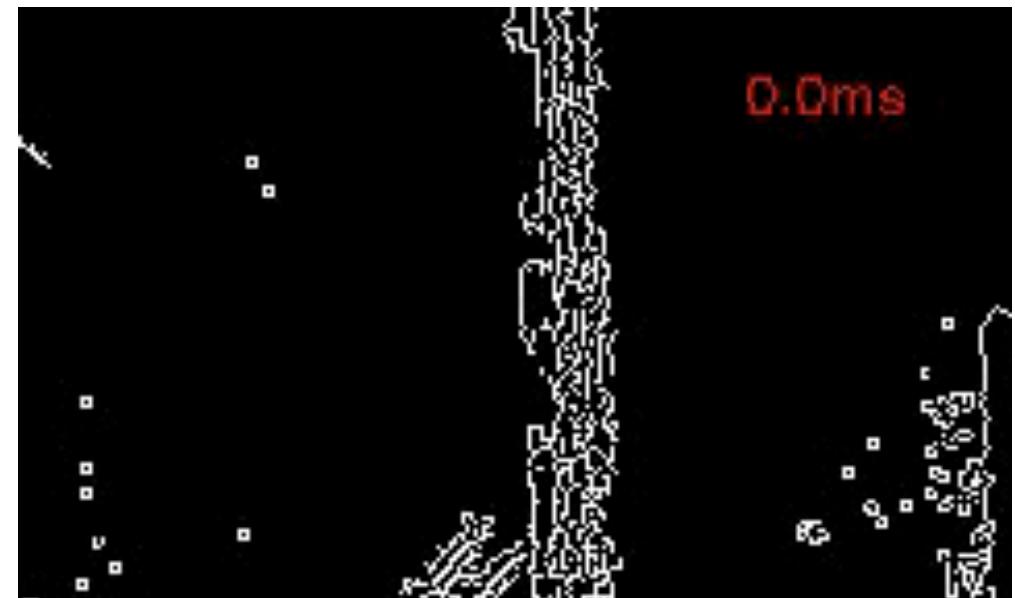
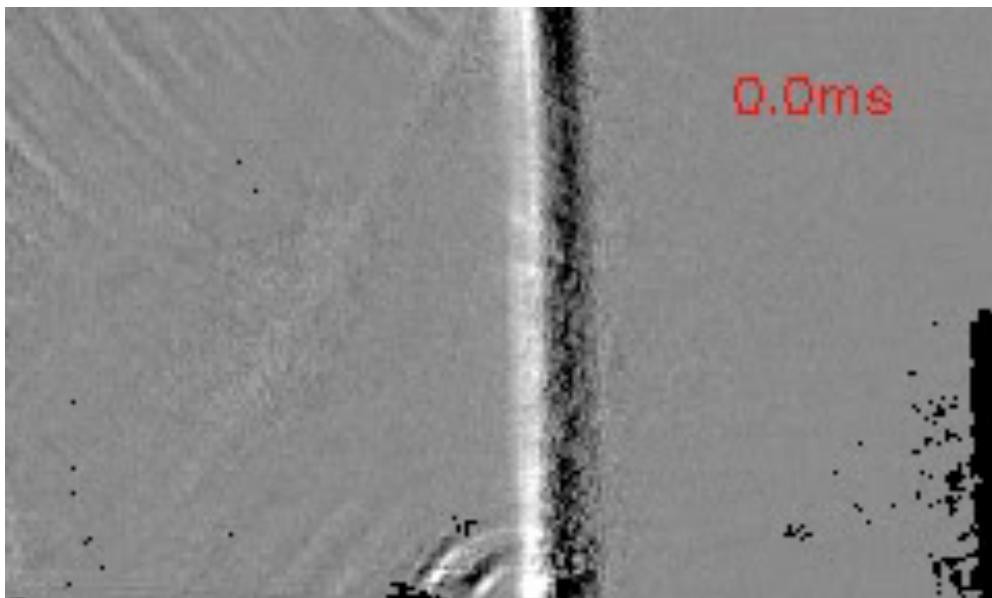


minThreshold =  $0.4 * \text{maxThreshold}$  для всех примеров

# Выделение границ (Edge detection)

## ➤ метод Кэнни (Canny)

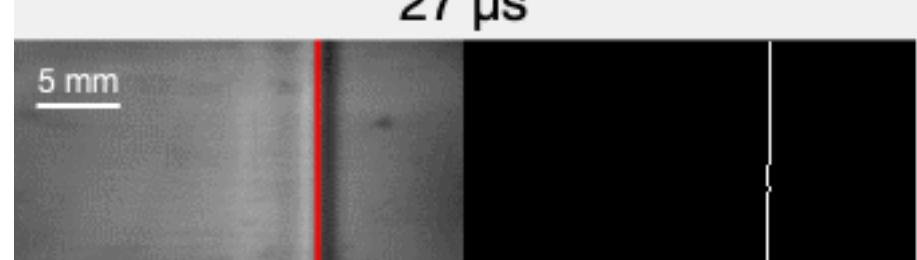
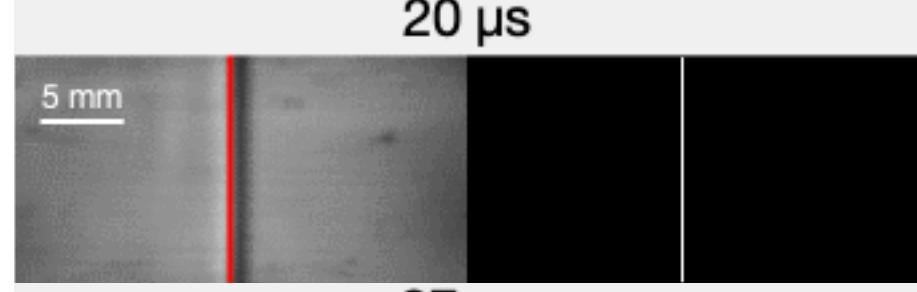
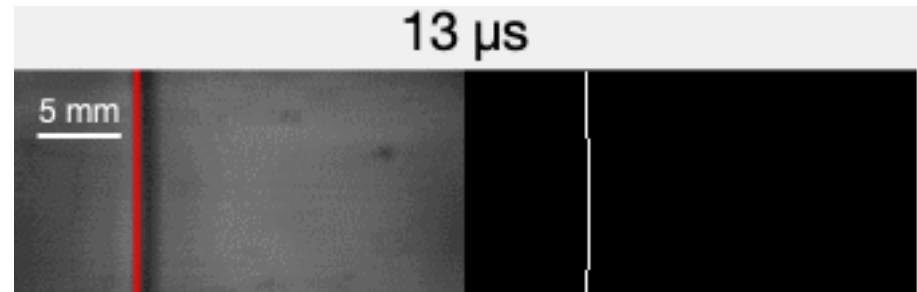
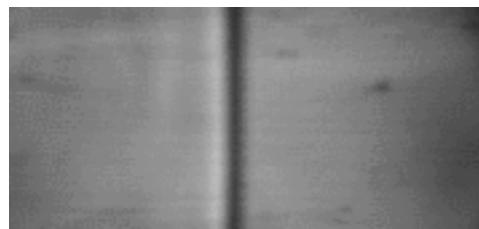
Пример 2: Прохождение плоской ударной волны через прямоугольный канал с точечным препятствием на его нижней стенке.  
Косой скачок уплотнения в сверхзвуковом потоке.



# Выделение границ (Edge detection)

## ➤ метод Кэнни (Canny)

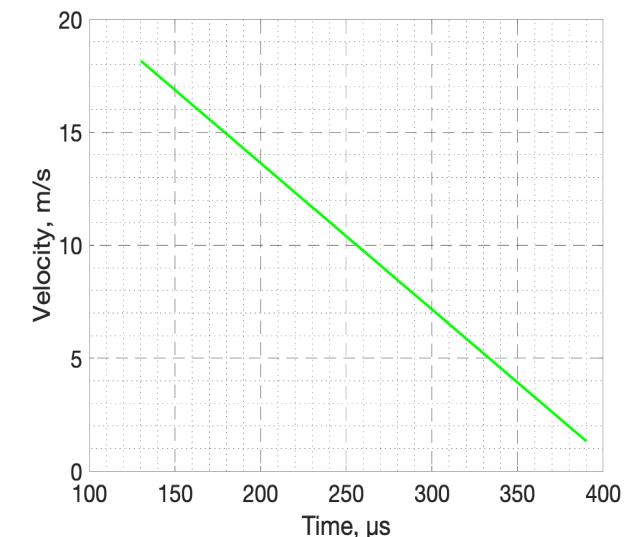
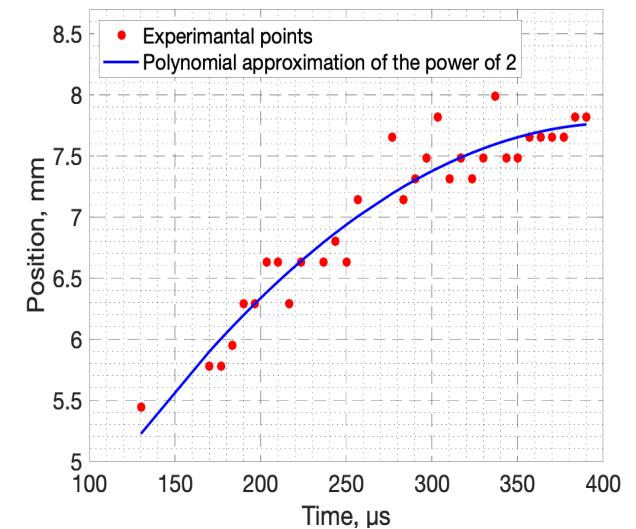
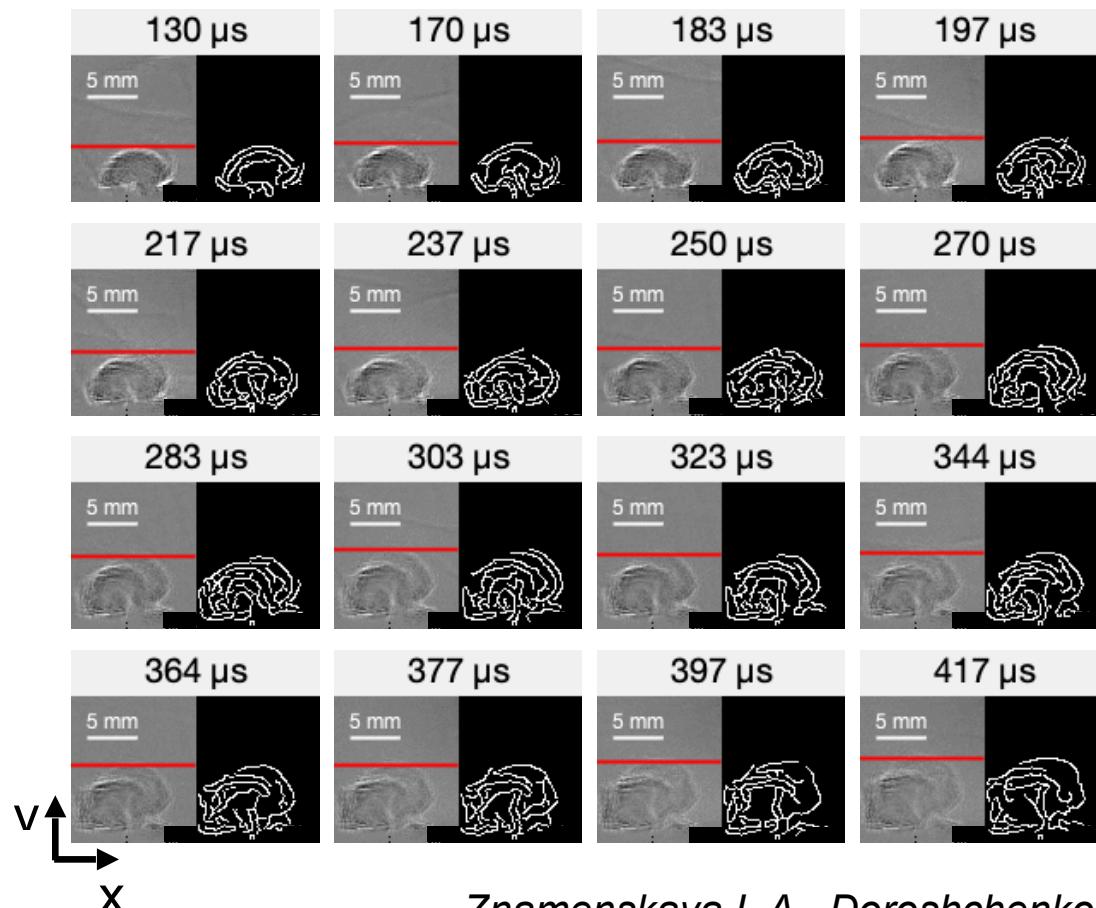
Пример 3: Число Маха УВ:  $M = 2 - 3$ .  
Скорость съемки до 525 000 кадров / с



# Выделение границ (Edge detection)

## ➤ метод Кэнни (Canny)

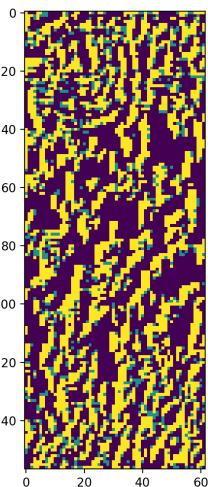
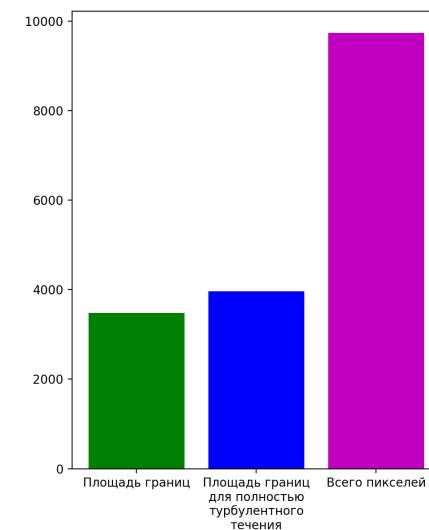
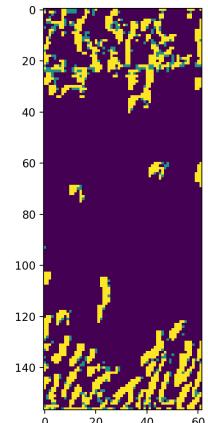
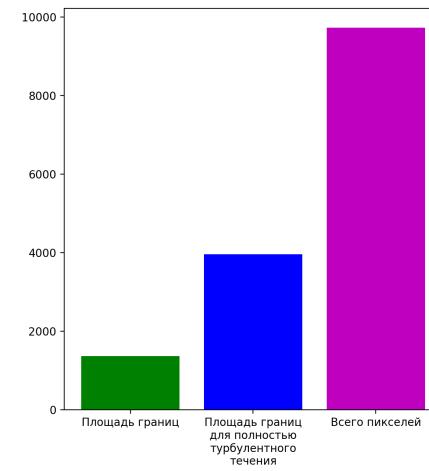
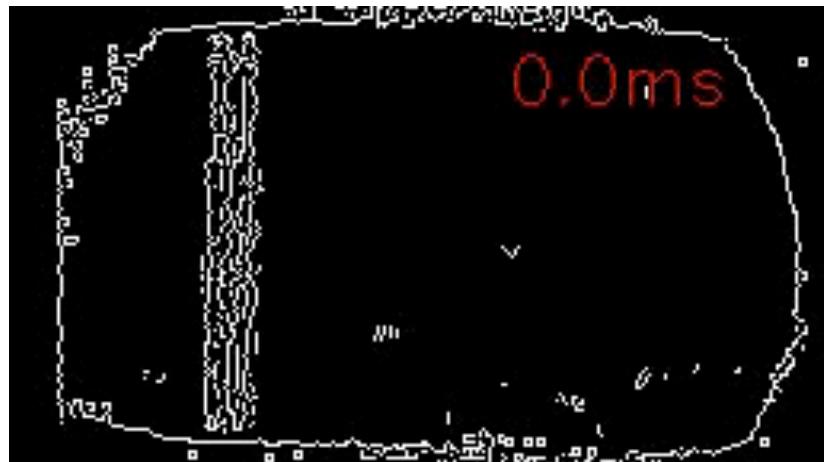
Пример 4: Автоматизированное распознавание конвективного термика



# Выделение границ (Edge detection)

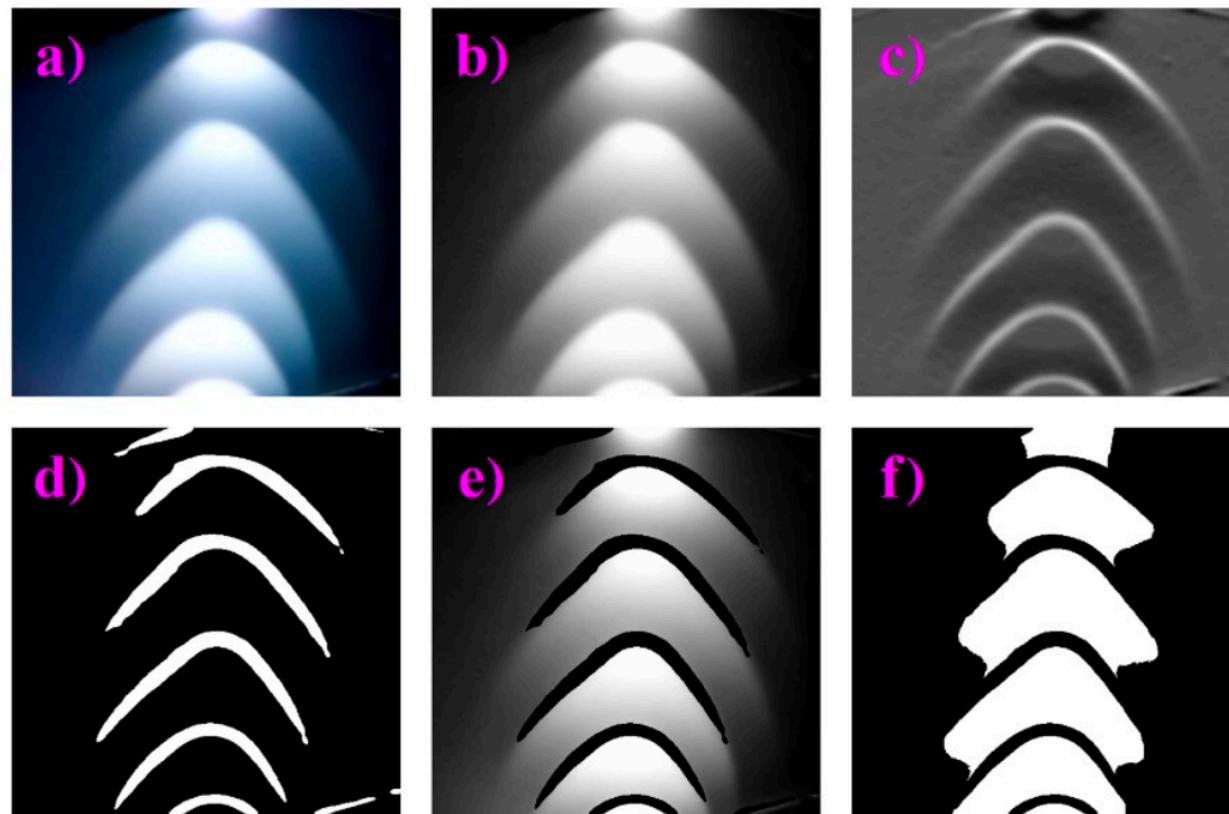
## ➤ метод Кэнни (Canny)

Пример 5: Оценка толщины пограничного слоя



# Выделение границ (Edge detection)

Выделение характерных структур свечения разряда в потоке



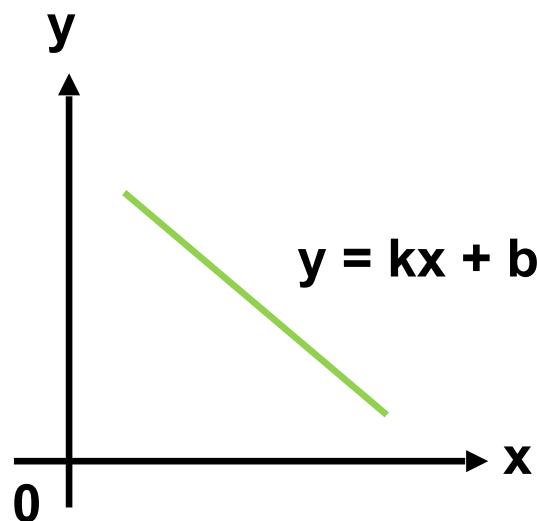
- A. Gaussian filtration.
- B. Grayscale image.
- C. Shadowgraph (Schlieren-like) image (**gradient filter**).
- D. Binary layers image.
- E. Composed image.
- F. Output mask layers (binary mask image).

**Figure 3.** Image processing for large-scale strata.

# Преобразование Хафа

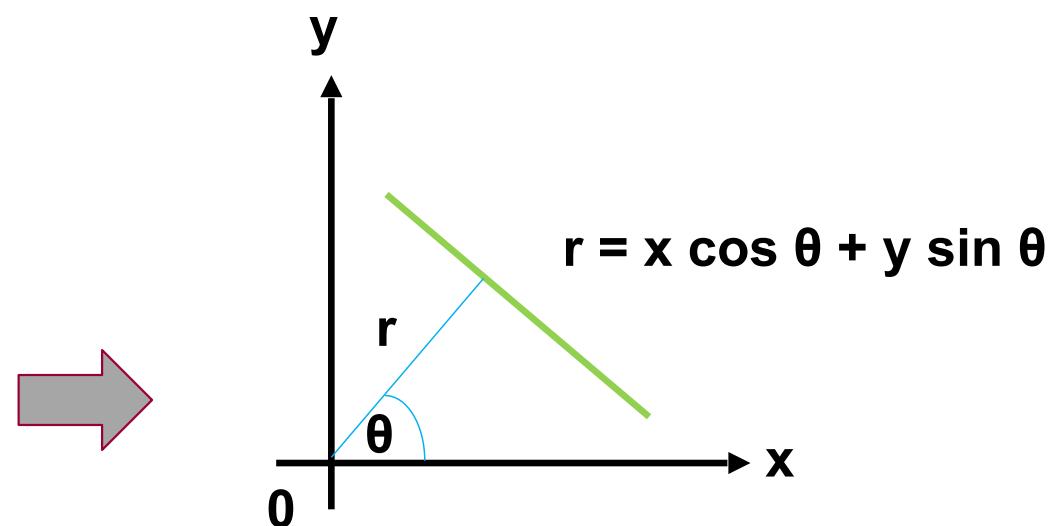
# Преобразование Хафа

Преобразование Хафа (Hough Transform) — вычислительный алгоритм, применяемый для параметрической идентификации геометрических элементов растрового изображения (патент 1962 г.)



*Декартова система координат*

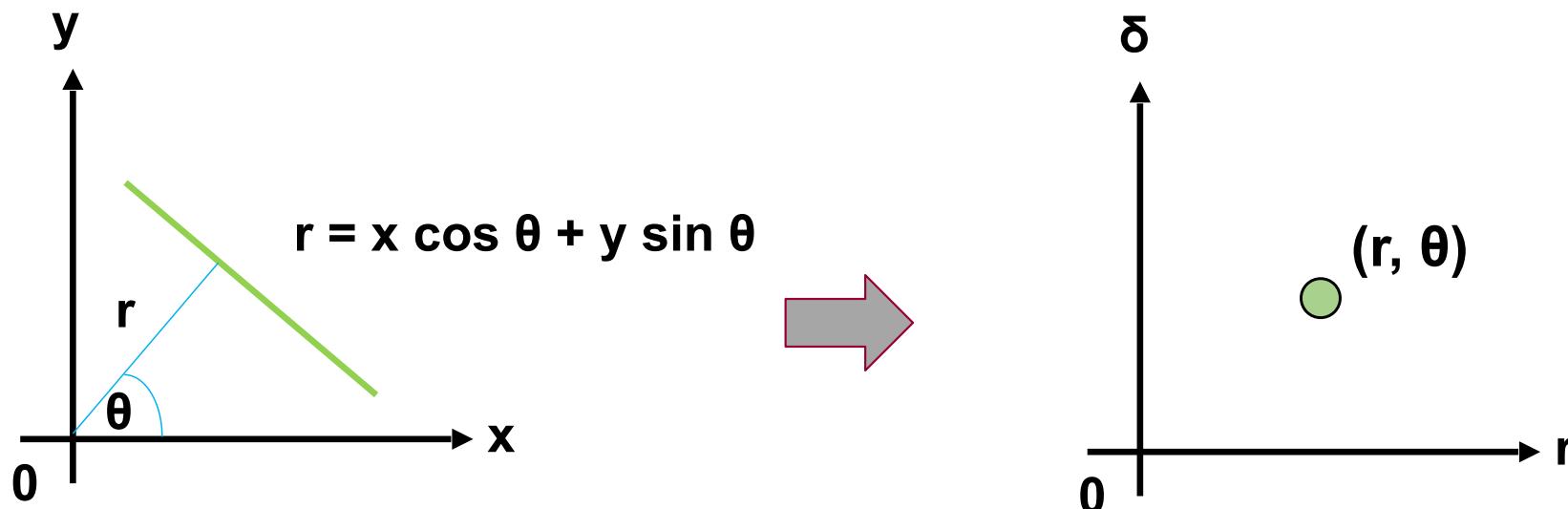
Проблема в том, что нет возможности задавать вертикальные прямые



*Полярная система координат*

# Преобразование Хафа

Преобразование Хафа (Hough Transform) — вычислительный алгоритм, применяемый для параметрической идентификации геометрических элементов растрового изображения (патент 1962 г.)



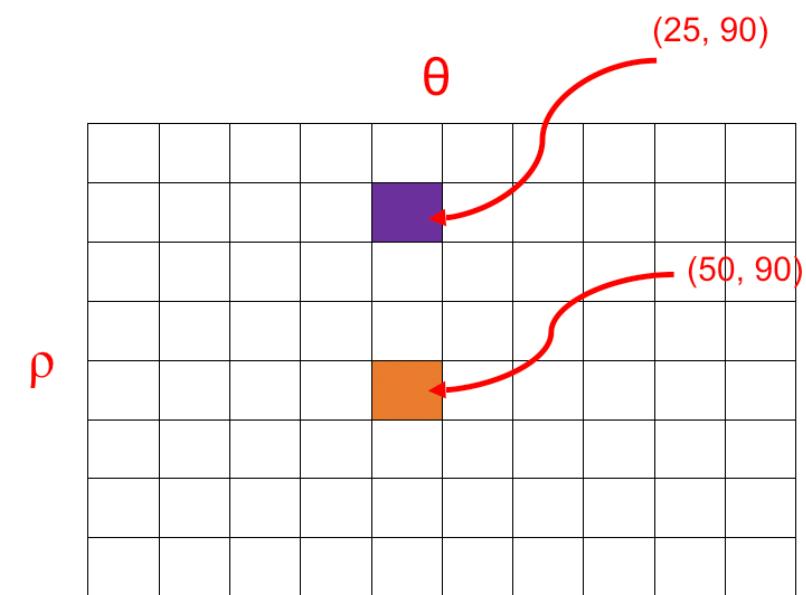
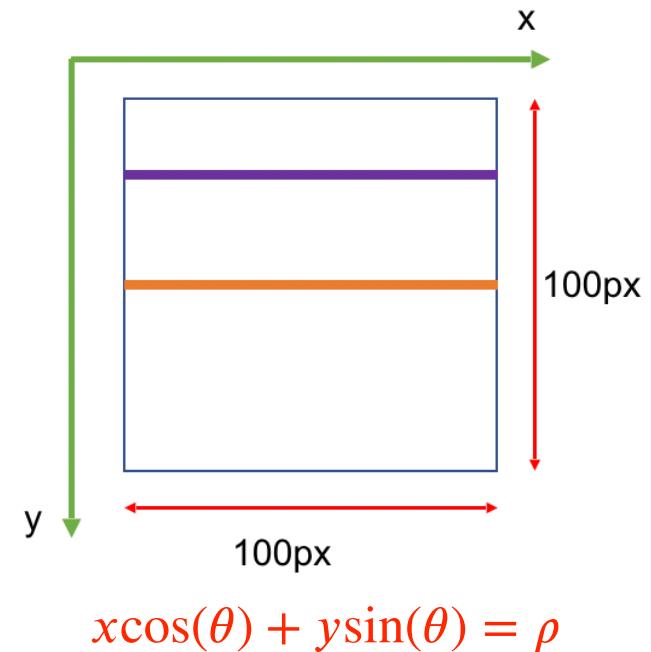
*Полярная система координат*

*Пространство параметров  
(пространство Хафа):*  
каждое уравнение прямой  
представляется одной точкой

# Преобразование Хафа

Иллюстрация алгоритма:

1. Имеется изображение размером 100x100 с горизонтальной линией
2. Создаем 2D таблицу-аккумулятор ( $\rho$ ,  $\theta$ ), каждой ячейке присваиваем значение 0.
3. Выбираем первую точку на линии и перебираем значения  $\theta$  в диапазоне 0, 1, 2, ... 180.
4. Для каждой полученной пары значений  $(\rho, \theta)$  увеличиваем соответствующее значение в ячейке аккумулятора на 1.
5. Выбираем следующую точку и повторяем шаги 3-4.



# Преобразование Хафа

Для вычисления преобразования Хафа в OpenCV есть функция  
**cv2.HoughLines()**

***lines = cv2.HoughLinesP(image, ρ accuracy, θ accuracy, threshold)***

Пример:

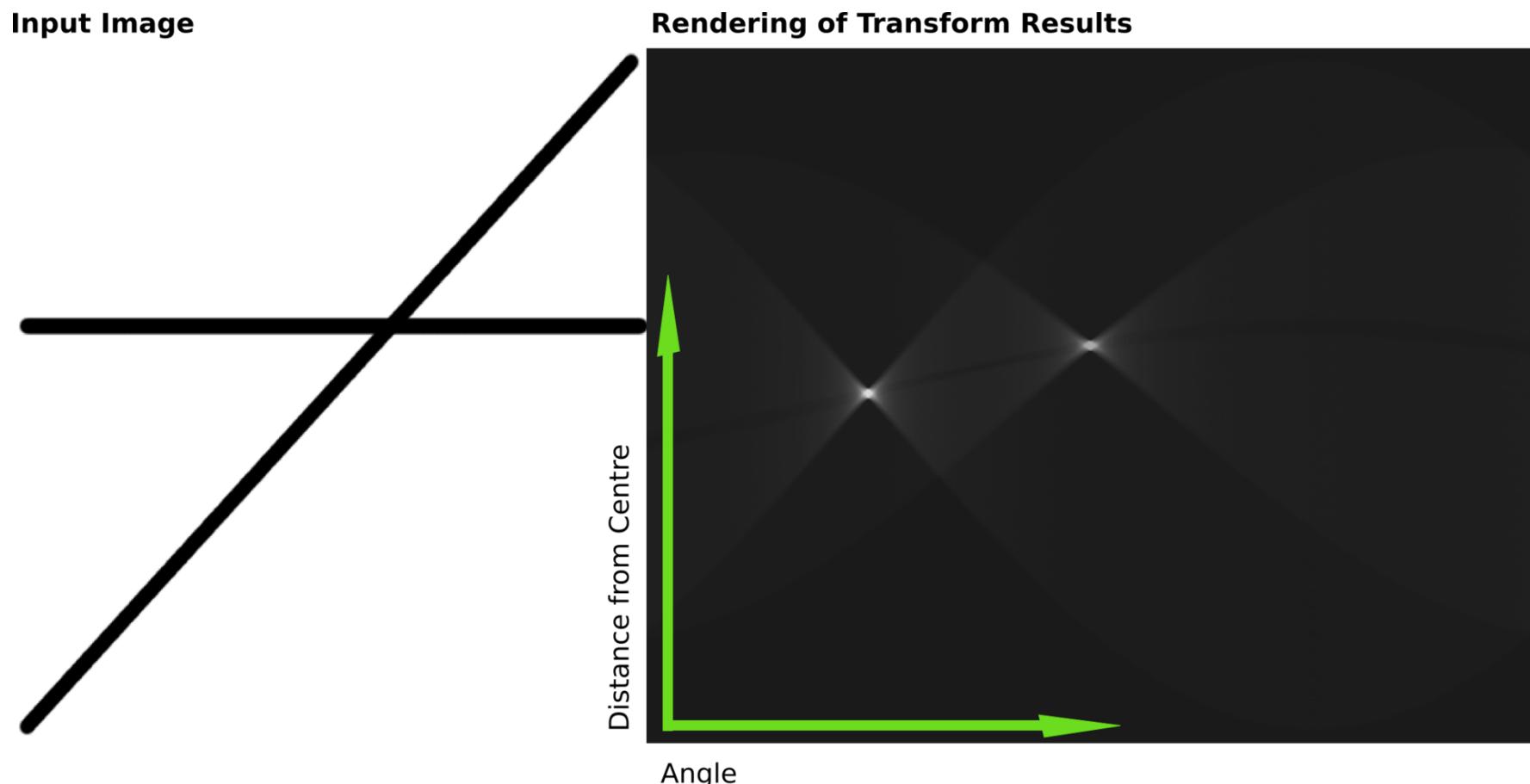
***lines = cv2.HoughLinesP(image, 1, np.pi / 180, 240)***

Основные параметры:

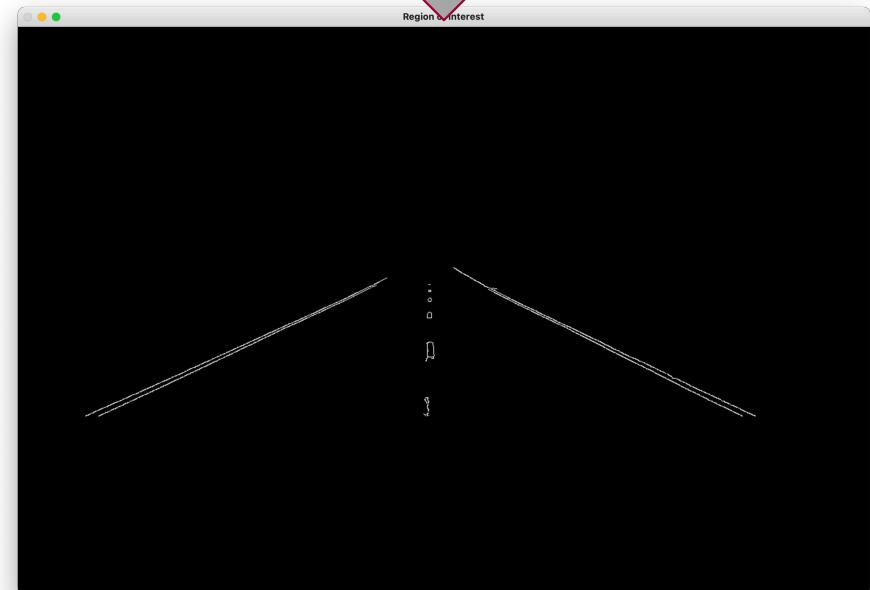
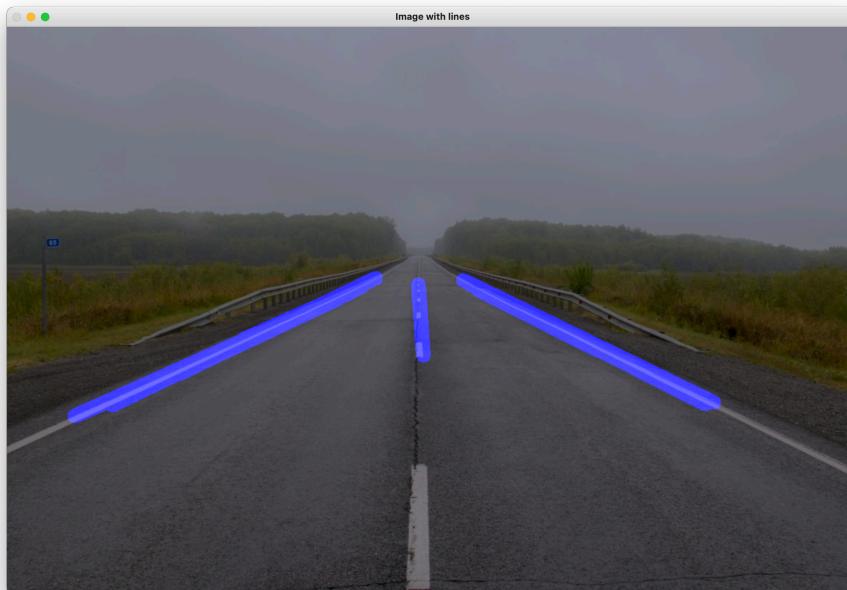
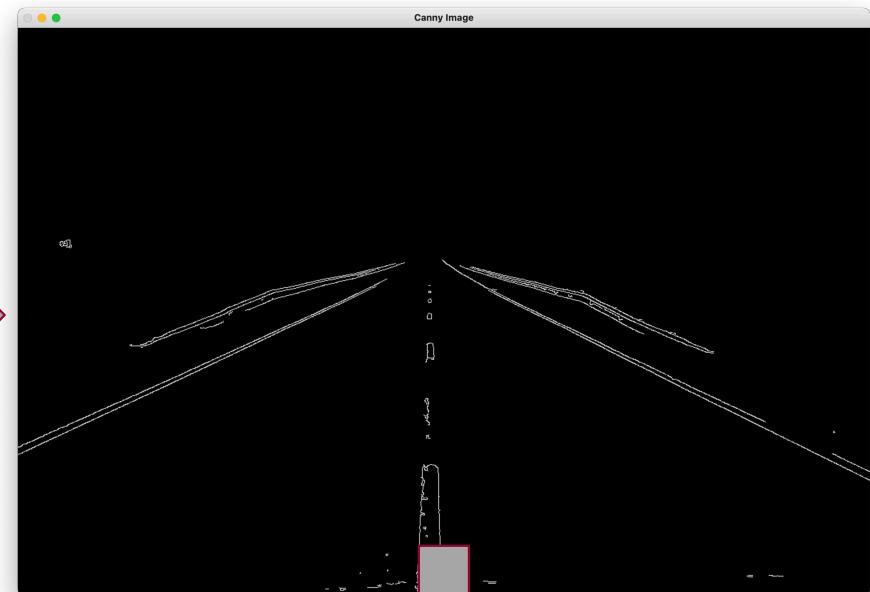
- **lines** – вектор, содержащий параметры ( $\rho$ ,  $\theta$ ) для обнаруженных прямых
- $\rho$  **accuracy** – разрешение параметра  $\rho$  в пикселях
- $\theta$  **accuracy** – разрешение параметра  $\theta$  в радианах
- **threshold** – минимальное количество пересечений («голосов») в таблице-аккумуляторе для обнаружения линии

# Преобразование Хафа

Результат преобразования Хафа для изображения с двумя пересекающимися прямыми



# Преобразование Хафа



# Преобразование Хафа

Пример 1 из газодинамики

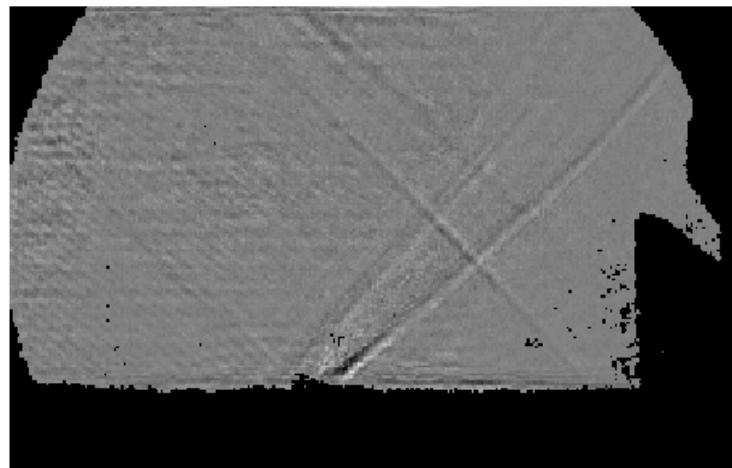


Видеозапись течения, полученная теневым методом: развитие течения за фронтом ударной волны в прямоугольном канале

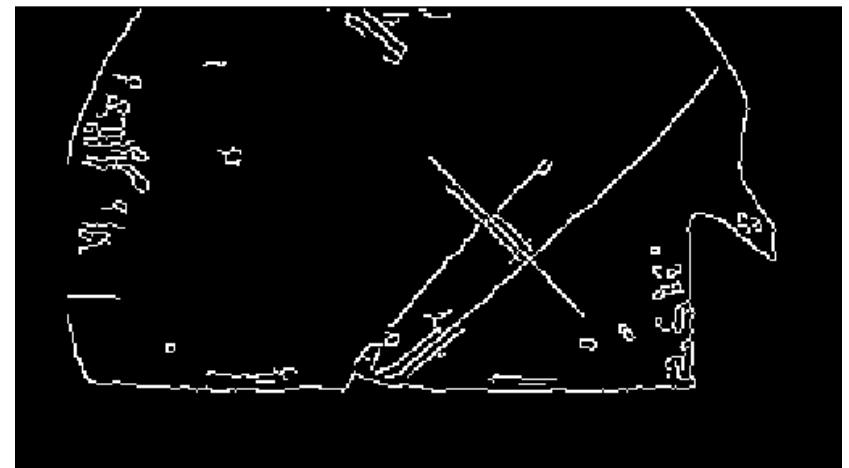
# Преобразование Хафа

Пример 1 из газодинамики

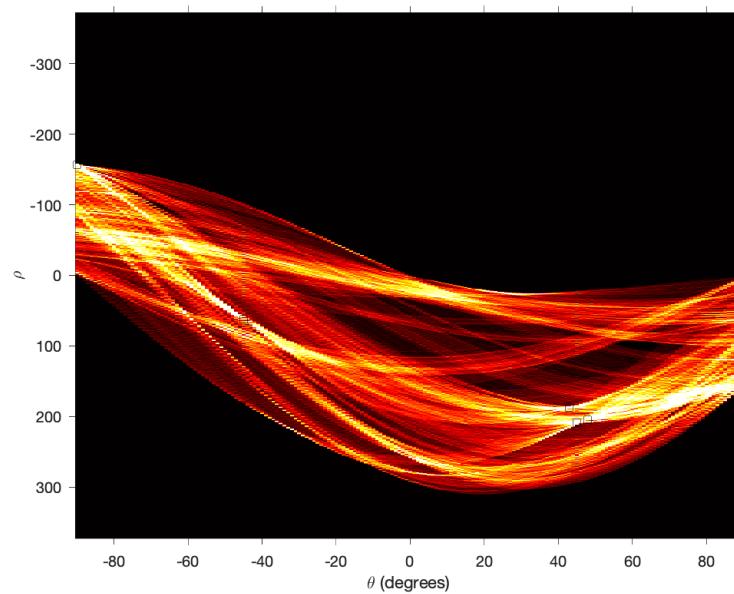
1



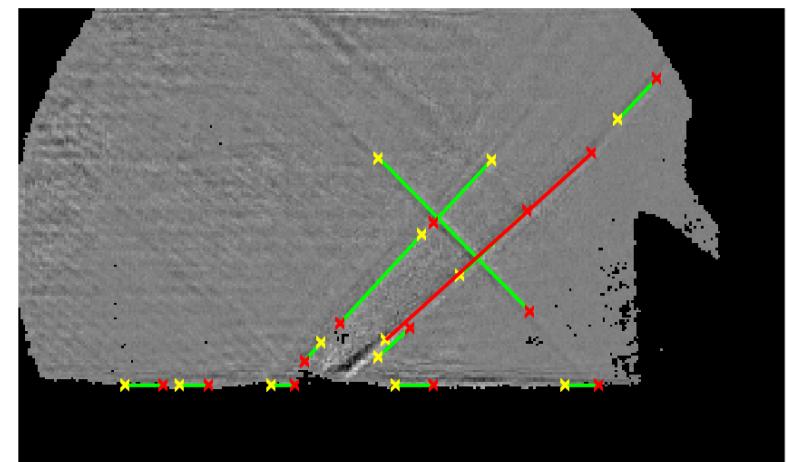
2



3

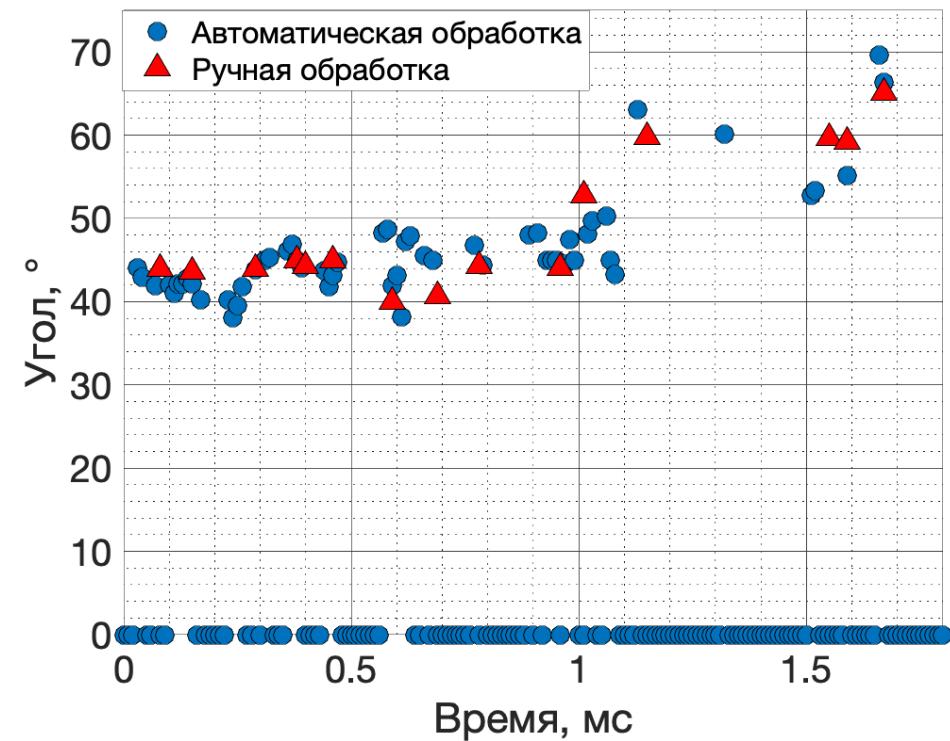
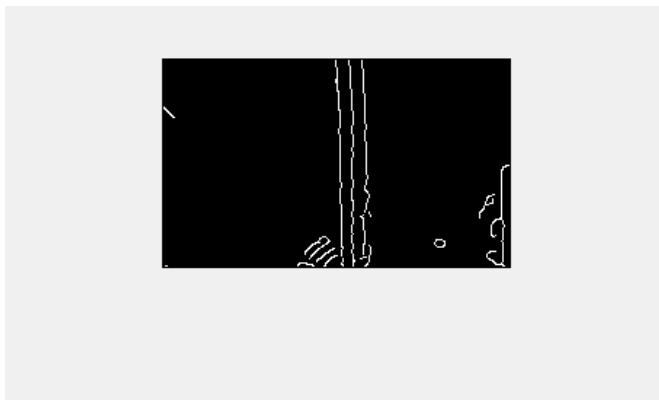
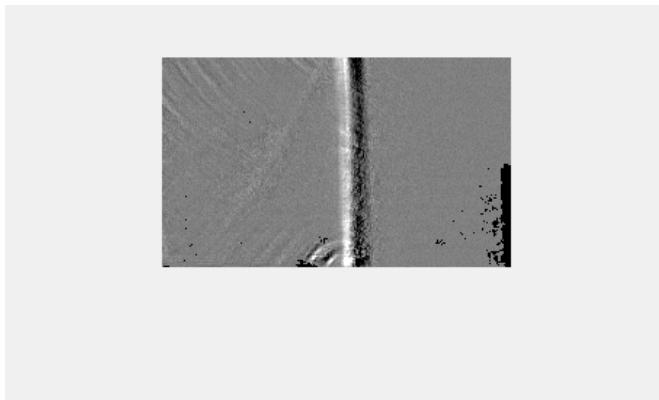


4



# Преобразование Хафа

Пример 2 из газодинамики: автоматическое определение угла наклона косого скачка уплотнения

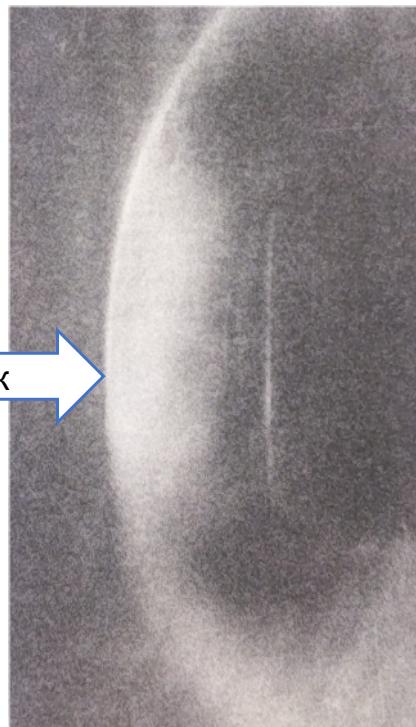


Течение становится дозвуковым при достижении  $90^\circ$  через 2,5 – 3,5 мс

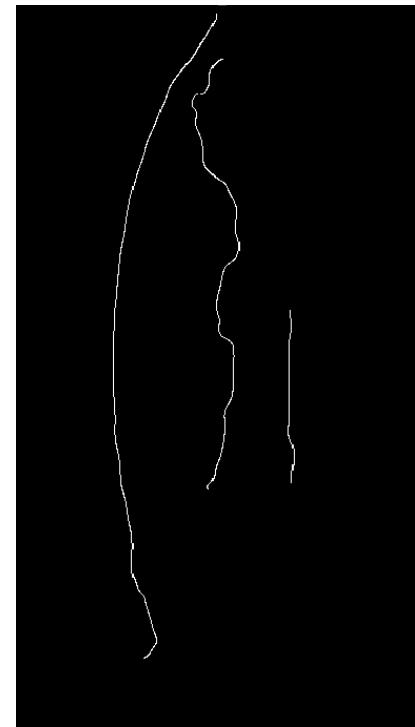
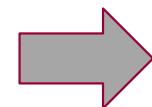
# Преобразование Хафа

Пример 3 из газодинамики:

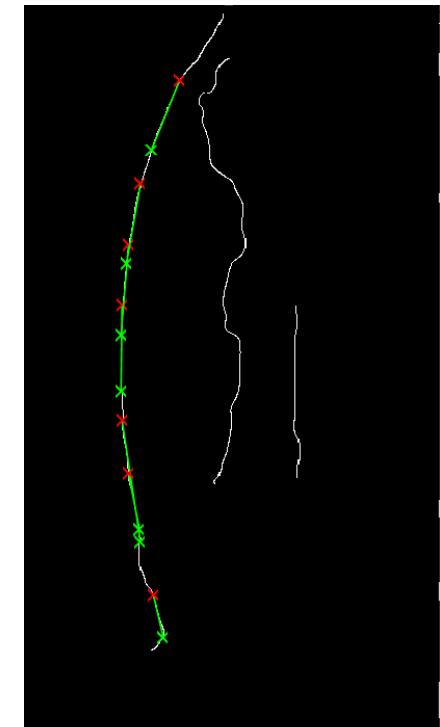
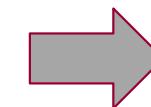
Выделение прямых сегментов головной ударной волны на шлирен-изображениях



Edge  
detection



Hough  
transform

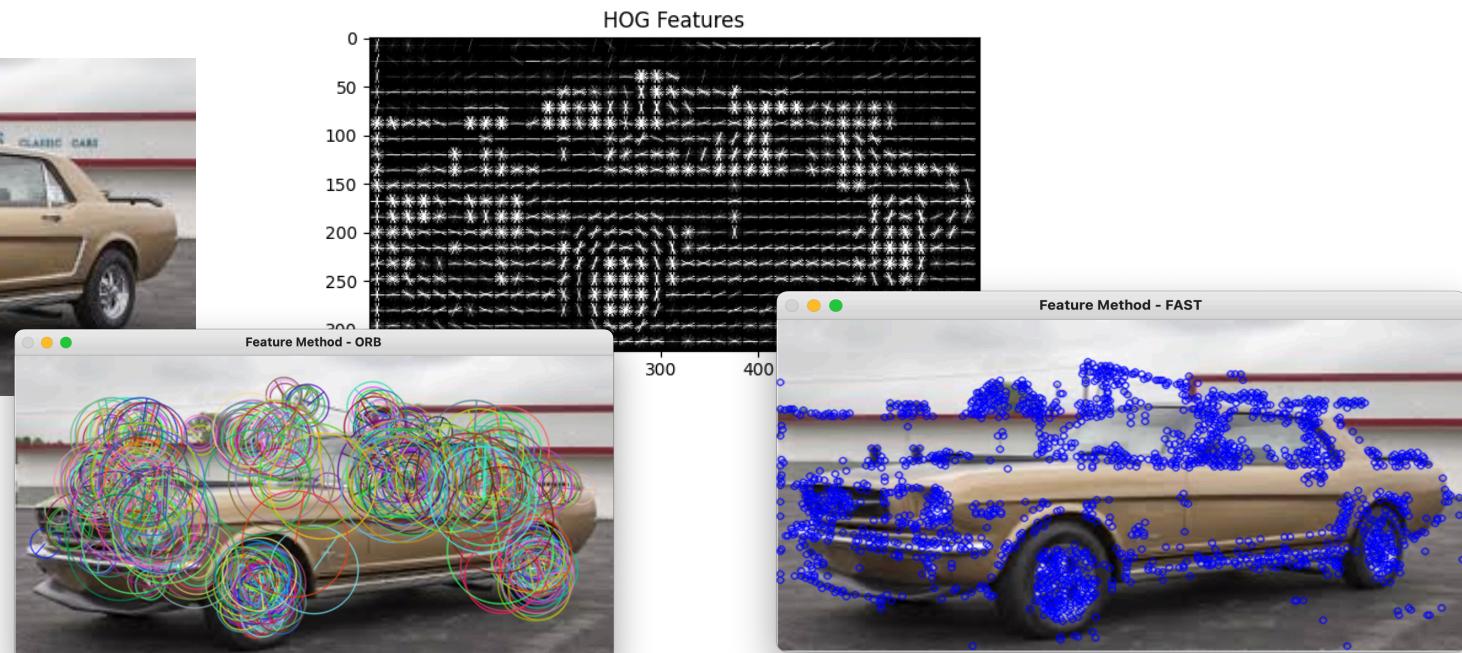


## Выявление признаков (Feature detection)

# Признаки изображений (image features)

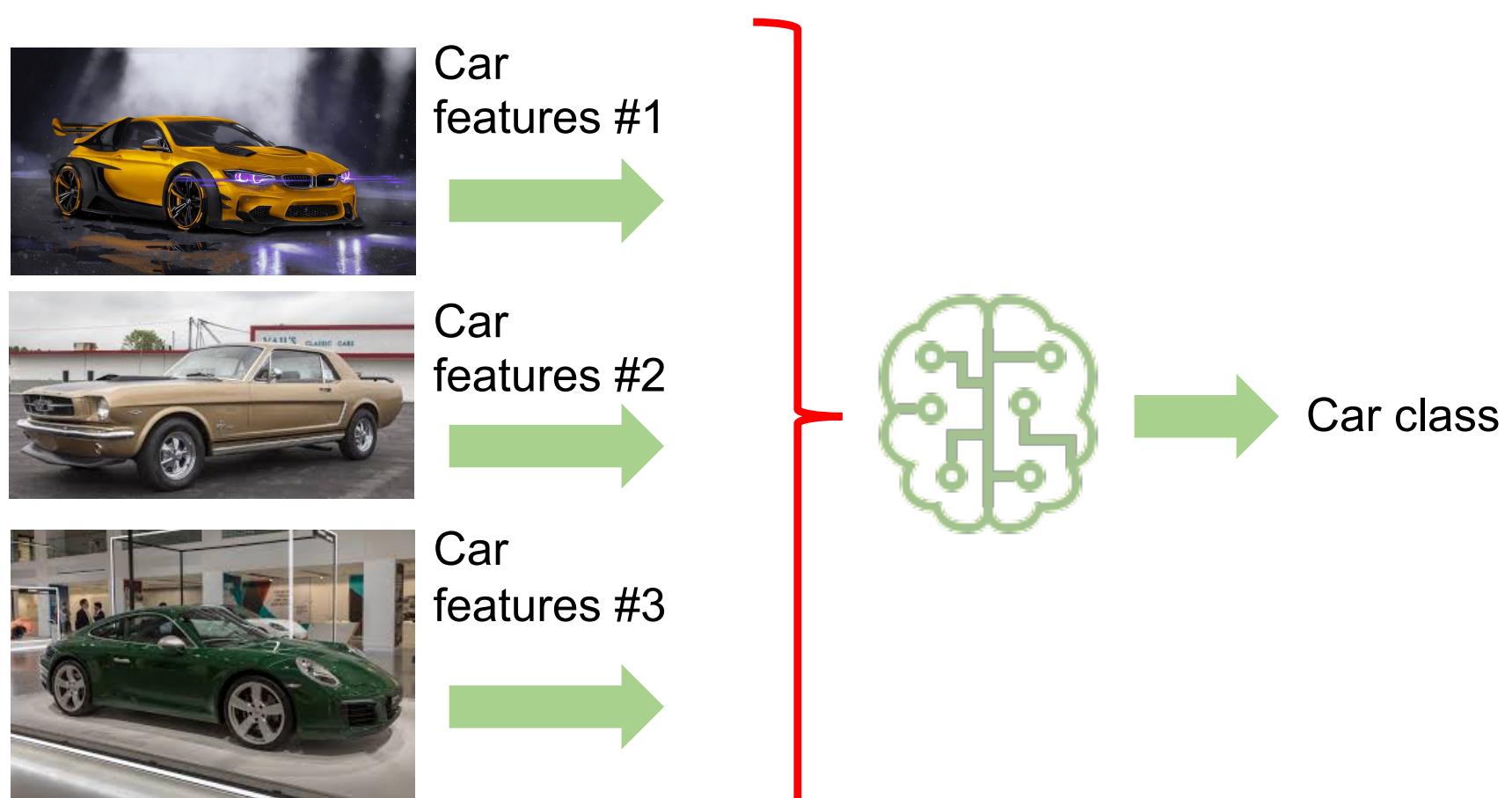
**Признаки изображений** (Image Features) – участки изображений, которые уникальны для данного изображения.

- Признаки могут быть представлены цветами, границами, группами пикселей.
- Признаки должны быть воспроизводимыми, т. е. одни и те же признаки должны находиться на двух или более изображениях одной и той же сцены.



# Признаки изображений (image features)

- Признаки изображений крайне важны для решения задач машинного обучения, так как они используются для анализа, описания и сравнения изображений.
- Признаки могут использоваться для обучения классификаторов для обнаружения различных объектов: людей, автомобилей и т.д.

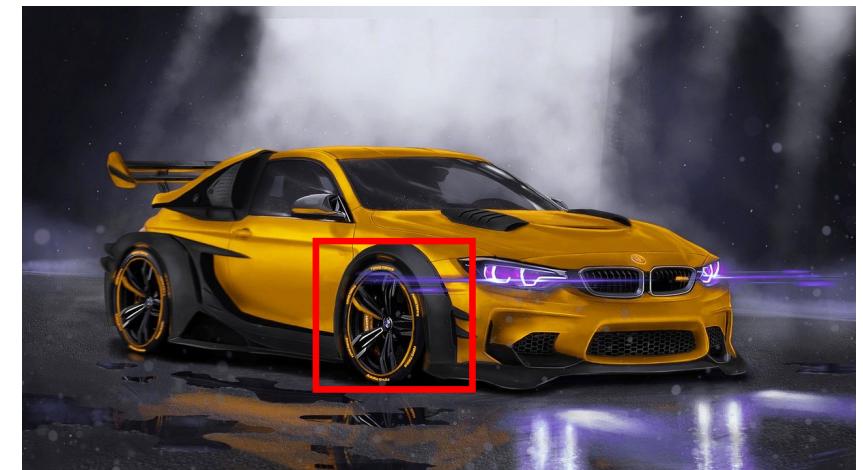
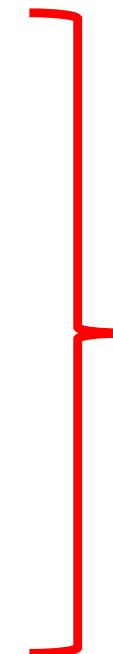


# Сравнение шаблонов

**Сравнение шаблонов** (Template matching) – техника, применяемая в цифровой обработке изображений для поиска небольших составных частей изображений на целом изображении

В OpenCV для решения этой задачи есть функции:  
**cv2.matchTemplate()**, **cv2.minMaxLoc()**

template



# Сравнение шаблонов

- **cv2.matchTemplate()**

проходит по всему изображению  $I$  размером  $(W, H)$ , сравнивает выбранные участки с шаблоном  $T$  размером  $(w, h)$  и создает результирующее изображение  $R$  размером  $(W-w+1, H-h+1)$ . Изображение  $R$  – результат выбранного преобразования, например, нормированной кросс-корреляции (cv2.TM\_CCOEFF\_NORMED).

```
result = cv2.matchTemplate(image_gray, template, cv2.TM_CCOEFF_NORMED)
```

$$R(x, y) = \frac{\sum_{x',y'} (T'(x', y') \cdot I'(x + x', y + y'))}{\sqrt{\sum_{x',y'} T'(x', y')^2 \cdot \sum_{x',y'} I'(x + x', y + y')^2}}$$

# Сравнение шаблонов



Левый верхний угол шаблона



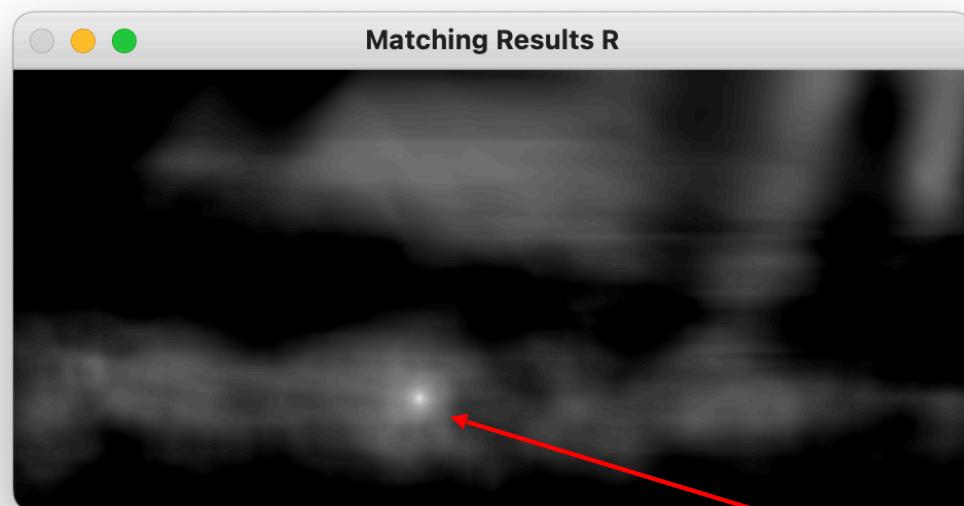
- **cv2.minMaxLoc()**

Далее, используя функцию **cv2.minMaxLoc()** по изображению R нужно найти максимальное значение, которое соответствуют левому верхнему углу искомого шаблона Т на изображении I.

# Сравнение шаблонов

- **cv2.minMaxLoc()**

Используя размеры шаблона  $w$  и  $h$  можно нарисовать соответствующую область на исходном изображении  $I$ , прибавив эти значения к координатам найденного левого верхнего угла.



Левый верхний угол

# Сравнение шаблонов

## Ограничения данного подхода:

- Ориентация шаблона должна соответствовать ориентации исходного изображения
- Точность значительно ухудшается при несоответствии размеров исходного изображения и шаблона
- Точность ухудшается при различной яркости, контрастности изображения и шаблона
- Разные ракурсы съемки также ухудшают результаты



# Сравнение шаблонов

Пример применения в физическом эксперименте:

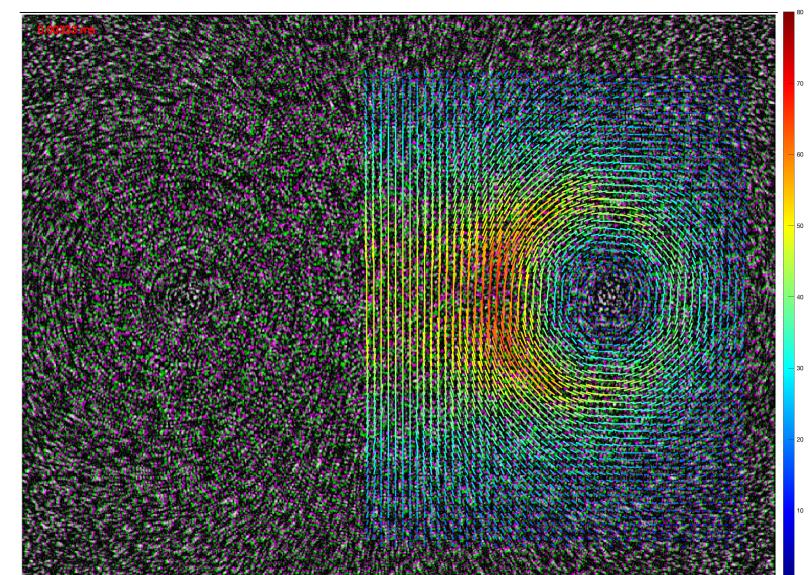
- ▶ **PIV (Particle Image Velocimetry)** – цифровая анемометрия по изображениям частиц, взвешенных в потоке
- ▶ **SIV (Schlieren Image Velocimetry)** – беззасевная цифровая анемометрия по изображениям структур течений (неоднородностей потока, погран. слоя, вихрей и др.)

*Совмещенные изображения;  
часть изображения обработана методом  
кросс-корреляции, по смещению частиц  
рассчитаны скорости*

$t_1$



$t_2 > t_1$



# Морфологические преобразования

# Морфологические преобразования

**Морфологические преобразования** – широкий набор методов обработки изображений, основанный на геометрических структурных элементах.

Структурный элемент применяется к обрабатываемому изображению, создавая новое изображение того же размера, что и исходное.

При этом значение каждого пикселя вычисляется на основе его собственного значения и значений соседних пикселей.

Количество задействованных в расчете пикселей задается структурным элементом.

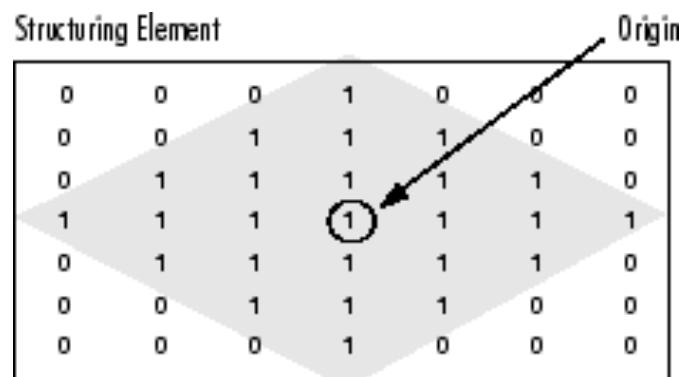
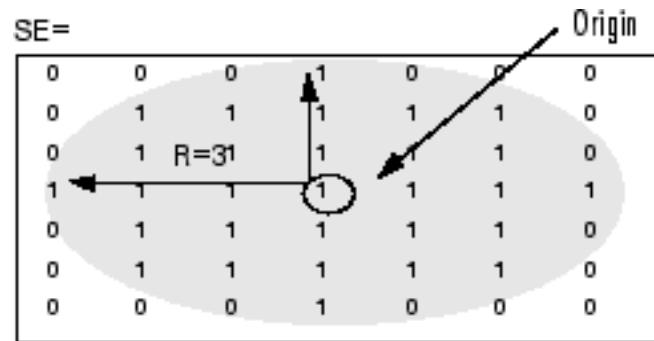
Обычно применяются к бинарным или монохромным изображениям.

Наиболее часто встречающиеся операции:

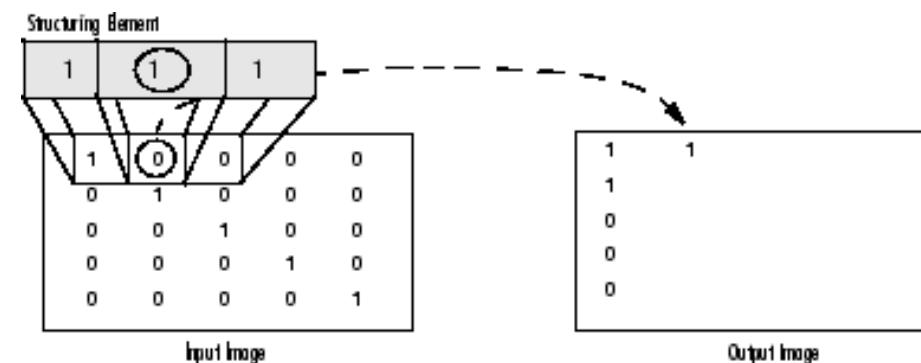
- эрозия (erosion)
- дилатация/наращивание (dilatation)

# Морфологические преобразования

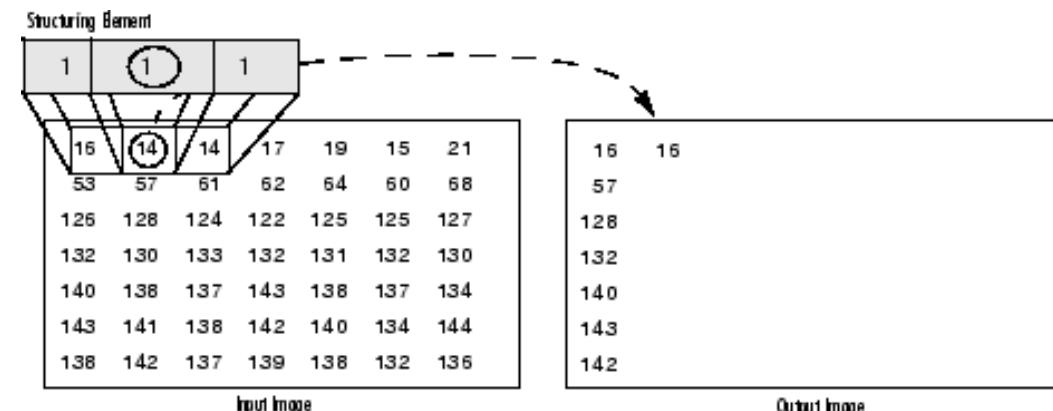
Примеры структурных элементов:



Пример операции dilation для бинарных изображений



Пример операции dilation для монохромных изображений



# Морфологические преобразования



Исходное изображение



Операция «**Erosion**» - матрица, содержащая структурный элемент, перемещается по каждому пикслю изображения (по аналогии с операцией свёртки) и каждому пикслю присваивается 1 только если **все** пиксели в окрестности (внутри структурного элемента) равны 1.



Операция «**Dilation**» - матрица, содержащая структурный элемент, перемещается по каждому пикслю изображения (по аналогии с операцией свёртки) и каждому пикслю присваивается 1 только **если хотя бы 1** пиксель в окрестности (внутри структурного элемента) равен 1.

# Морфологические преобразования



Операция «**Opening**» - последовательное применение операций «*erosion*», затем «*dilation*». Первая операция позволяет удалить шум – малые объекты на изображении, а вторая – восстановить первоначальные размеры крупных объектов, не являющихся шумом

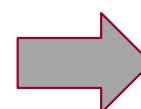


Операция «**Closing**» - последовательное применение операций «*dilation*», затем «*erosion*». Это операция, обратная opening. Позволяет заполнять полости в изображении объекта.

Выделение углов (corner detection)

# Выделение углов (corner detection)

- Выделение границ – интенсивность значительно меняется в одном направлении
- Выделение углов – интенсивность значительно меняется при движении в любом направлении от рассматриваемой точки



# Выделение углов (corner detection)

Один из наиболее распространенных методов – ***Harris corner detection***.

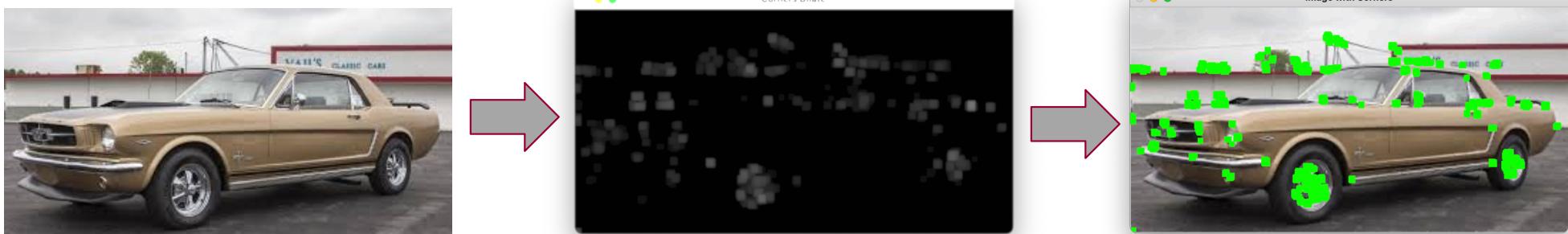
Он основан на смещении окна опроса относительно каждого пикселя во всех направлениях и вычислении разности в интенсивности.

В OpenCV для выделения углов есть функция  
**`cv2.cornerHarris (img, blockSize, ksize, k)`**:

- *img* – исходное изображение (монохромное)
- *blockSize* – размер окна опроса
- *ksize* – параметр апертуры для оператора Собеля
- *k* – свободный параметр в уравнении (0.1 по умолчанию)

# Выделение углов (corner detection)

- В отличие от метода сравнения шаблонов, выделение углов работает, даже если изображение повернуто, изменена его яркость
- Метод выделения углов плохо работает при масштабировании изображений



## Гистограммы направленных градиентов

# Гистограммы направленных градиентов

**Гистограмма направленных градиентов** (Histogram of oriented gradients, HOG):

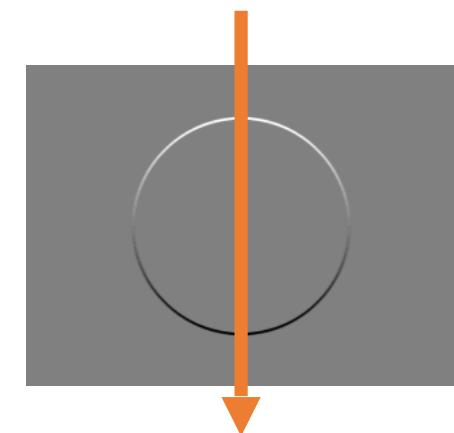
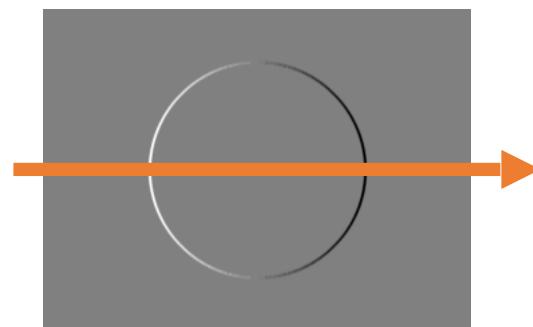
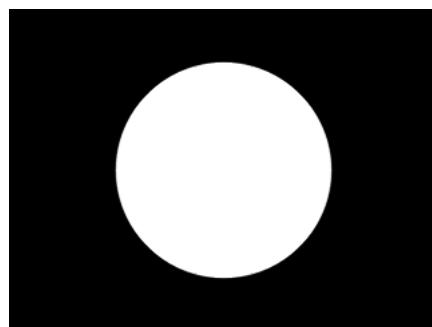
- для функции  $f(x, y)$  градиентом обозначим вектор  $(f_x, f_y)$
- поскольку  $f(x, y)$  – дискретная функция, для расчета градиентов переходим от дифференцирования к разности, по аналогии с операцией свертки:

-1	0	1
----	---	---

$f_x$  gradient kernel

-1
0
1

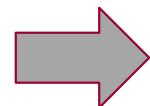
$f_y$  gradient kernel



# Гистограммы направленных градиентов

элемент изображения

1	2	10	1	39
3	12	2	4	7
37	1	25	1	1
0	2	19	65	21
98	41	78	10	2



$$f_x = |21 - 19| = 2$$

$$f_y = |10 - 1| = 9$$

$$|f| = \sqrt{2^2 + 9^2} = 9.7$$

$$\alpha = \arctan\left(\frac{9}{2}\right) = 77^\circ$$

направление  
градиента

величина  
градиента

# Гистограммы направленных градиентов

элемент изображения

1	2	10	1	39
3	12	2	4	7
37	1	25	1	1
0	2	19	↗	21
98	41	78	10	2

$$\alpha = \arctan\left(\frac{9}{2}\right) = 77^\circ$$

$$0 \leq \alpha \leq 180^\circ$$

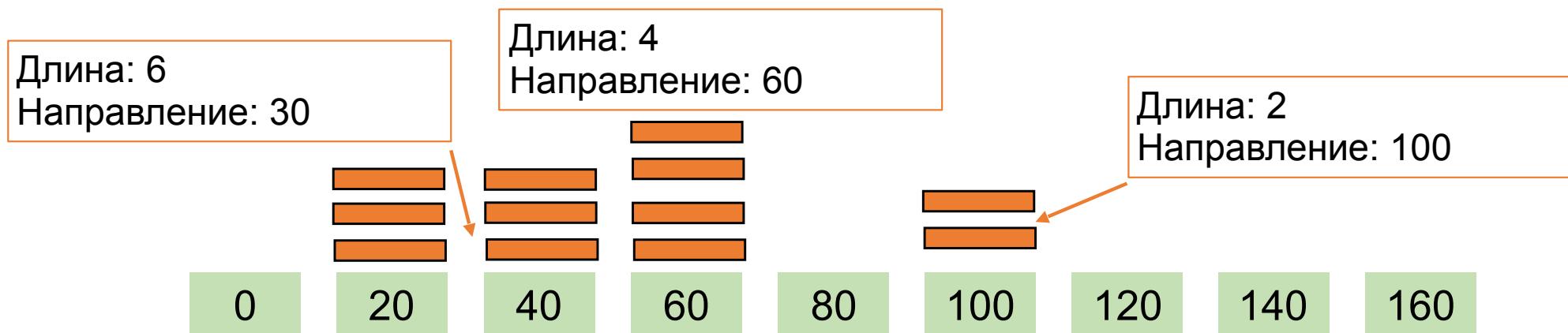
Изображение разделяется на блоки размером (как правило) 8x8 пикселей. Гистограмма строится для каждого такого блока



# Гистограммы направленных градиентов

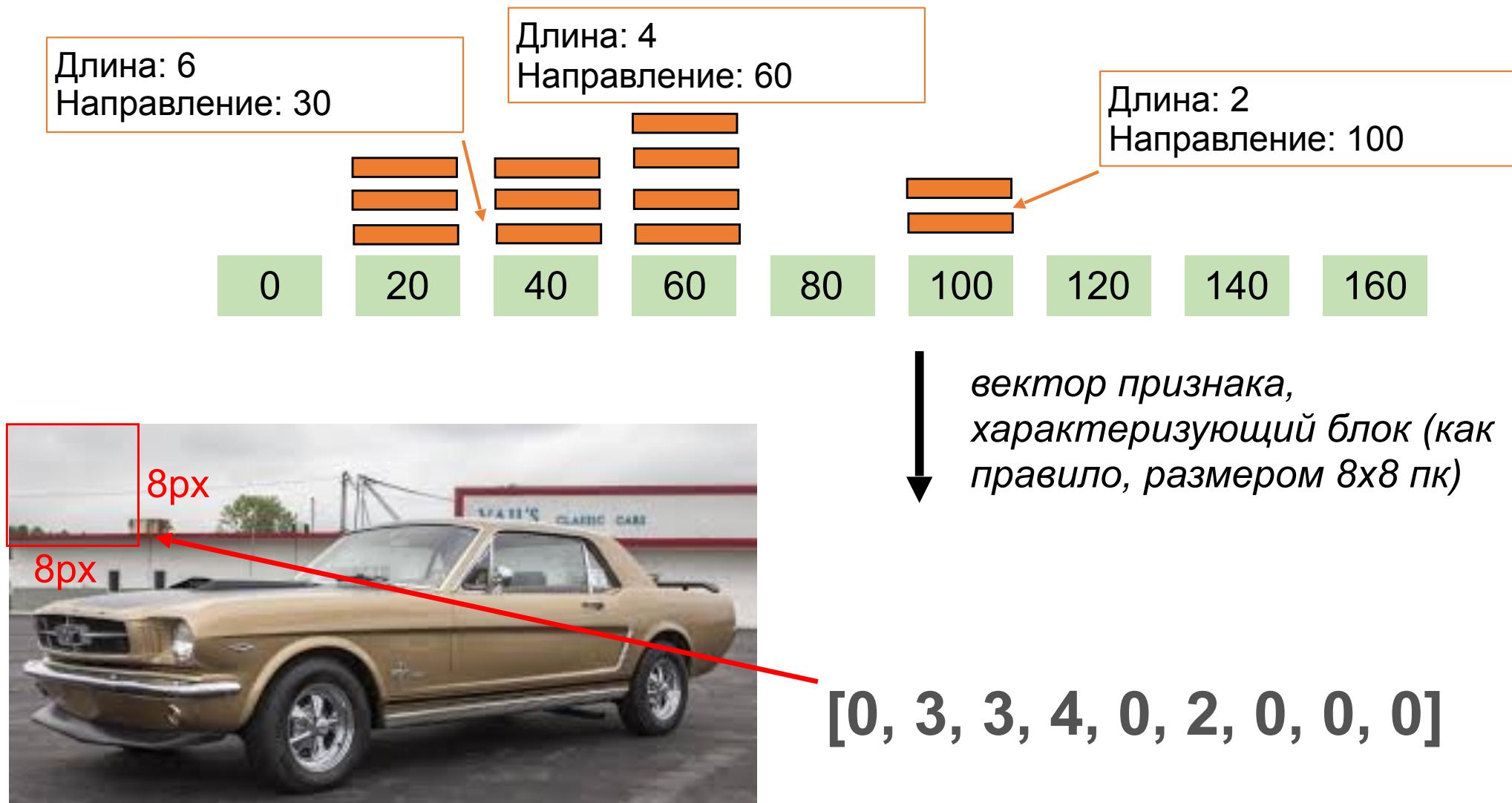
1	2	10	1	39
3	12	2	4	7
37	1	25	1	1
0	2	19	/	21
98	41	78	10	2

- Необходимо составить **гистограмму направленных градиентов**. Получится некоторое распределение углов и длин векторов, которое характеризует изображение.
- Если  $\alpha > 180^\circ$ , то угол преобразуется в диапазон  $[0, 180]$ . Напр.,  $185^\circ$  — в  $5^\circ$



- Для каждого градиента в соответствующую его углу ячейку добавляется величина его длины
- Если угол попадает в промежуток между ячейками, его длина делится пропорционально между двумя ближайшими ячейками

# Гистограммы направленных градиентов



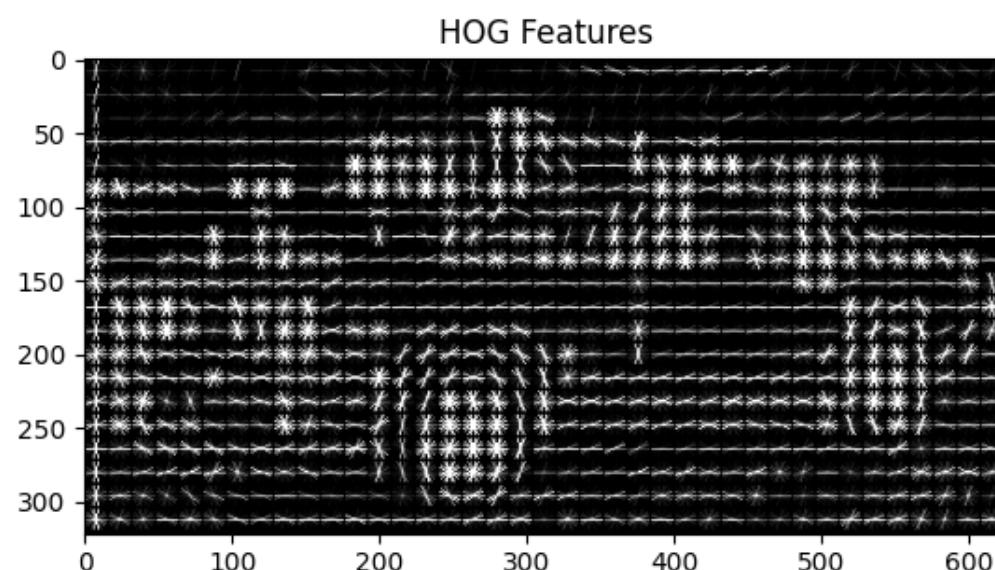
# Гистограммы направленных градиентов



- Далее для каждой комбинации из 4 блоков производится нормализация векторов признаков, чтобы сделать полученные векторы менее зависимыми от изменения освещенности

```
features, hog_image = hog(image_gray,
```

```
orientations = 9,  
pixels_per_cell = (16, 16),  
cells_per_block = (1, 1),  
transform_sqrt = False,  
visualize = True,  
feature_vector = False)
```



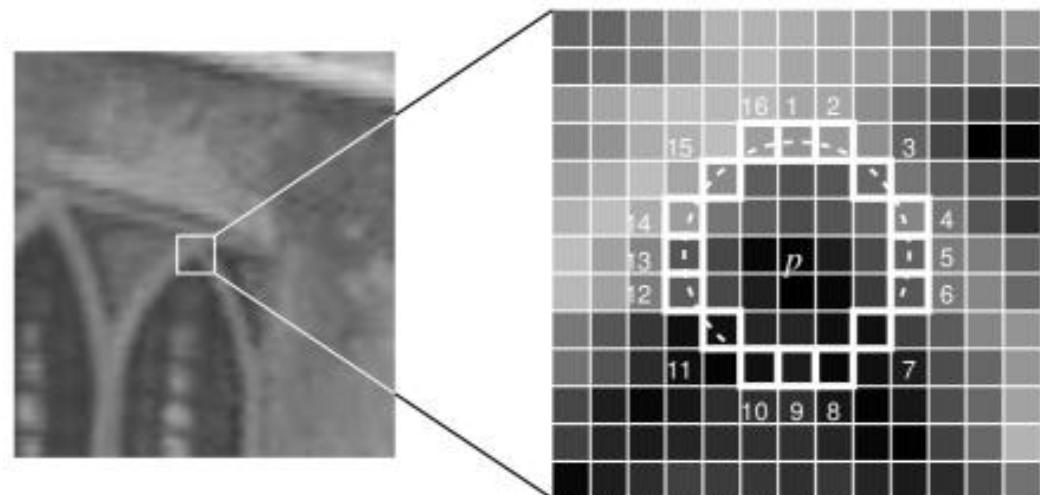
## FAST/SURF/ORB/SIFT Feature Detectors

# FAST/SURF/ORB/SIFT Feature Detectors

**Особые точки** (feature points) - точки изображения, локальные окрестности которых обладают некоторыми отличительными особенностями в сравнении с окрестностями остальных точек изображения: края, углы, «блобы», рёбра и пр.

- Алгоритм **FAST** (**F**eatures from **A**ccelerated **S**egment **T**est) был предложен в работе

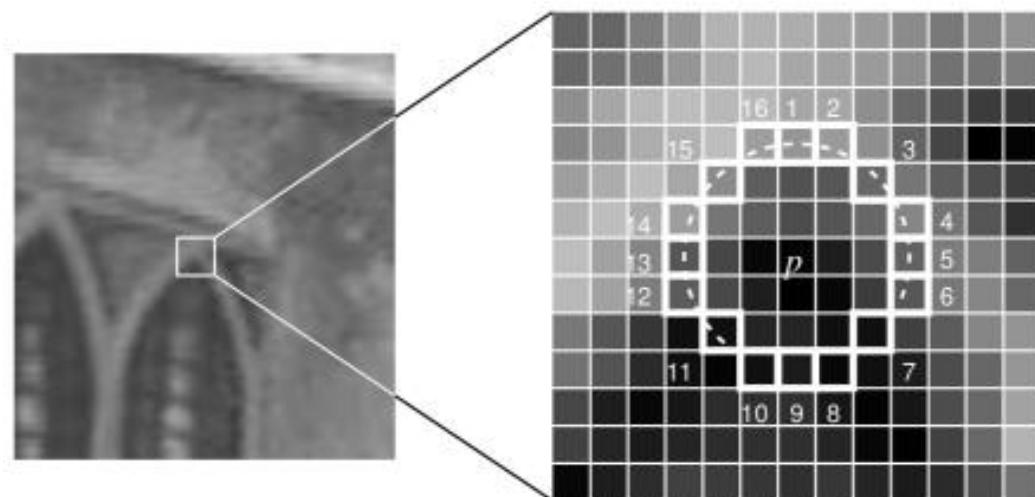
*Rosten E., Drummond T. (2006)  
Machine Learning for High-Speed  
Corner Detection. In: Leonardis A.,  
Bischof H., Pinz A. (eds) Computer  
Vision – ECCV 2006. ECCV 2006.  
Lecture Notes in Computer Science, vol  
3951.*



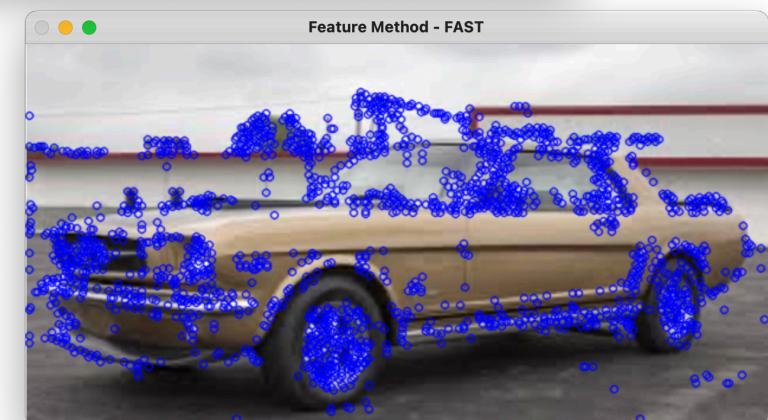
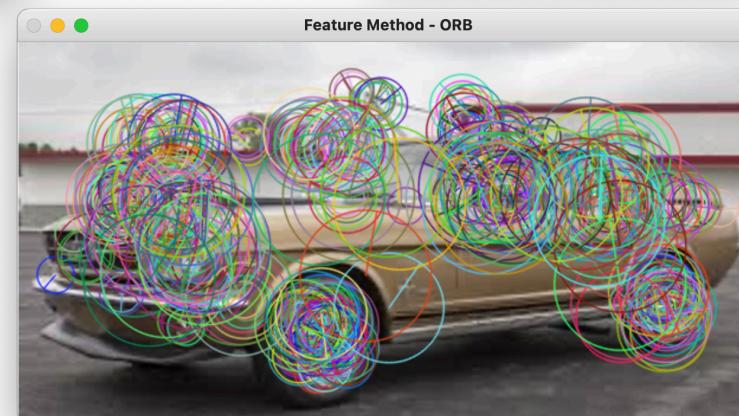
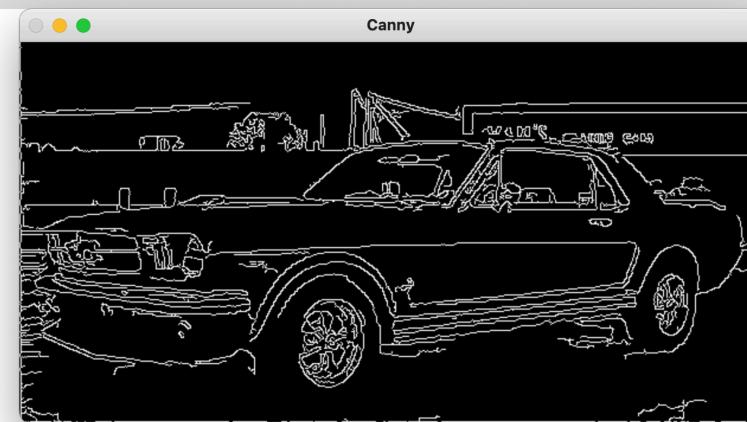
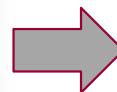
# FAST/SURF/ORB/SIFT Feature Detectors

- Алгоритм **FAST** (**F**eatures from **A**ccelerated **S**egment **T**est)
  1. Выбор пикселя на изображении р интенсивностью  $I_p$
  2. Выбор порогового значения  $t$
  3. Рассмотрим окружность радиусом 16 пикселей с центром  $p$
  4. Пиксель  $p$  будет считаться углом, если существует набор смежных пикселей, лежащих на окружности, каждый из которых ярче, чем  $I_p + t$  или темнее, чем  $I_p - t$

В упрощенном варианте рассматриваются только пиксели 1, 9, 5, 13.  
 $p$  будет считаться углом, если 3 из них ярче, чем  $I_p + t$  или темнее, чем  $I_p - t$ .



# FAST/SURF/ORB/SIFT Feature Detectors



## Кластеризация методом k-средних

# Кластеризация методом k-средних

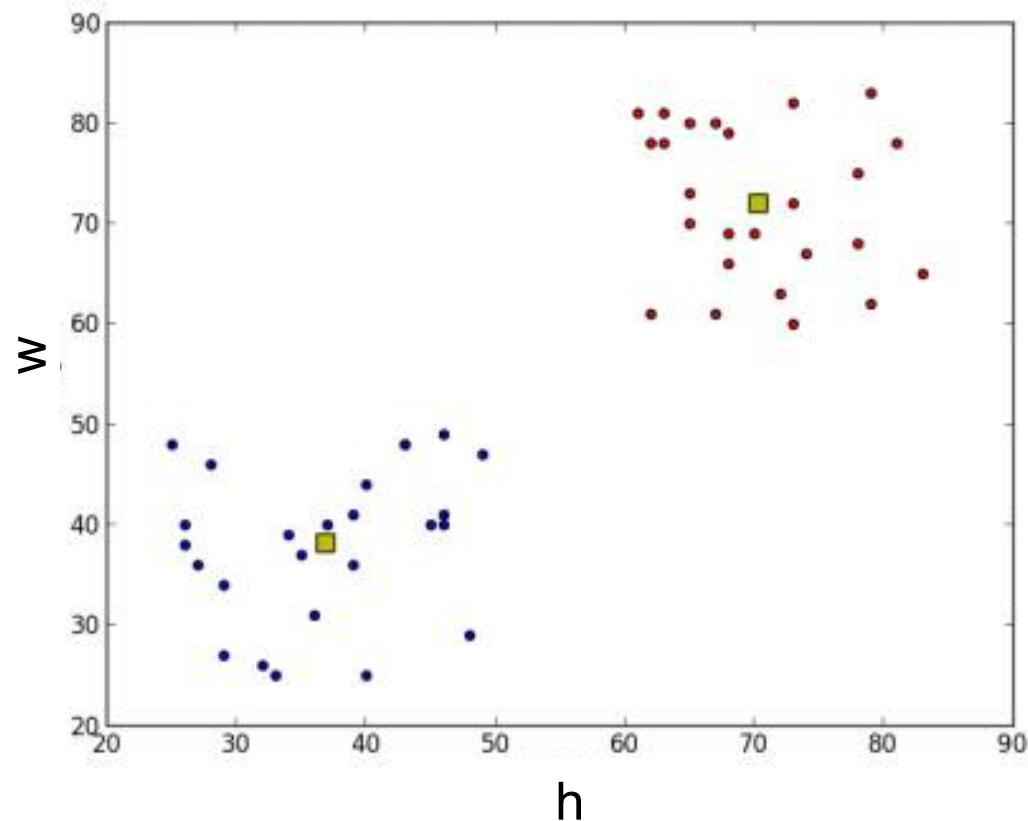
**Кластеризация методом k-средних (k-Means Clustering):**

разбивает множество элементов векторного пространства на заранее известное число кластеров  $k$ .

Критерий сравнения объектов — *расстояние между ними*

Пример 1: группировка изображений на 2 группы по признаку высоты и ширины.

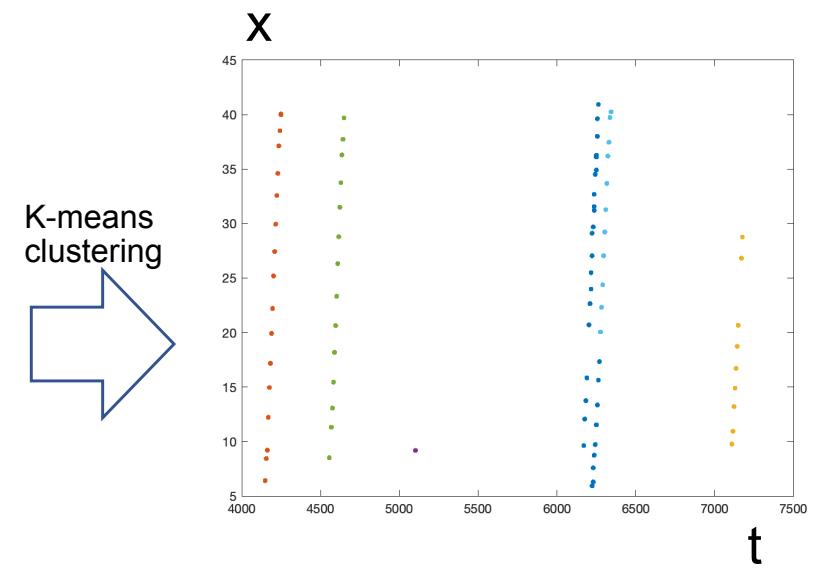
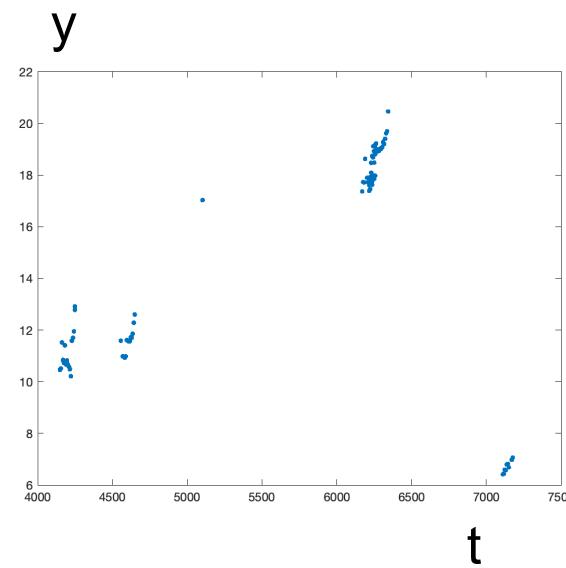
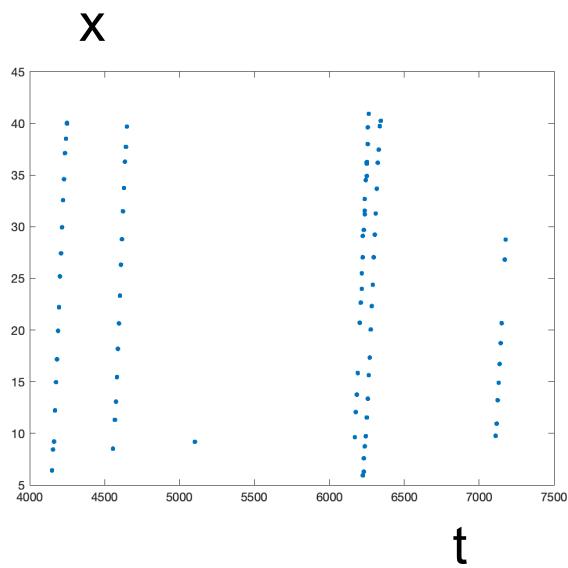
В данном случае расстояния вычисляются в 2D пространстве ( $w, h$ ) для  $k = 2$ . В результате работы алгоритма находятся точки в центре каждой группы.



# Кластеризация методом k-средних

Пример 2: пусть положения частиц заданы точками на плоскостях  $(x, t)$ ,  $(y, t)$ , но не сказано к какой группе принадлежат отдельные траектории. Для выделения отдельных траекторий можно использовать *кластеризацию*. Составляется матрица из характерных признаков частиц – время и координаты по  $x$  и  $y$ , выбирается  $k = 6$ . В результате алгоритм способен разделить траектории на 6 групп.

$$observation = \begin{pmatrix} t_1 & x_1 & y_1 \\ \dots & \dots & \dots \\ t_n & x_n & y_n \end{pmatrix}$$



# Кластеризация методом k-средних

Пример 3: кластеризация по цветам изображения.

Это способ сокращения количества цветов в изображении, что упрощает работу с ним с помощью программных методов. Признаками в данном случае являются цвета (R, G, B).

*Исходное изображение*



K = 2



K = 150



K = 20