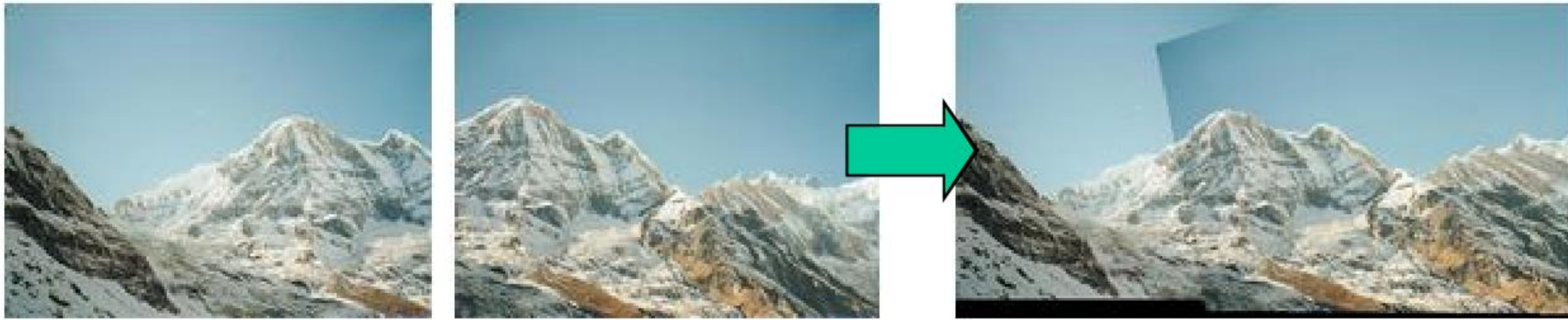


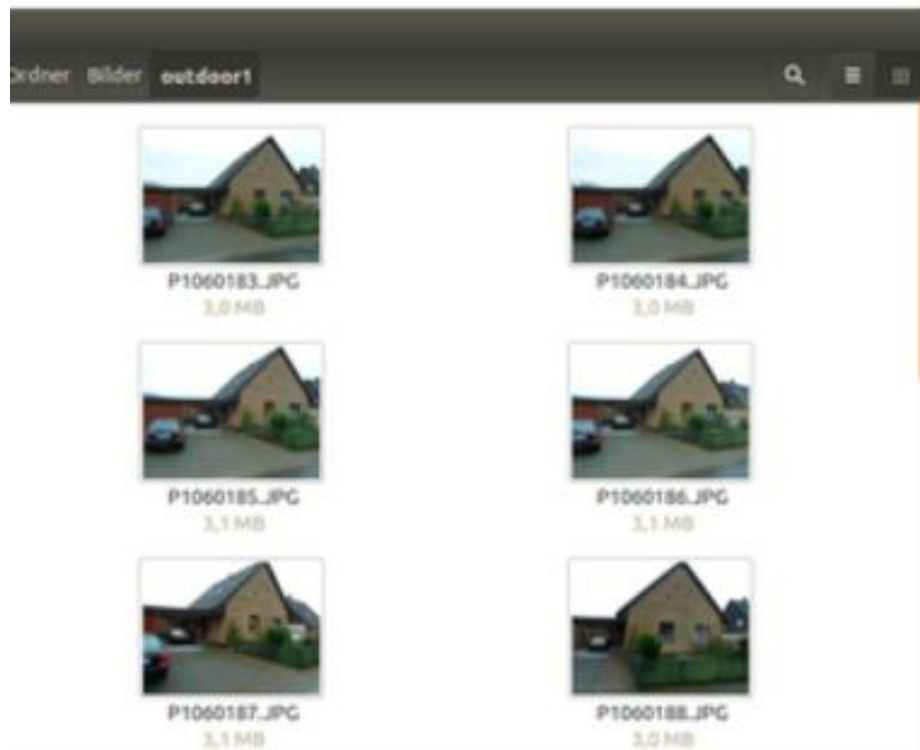
## Тема: Детектирование локальных особенностей



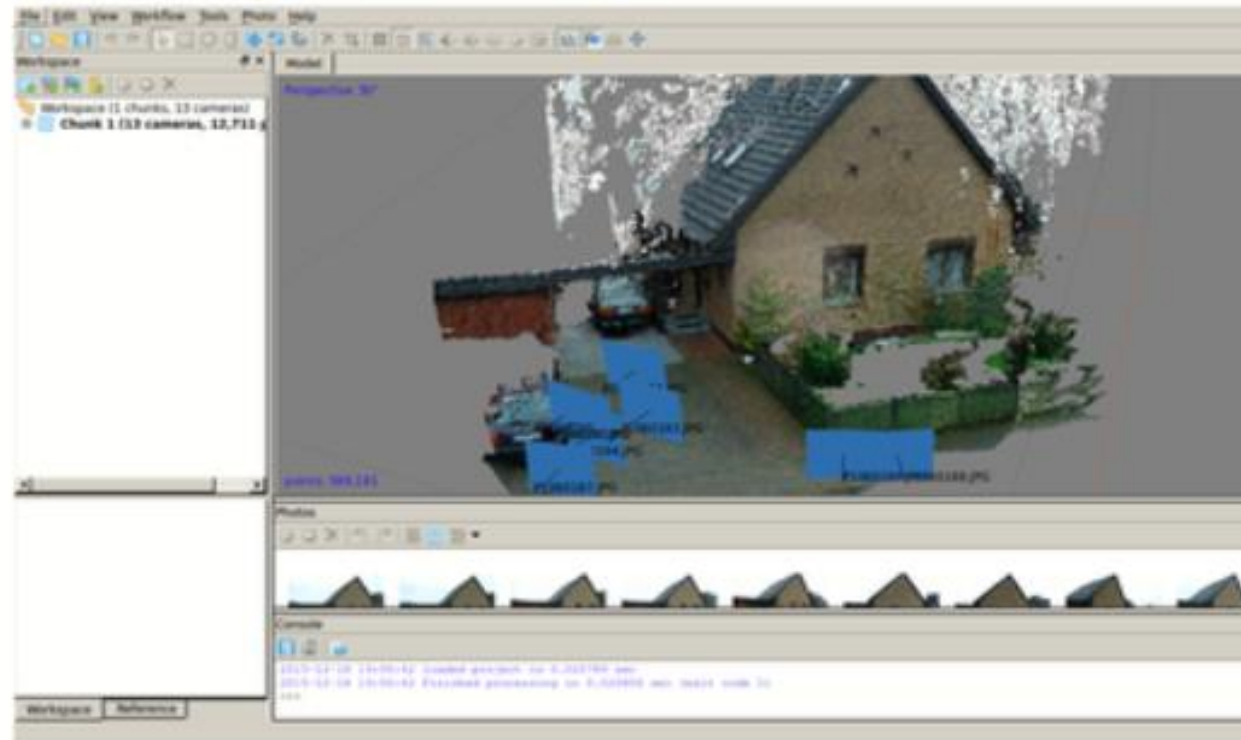
Сопоставление изображений является одним из фундаментальных аспектов многих задач компьютерного зрения, таких как:

- создание панорамных снимков,
- создание стереопары и реконструкция трехмерной модели объекта по его двумерным проекциям,
- распознавание объектов и поиск по образцу из какой-либо базы,
- слежение за движением объекта в видеопоследовательности и т.д.

# Реконструкция трехмерной модели объекта по его двумерным проекциям



Набор снимков CV-камер



Восстановление объекта по  
пространственному облаку точек

# **Первый подход к сопоставлению изображений**

Самый простой подход к сопоставлению изображений компьютером – это **попиксельное сравнение изображений**.

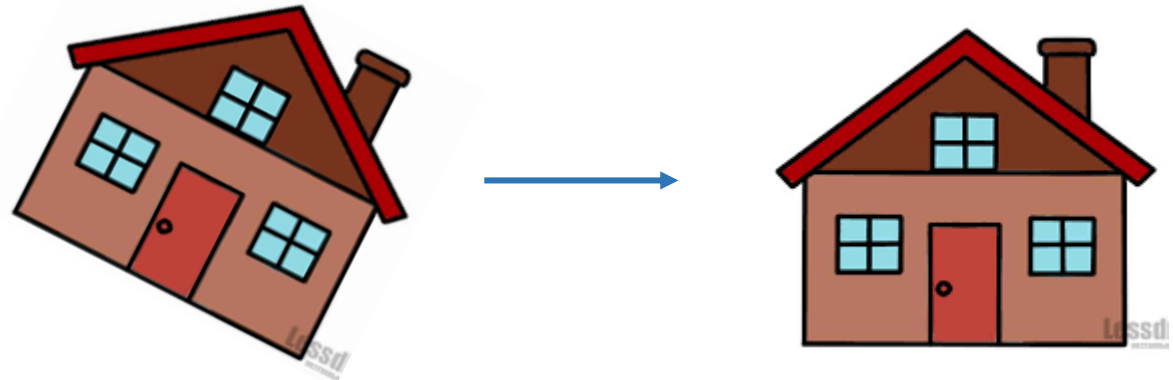
Для каждого пикселя вычисляется значение интенсивности, и на основе этих значений получается определенная характеристика всего изображения в целом. И тогда задача сводится к сравнению этих характеристик. И хотя такие алгоритмы просты и не требуют больших вычислительных затрат, результат их работы оставляет желать лучшего.

**Метод будет выдавать неприемлемый результат в случае появления новых объектов, перекрытия одних объектов другими, появления шума, изменение положения объекта в пространстве и так далее.**

# А если изображение повернуто

---

Есть два изображения  
одного и того же объекта.  
Как нам совместить  
изображения  
автоматически?



Нужно найти такое преобразование (совмещение изображений), при котором изображения больше всего совпадут

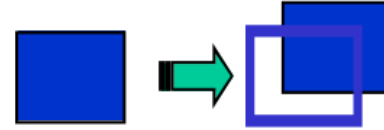
Для этого нужно определить:

1. Какое преобразование использовать?
2. Как оценить совпадение (похожесть изображений)?

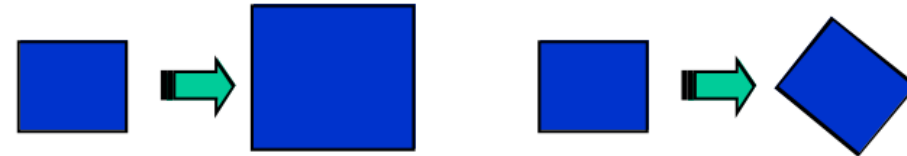
# 1. Какое преобразование использовать?

---

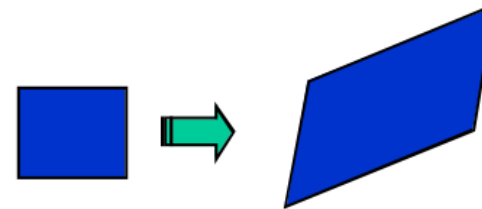
- Параллельный перенос



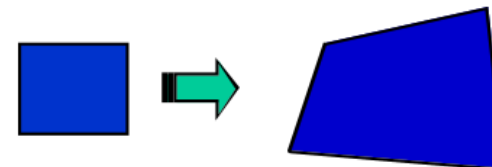
- Подобие (перенос, масштаб, поворот)



- Аффинное



- Проективное (гомография)



## 2. Как оценить совпадение

---

Попиксельное сравнение изображений:

$$\sum_X \sum_Y |I_1(X, Y) - I_2(X, Y)|$$

L1 метрика (SAD - Sum of absolute differences)

$$\sum_X \sum_Y (I_1(X, Y) - I_2(X, Y))^2$$

L2 метрика (SSD - Sum of squared differences)

$$\sum_X \sum_Y I_1(X, Y) I_2(X, Y)$$

Кросс-корреляция (CC - Cross-correlation)

- SAD, SSD – минимизируются (0 – точное совпадение)
- CC – максимизируется (1 – точное совпадение)



# Второй подход к сопоставлению изображений

Вопрос: Как люди собирают пазлы и сопоставляют изображения ?



Пример:

В верхней части изображения даны шесть небольших фрагментов изображения. Найдите точное расположение этих фрагментов на исходном изображении.

А и В — однотонные поверхности.

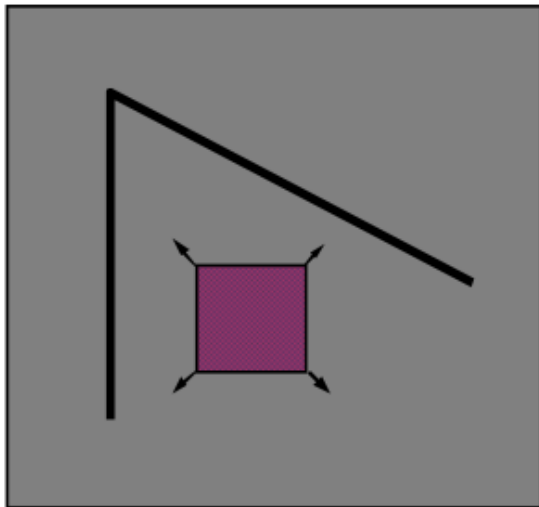
С и D — края здания.

Е и F — некоторые углы здания.

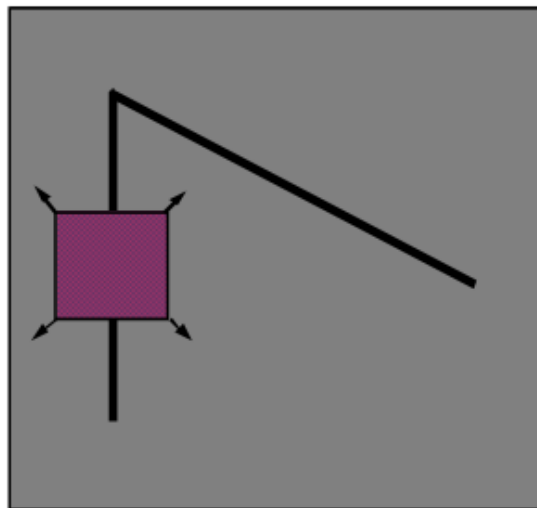
**Какие фрагменты однозначно определяют свое местоположение?**

# Локальные особенности

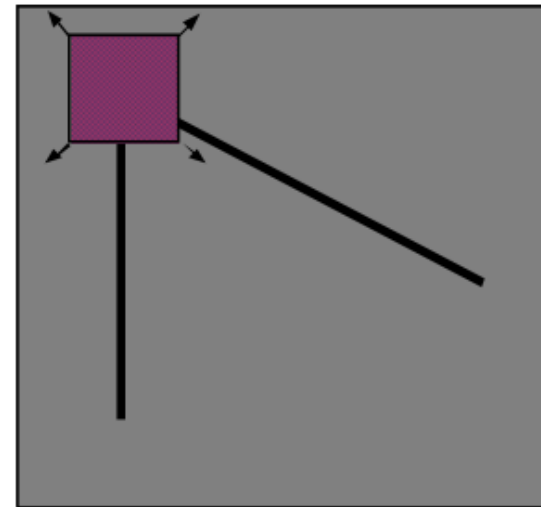
---



*монотонный* регион:  
в любом направлении  
изменений нет



*«край»*:  
вдоль края  
изменений нет



*«уголок»*:  
изменения при  
перемещении  
в любую сторону



# Локальные особенности



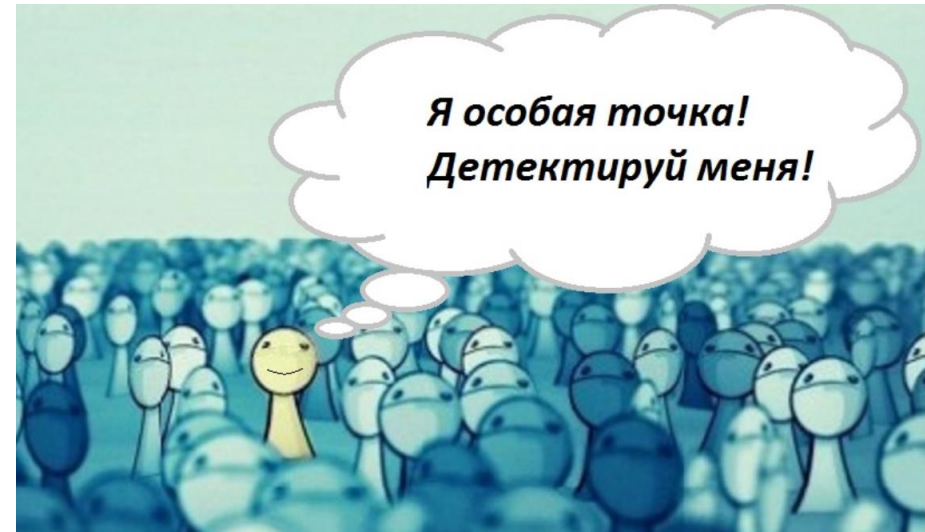
*features – особые точки*

Особые точки (в разных источниках – *features/characteristic points/local feature points/interest point/локальные особенности*) – говоря неформально – “хорошо различимые” фрагменты изображения. Это точки (пиксели) с характерной (особой) окрестностью – т.е. отличающиеся своей окрестностью от всех соседних точек. Классический пример локальной особенности – вершина угла (а не какая-то произвольная точка на прямой или на однородном фоне). Описываются вектором признаков вычисляемых на основе интенсивности/градиентов или других характеристик точек окрестности. Используя особые точки можно анализировать как изображения целиком так и объекты на них. Хорошие характерные точки

позволяют справиться с изменением масштаба, ракурса и перекрытиями сцены или объекта.

Требования, которым должны удовлетворять особые точки:

1. **Отличимость**: Особая точка должна выделяться на фоне и быть уникальной в своей окрестности.



2. **Инвариантность**: Выбор особых точек должен быть независимым от афинных преобразований.

3. **Стабильность**: Выбор особых точек должен быть устойчив к шуму и ошибкам.

4. **Уникальность**: Помимо локальной отличимости, особая точка должна также обладать свойством глобальной уникальности для того, чтобы улучшить различимость повторяющихся паттернов.

5. **Интерпретируемость**: Особые точки должны определяться таким образом, чтобы их можно было использовать для анализа соответствий и лучшей интерпретации изображения.

Процесс сравнения изображений разделяется на три этапа.

1. Первый этап — **нахождение множества особых точек с помощью методов, называемых детекторами**. Данные методы обеспечивают инвариантность нахождения одних и тех же точек относительно преобразований изображения. Однако, недостаточно использования лишь детектора, так как результатом его работы является множество координат особых точек, которые на каждом изображении различны.

2. Для этого на втором этапе происходит построение дескрипторов. **Дескриптор** — это описание точки, уникально идентифицирующее её среди множества всех точек. Дескриптор должен обеспечивать инвариантность нахождения соответствий между точками относительно преобразований изображения. Некоторые методы выполняют сразу обе задачи, детектирование особых точек и построение дескрипторов.

3. Третий этап заключается в сравнении дескрипторов и поиске точек, совпадающих на обоих изображениях.

Существуют различные подходы к определению особых точек.

Это могут быть методы, основанные на:

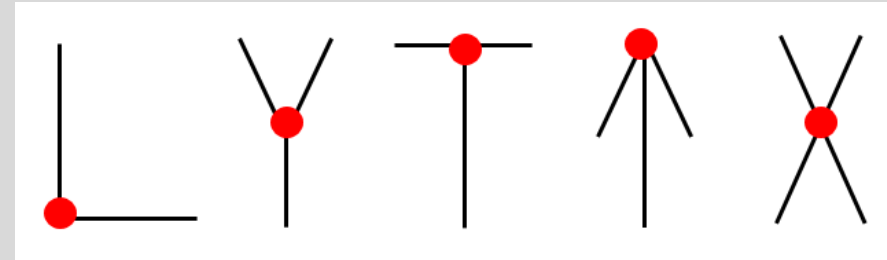
- измерении интенсивности изображения
- поиске максимальной кривизны контуров
- определяющие пересечение контуров и т. д.

<https://habr.com/ru/post/244541/>

Рассмотрим локальные особенности в пересечениях контуров

**Углы** (corners) – особые точки, которые формируются из двух или более граней.

Или **угол** – это точка, у которой в окрестности интенсивность изменяется относительно центра  $(x, y)$ .



Главное свойство таких точек заключается в том, что в области вокруг угла у градиента изображения преобладают два доминирующих направления, что делает их различимыми.

В зависимости от количества пересекаемых граней существуют разные виды уголков на рис. Различные детекторы углов по-разному реагируют на каждый из таких видов уголков.

# Детекторы углов

---

Итак углы — это области изображения с большим разбросом интенсивности во всех направлениях. Одна из первых попыток найти эти углы была предпринята Крисом Харрисом и Майком Стивенсом в 1988 году, поэтому теперь этот метод называется «**Детектор углов Харриса**». Они рассмотрели производные яркости изображения для исследования изменений яркости по множеству направлений.

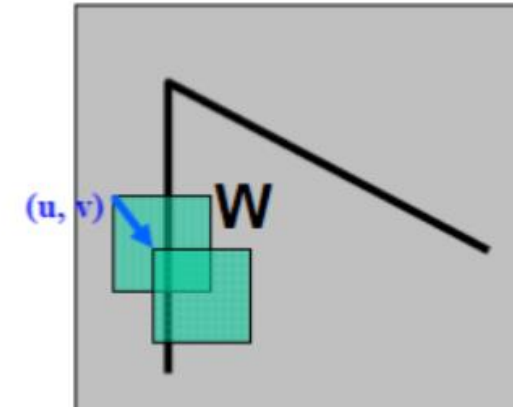
Детектор Харриса является наиболее оптимальным детектором углов и широко применяется.

Детектор Харриса инвариантен к поворотам, частично инвариантен к аффинным изменениям интенсивности. К недостаткам стоит отнести чувствительность к шуму и зависимость детектора от масштаба изображения (для устранения этого недостатка используют многомасштабный детектор Харриса (multi-scale Harris detector)).

# Детектор углов Харриса

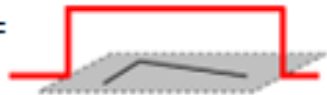
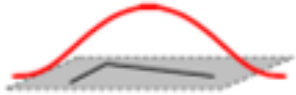
Для данного изображения  $I$  рассмотрим окно  $W$  (обычно размер окна равен  $5 \times 5$  пикселей, но может зависеть от размера изображения) в центре  $(x, y)$ , а также его сдвиг на  $(u, v)$ .

Тогда взвешенная сумма квадрата разностей (sum of squared differences (SSD)) между сдвинутым и исходным окном (т.е. изменение окрестности точки  $(x, y)$  при сдвиге на  $(u, v)$ ) равна:



$$E(u, v) = \sum_{(x, y) \in W} w(x, y) (I(x + u, y + v) - I(x, y))^2 \approx \sum_{(x, y) \in W} w(x, y) (I_x(x, y)u + I_y(x, y)v)^2 \approx \begin{pmatrix} u & v \end{pmatrix} M \begin{pmatrix} u \\ v \end{pmatrix},$$

где  $w(x, y)$  – весовая функция (обычно используется функция Гаусса или бинарное окно):

$w(x, y) =$   или 

1 – в окне, 0 – вне окна      Функция Гаусса

$M$  – автокорреляционная матрица:

$$M = \sum_{(u, v) \in W} w(u, v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

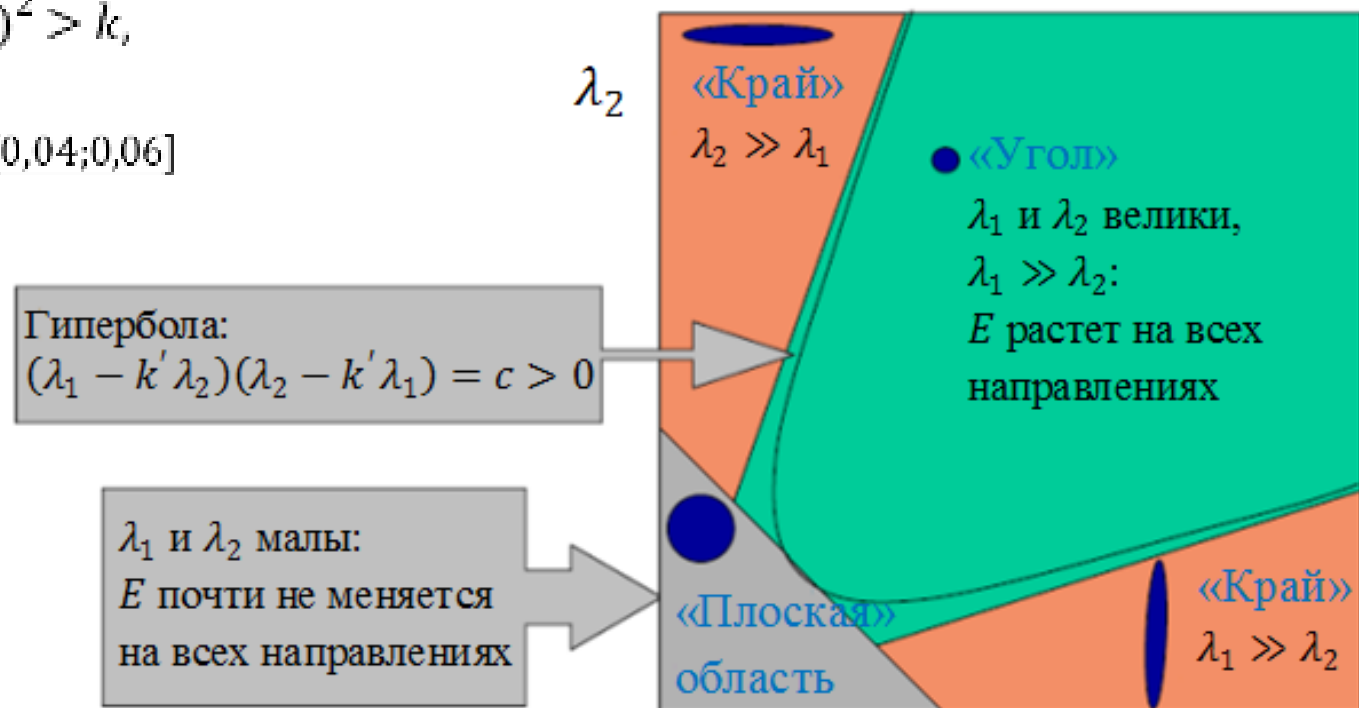
Здесь  $I_x$  и  $I_y$  – производные изображения по осям  $x$  и  $y$  соответственно (как в фильтре Собеля).



После этого авторы создали оценку (меру отклика), которая будет определять, может ли окно содержать угол или нет:

$$R = \det M - k(\text{tr} M)^2 > k,$$

где  $k$  – эмпирическая константа,  $k \in [0,04;0,06]$



Таким образом, значения этих собственных значений определяют, является ли область угловой, краевой или плоской.

- Когда  $|R|$  мало, что бывает, когда  $\lambda_1$  и  $\lambda_2$  малы, область плоская.
- Когда  $R < 0$ , что бывает когда  $\lambda_1 \gg \lambda_2$  или наоборот, область является краевой.
- Когда  $R$  большое, что происходит, когда  $\lambda_1$  и  $\lambda_2$  большие и  $\lambda_1 \sim \lambda_2$ , область представляет собой угол.



# Детектор углов Харриса (Python)

```
cv.cornerHarris(src, blockSize, ksize, k[, dst[, borderType]]) ->dst
```

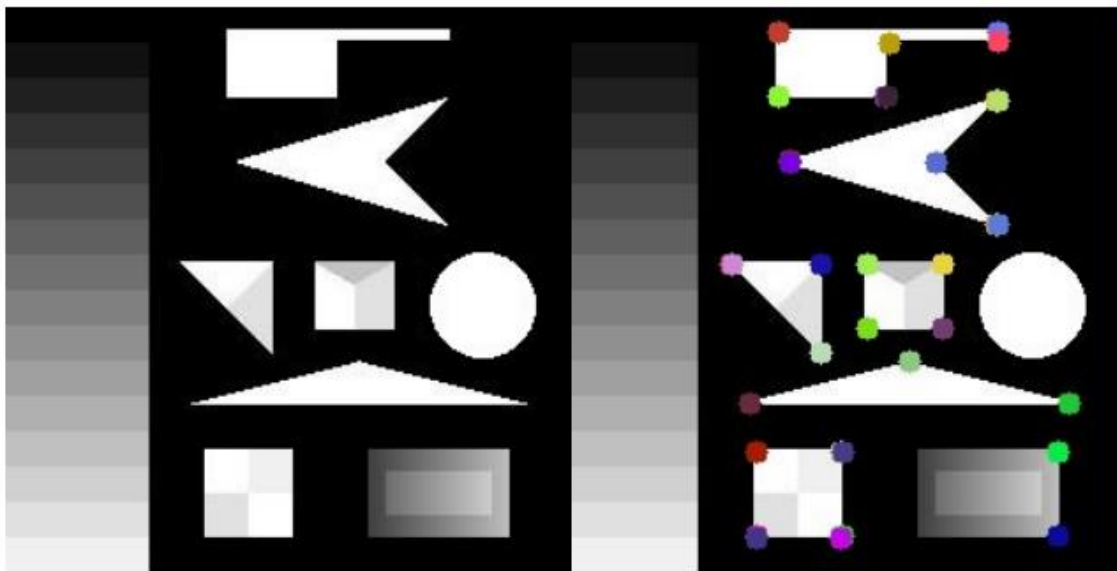
**src** — входное изображение, оно должно быть в оттенках серого и иметь тип float32.

**blockSize** - это размер окрестности, учитываемой для обнаружения углов.

**ksize** - Используемый параметр апертury производной Собеля.

**k** - эмпирическая константа детектора Харриса.

**borderType** - метод пиксельной экстраполяции.



Особые точки, найденные методом Харриса на тестовом изображении

Так как детектор Харриса является детектором углов, он не обнаружил особых точек на окружности и рёбрах объектов. Также, детектор игнорирует углы, яркость которых не очень высока (прямоугольники разной яркости в левой части изображения и углы внутри некоторых фигур).

# Результаты работы детектора Харриса



Детектор Харриса инвариантен к поворотам, частично инвариантен к аффинным изменениям интенсивности. К недостаткам стоит отнести чувствительность к шуму и зависимость детектора от масштаба изображения



## Детектор Shi-Tomasi (Ши – Томаси)

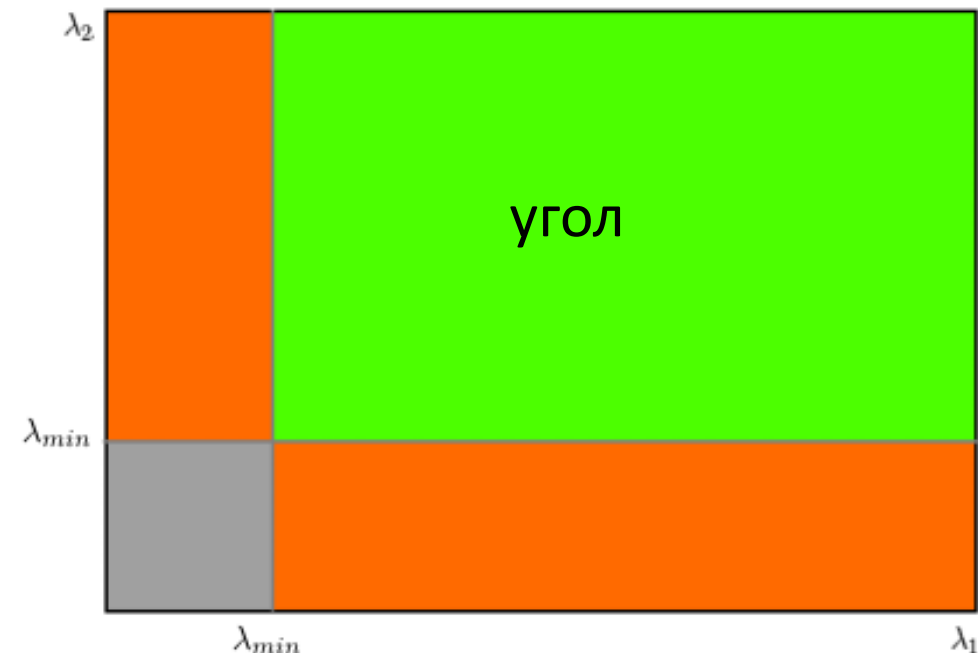
В 1994 году, Дж. Ши и К. Томаси внесли в него небольшую модификацию в детектор Хариса, которая показывает лучшие результаты по сравнению с детектором углов Харриса. Функция подсчета очков в Harris Corner Detector была предоставлена:

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$$

Вместо этого Ши-Томаси предложил:

$$R = \min(\lambda_1, \lambda_2)$$

Если  $R$  больше порогового значения, оно считается углом. Если мы нанесем его  $\lambda_1 - \lambda_2$  пространстве, как мы это сделали в Harris Corner Detector, мы получим следующее изображение:



# Детектор углов Ши-Томаси (Python)

В OpenCV есть функция **cv2.goodFeaturesToTrack()** . Она находит N самых сильных углов на изображении по методу Ши-Томаси.

```
cv.goodFeaturesToTrack(image, maxCorners, qualityLevel, minDistance[,  
corners[, mask[, blockSize[, useHarrisDetector[, k]]]]) -> corners
```

**image** изображением в градациях серого

**maxCorners** количество углов

**qualityLevel** уровень качества, который представляет собой значение от 0 до 1, обозначающее минимальное качество угла, ниже которого все отвергаются

**minDistance** минимальное расстояние между соседними углами

**blockSize** задает размер окрестности при вычислении угла; типичное значение равно 3, но для изображений с высоким разрешением его можно немного увеличить

**useHarrisDetector** параметр, указывающий, следует ли использовать детектор Харриса (angleHarris ) или angleMinEigenVal

Все углы ниже уровня качества отбраковываются. Затем оставшиеся углы сортируются по качеству в порядке убывания. Затем функция берет первый самый сильный угол, отбрасывает все близлежащие углы в диапазоне минимального расстояния и возвращает N самых сильных углов.

```
corners = cv2.goodFeaturesToTrack(gray, 25, 0.01, 10)
```



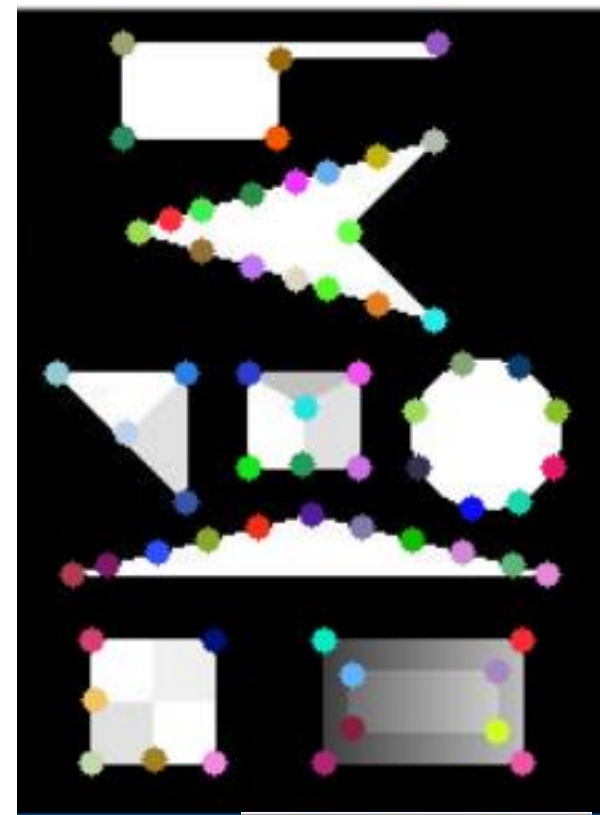
```
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('simple.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

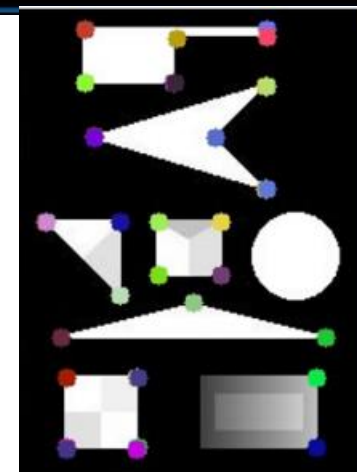
corners = cv2.goodFeaturesToTrack(gray,25,0.01,10)
corners = np.int0(corners)

for i in corners:
    x,y = i.ravel()
    cv2.circle(img,(x,y),3,255,-1)

plt.imshow(img),plt.show()
```

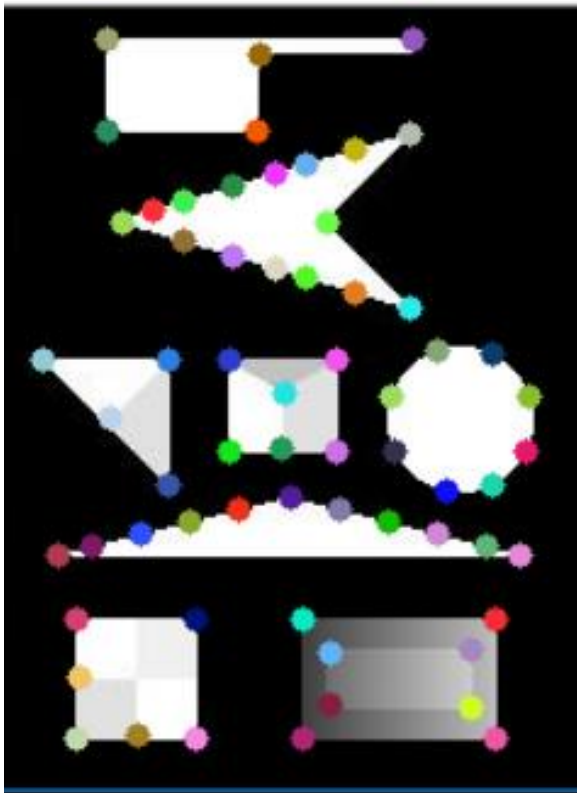


Так было в детекторе Хариса

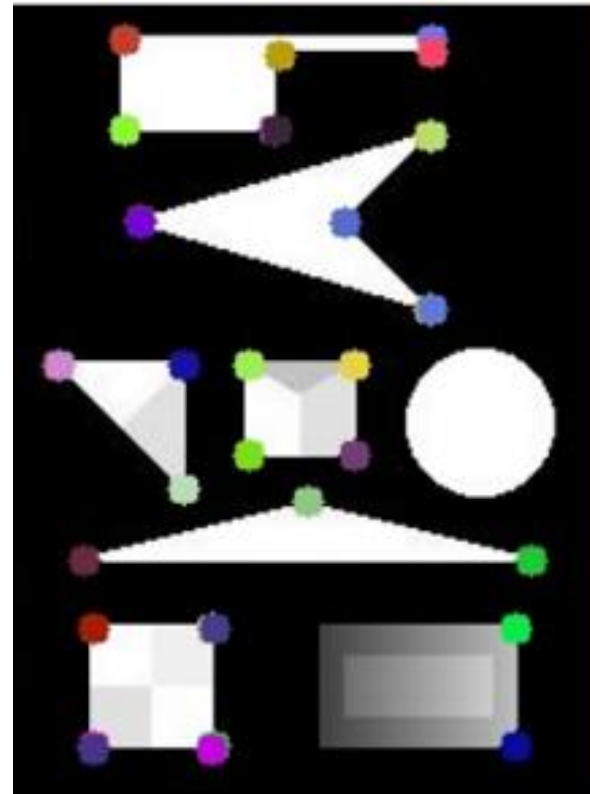


## Сравним результаты детектирования углов

Детектор Ши-Томаси



Детектор Харриса



Алгоритм обнаружил Ши-Томаси некоторые из углов, которые не обнаружил детектор Харриса, однако допустил множество ошибок, найдя особые точки на диагональных рёбрах и окружности. Плюсом Shi-Tomasi можно считать отсутствие дубликатов точек в окрестностях углов. Так же как и детектор Харриса, Shi-Tomasi игнорировал углы недостаточной яркости (например, прямоугольники в левой части изображения), хоть таких и оказалось меньше.

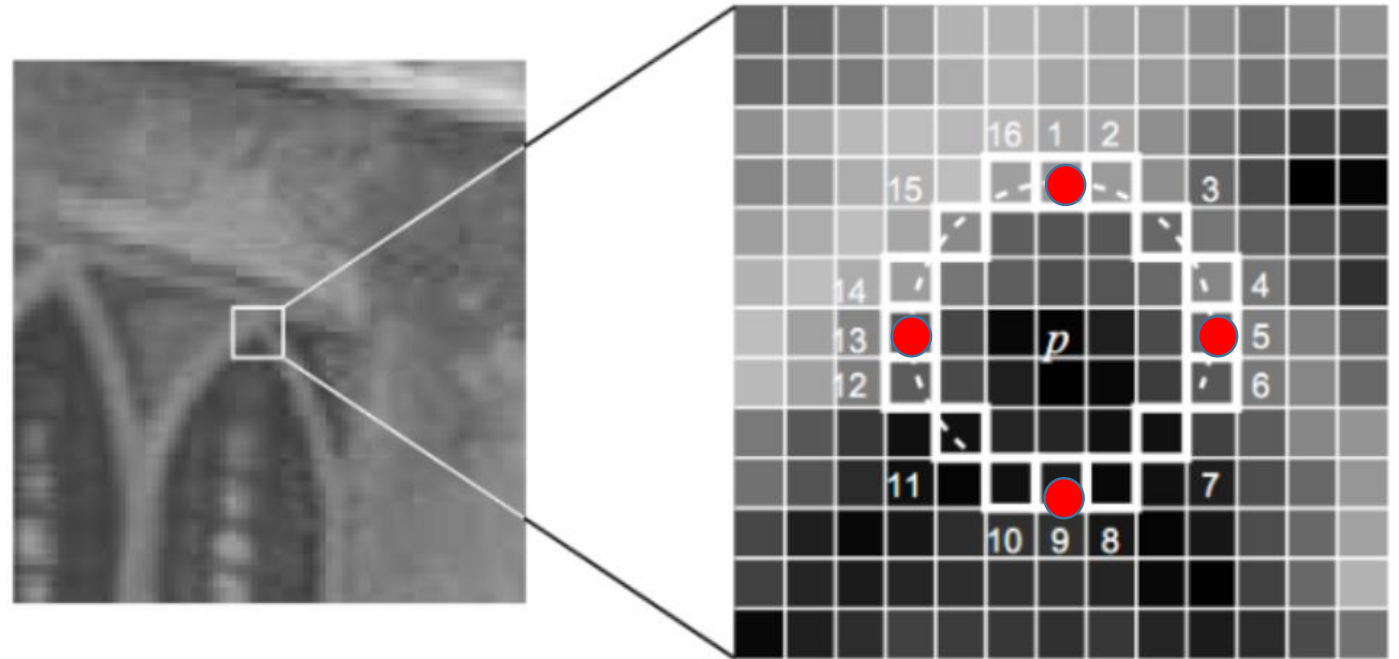
# Алгоритм FAST

В середине 2000-х годов появились алгоритмы быстрого поиска особенных точек. Наиболее ярким представителем данного класса алгоритмов является алгоритм **FAST** (*features from accelerated segment test* — особенности, полученные из ускоренной проверки сегментов).

<https://habr.com/ru/post/244541/>

В алгоритме рассматривается окружность из 16 пикселей. Яркость пикселей, лежащих на окружности, сравнивается с яркостью центральной точки и на основании ряда проверок принимается решение, является ли центральная точка характерной.

Последовательность проверок и их общее число подбираются и оптимизированы заранее на основе обширной обучающей выборки изображений.



Алгоритм FAST хорошо зарекомендовал себя в приложениях, осуществляющих слежение за объектами в реальном времени.

В OpenCV реализован алгоритм ORB [алгоритм ORB \(Oriented FAST and Rotated BRIEF\)](#)

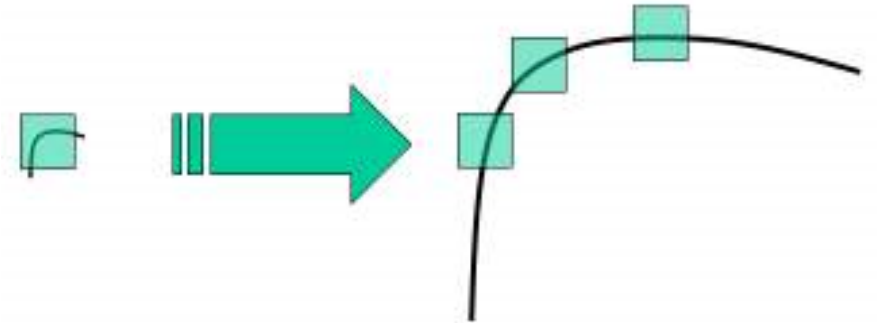


# Масштабирование

---

Рассмотренный ранее детектор углов Хариса инвариантен к вращению, что означает, что даже если изображение повернуто, мы можем найти те же самые углы. Это очевидно, потому что углы остаются углами и в повернутом изображении.

Но как насчет масштабирования? Угол может не быть углом, если изображение масштабировано. Например, проверьте простое изображение ниже. Угол маленького изображения в маленьком окне становится плоским при увеличении масштаба в том же окне. Таким образом, угол Харриса не является масштабно-инвариантным.



В 2004 году Д. Лоу из Университета Британской Колумбии в своей статье « Отличительные признаки изображения из масштабно-инвариантных ключевых точек » придумал новый алгоритм Scale Invariant Feature Transform (**SIFT**) , который **извлекает ключевые точки** и **вычисляет их дескрипторы**.

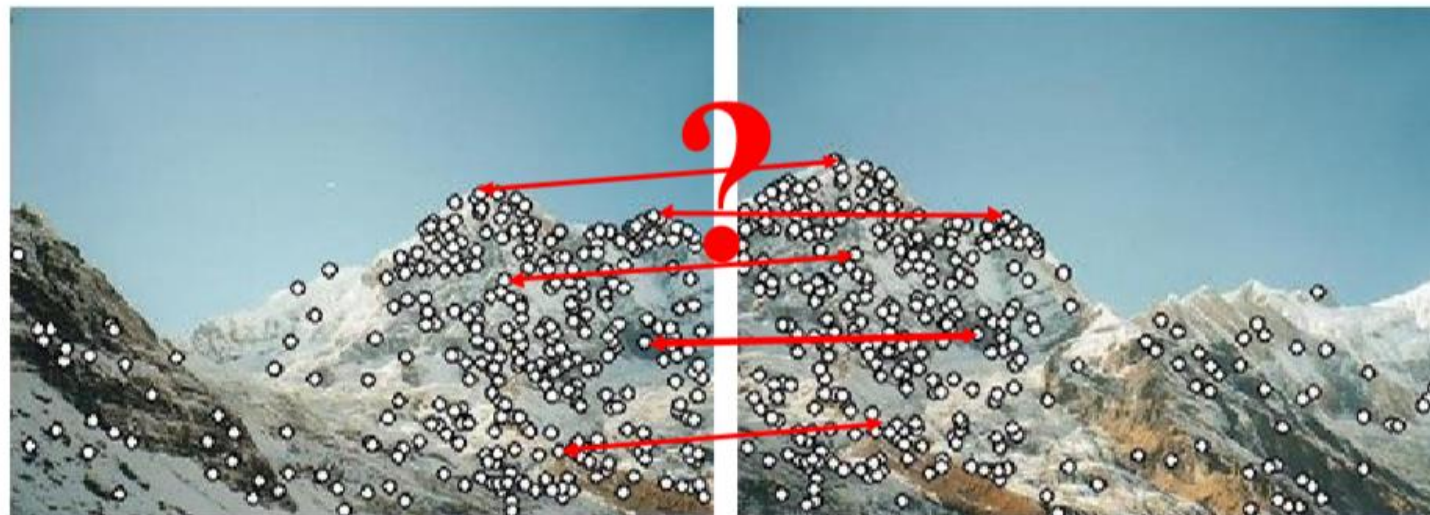
Рассмотрим подробнее понятие дескриптор



# Дескрипторы

---

Точки найдены, а как их отличить друг от друга



**Дескриптор** — идентификатор ключевой точки, выделяющий её из остальной массы особых точек.

Дескриптор представляет собой запись фрагмента картинки в числовом виде, с помощью дескрипторов можно достаточно точно сравнивать фрагменты без использования самих фрагментов, затем проводится кластеризация, т.е. распределение похожих дескрипторов по кластерам

# Дескрипторы

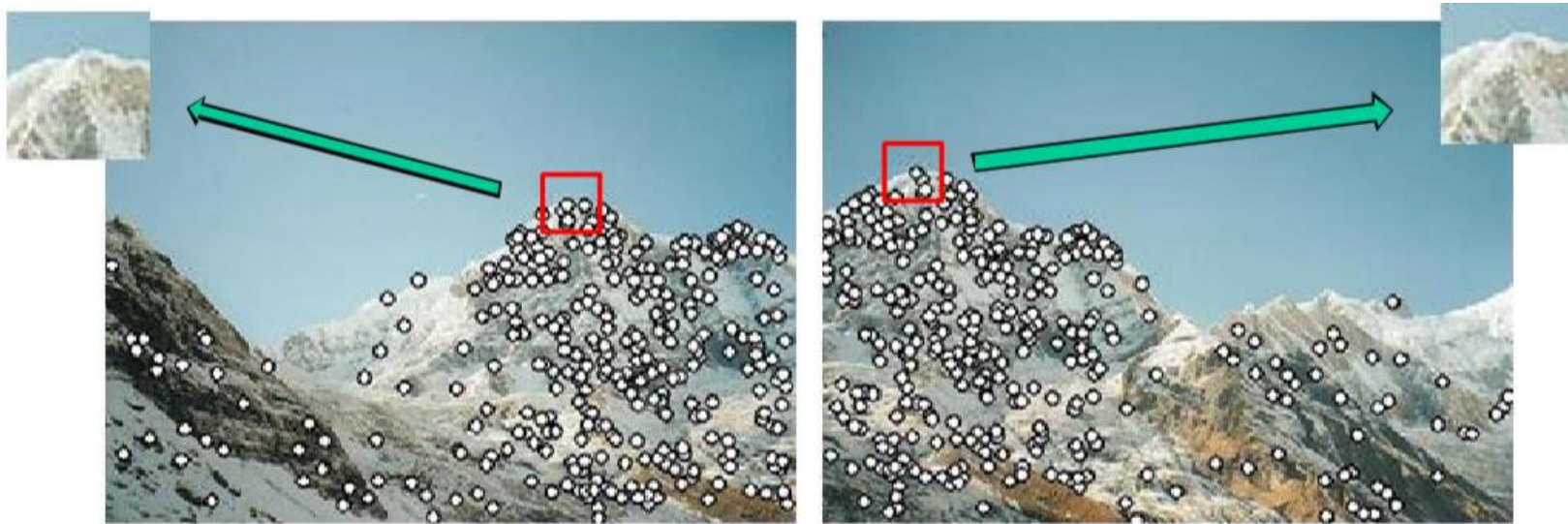
---

Дескрипторы должны быть:

- специфичны (различные для разных точек)
- локальны (зависеть только от небольшой окрестности)
- инвариантны (к искажениям, изменениям освещенности)
- просты в вычислении

# Дескрипторы. Простейший подход

---



- Возьмём квадратные окрестности, со сторонами, параллельными строкам и столбцам изображения
- Яркости пикселей будут признаками
- Сравнивать будем как изображения попиксельно (SAD, SSD)
- Такая окрестность инвариантна только к сдвигу изображения

# Дескрипторы. Инвариантность к яркости

---

- Можем добиться следующим образом:
  - Локальная нормализация гистограммы
  - Дескрипторы, основанные на градиенте яркости, инвариантны к сдвигу яркости
  - Нормирование яркости - вычесть среднее значение, поделить на дисперсию

$$I' = (I - \mu) / \sigma$$



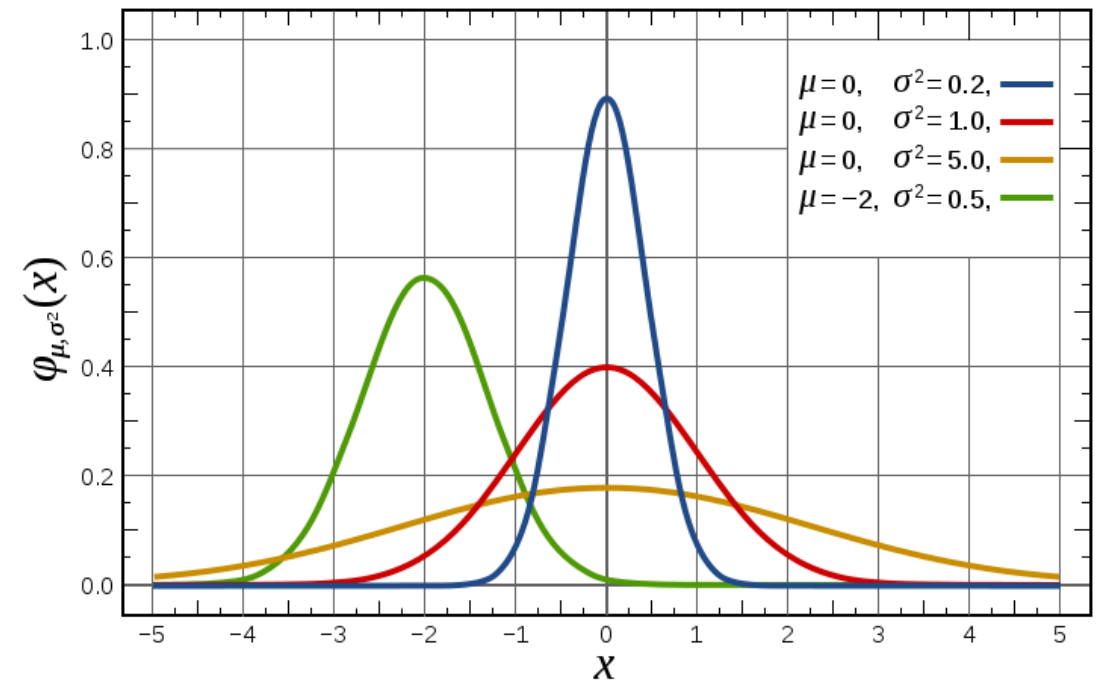
нормализация



## Алгоритм SIFT состоит в основном из четырех шагов.

### 1. Обнаружение экстремумов в масштабном пространстве

На предыдущем слайде видно, что мы не можем использовать одно и то же окно для обнаружения ключевых точек с разным масштабом. Это нормально с маленьким углом. Но для обнаружения больших углов нам нужны большие окна. Для этого используется масштабно-пространственная фильтрация. В нем находится лапласиан гаусса для изображения с различными сигма-значениями. Сигма действует как параметр масштабирования. Ядро Гаусса с низким сигма-значением дает высокое значение для маленького угла, в то время как ядро Гаусса с высоким значением сигма хорошо подходит для большего угла.



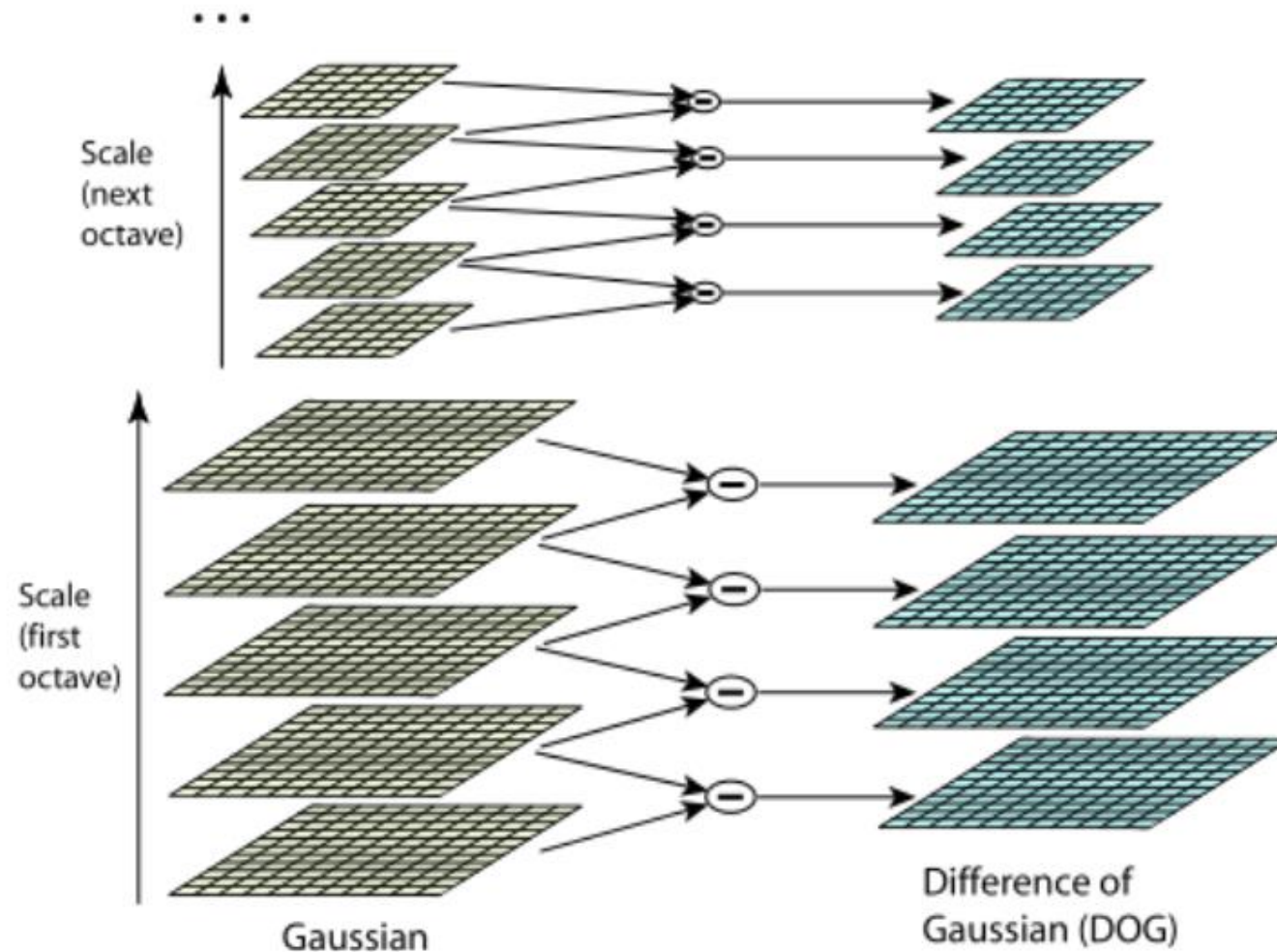


Для ускорения расчетов алгоритм SIFT использует разность гауссианов DoG (Difference of Gaussia)

<https://habr.com/ru/post/106302/>

Разностью гауссианов называют изображение, полученное путем попиксельного вычитания одного гауссиана исходного изображения из гауссиана с другим радиусом размытия.

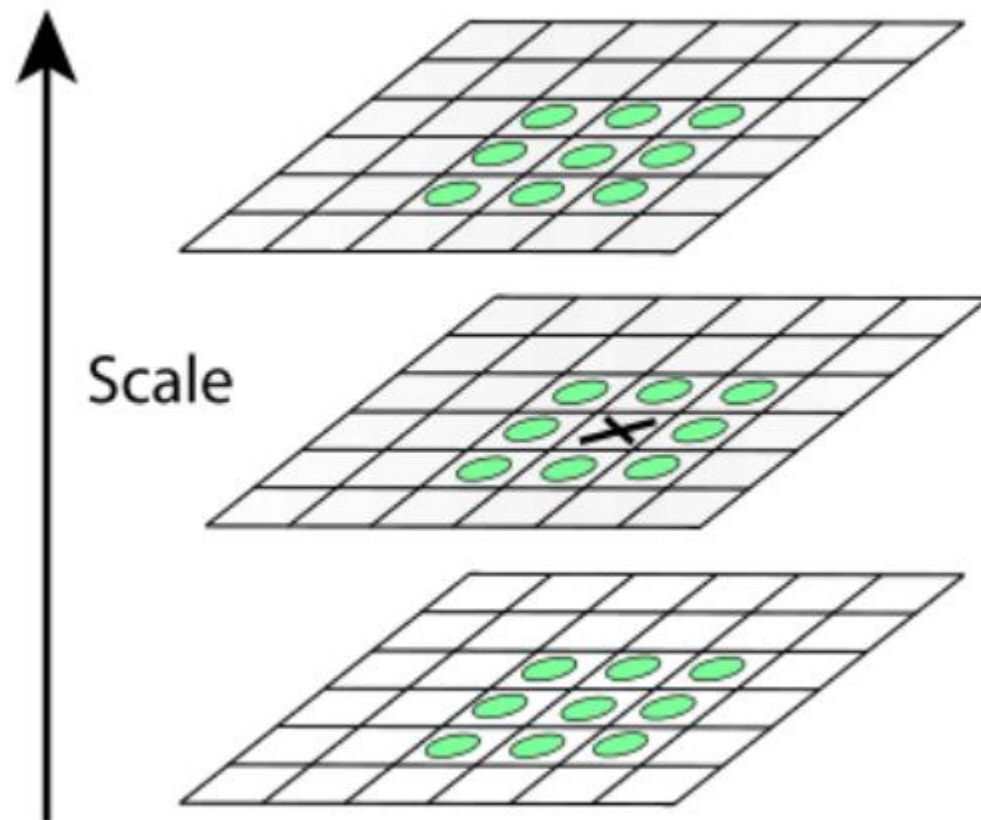
Различная степень размытия изображения гауссовым фильтром может быть принята за исходное изображение, взятое в некотором масштабе.





# DoG (Difference of Gaussia)

---



Будем считать точку особой, если она является локальным экстремумом разности гауссианов.

В каждом изображении из пирамиды DoG ищутся точки локального экстремума. Каждая точка текущего изображения DoG сравнивается с её восемью соседями и с девятью соседями в DoG, находящимися на уровень выше и ниже в пирамиде. Если эта точка больше (меньше) всех соседей, то она принимается за точку локального экстремума.

# Шаги алгоритма SIFT

## 2. Локализация ключевых точек

Как только потенциальные ключевые точки найдены, их необходимо уточнить, чтобы получить более точные результаты. Авторы метода использовали разложение масштабного пространства в ряд Тейлора, чтобы получить более точное местоположение экстремумов, и если интенсивность в этих экстремумах меньше порогового значения, оно отбрасывается. Этот порог называется `counterThreshold` в OpenCV .

DoG имеет более высокую реакцию на края, поэтому края также необходимо удалить. Для этого используется концепция, аналогичная угловому детектору Харриса. Они использовали матрицу Гессе (H)  $2 \times 2$  для вычисления главной кривизны. Из углового детектора Харриса мы знаем, что для ребер одно собственное значение больше другого. Итак, здесь они использовали простую функцию,

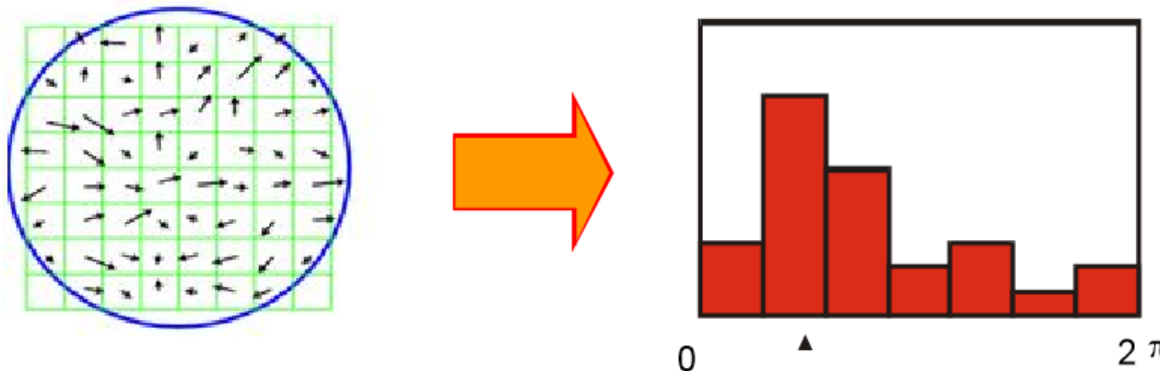
Если это отношение превышает пороговое значение, называемое в OpenCV `edgeThreshold` , эта ключевая точка отбрасывается.

Таким образом, он устраняет любые низкоконтрастные ключевые точки и краевые ключевые точки, и остаются только точки сильного интереса.

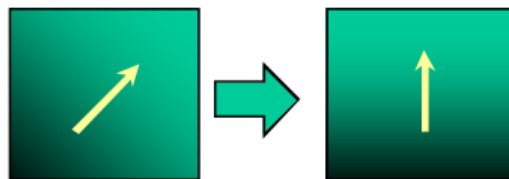
### 3. Назначение ориентации

Каждой ключевой точке назначается ориентация для достижения инвариантности к вращению изображения. Вокруг ключевой точки берется окрестность в зависимости от масштаба, и в этой области вычисляются величина и направление градиента. Создается гистограмма ориентации с 36 ячейками, охватывающими 360 градусов. Берется самый высокий пик на гистограмме.

- Идея: найти основное (доминантное) направление градиентов пикселей в окрестности точки
- Посчитаем гистограмму, взвешивая вклад по гауссиане с центром в точке

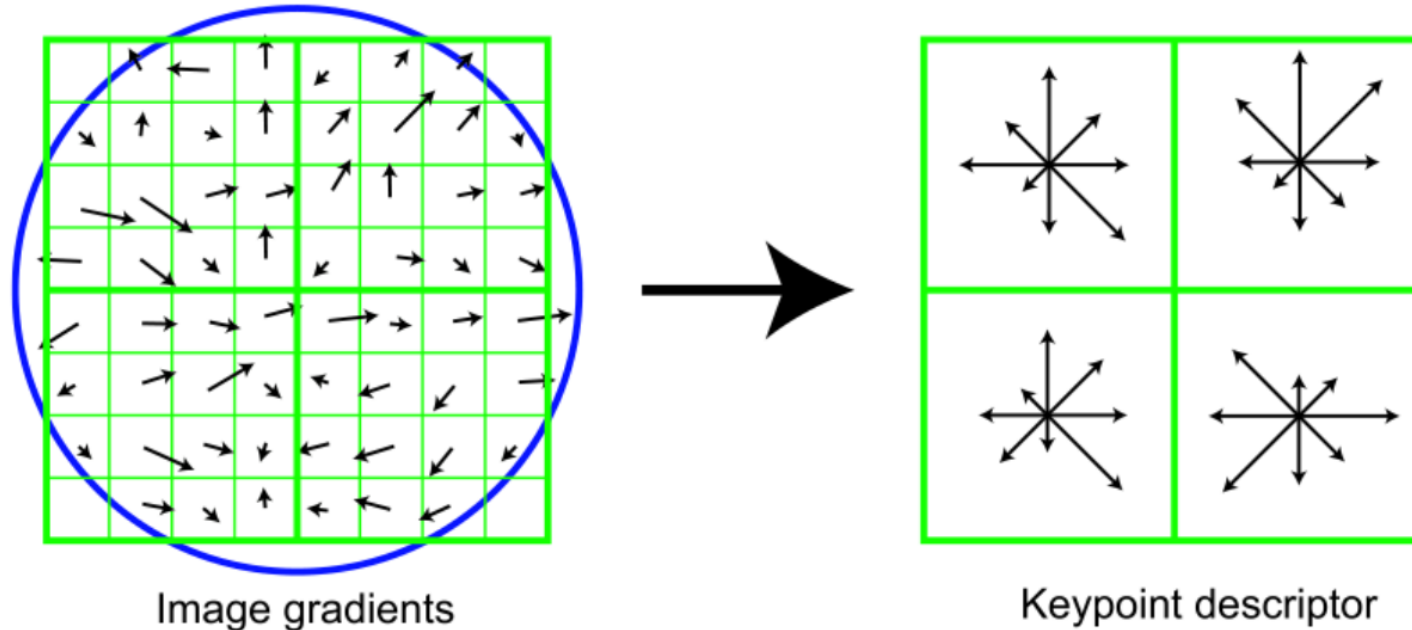


- Повернуть фрагмент так, чтобы доминантное направление градиента было направлено вверх



#### 4. Дескриптор ключевой точки

Берется окрестность  $16 \times 16$  вокруг ключевой точки. Он разделен на 16 подблоков размером  $4 \times 4$ . Для каждого подблока создается гистограмма ориентации с 8 бинами. Таким образом, всего доступно 128 значений бинов. Он представлен в виде вектора для формирования дескриптора ключевой точки длиной 128. В дополнение к этому, предпринимаются некоторые меры для обеспечения устойчивости к изменениям освещения, вращению и т. д.

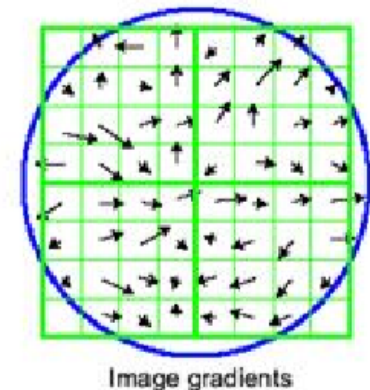


## 5. Сопоставление ключевых точек

Ключевые точки между двумя изображениями сопоставляются путем определения их ближайших соседей. Но в некоторых случаях второе ближайшее совпадение может быть очень близко к первому. Это может произойти из-за шума или по другим причинам. В этом случае берется отношение ближайшего расстояния ко второму ближайшему расстоянию. Если он больше 0,8, они отбраковываются. Согласно статье, он устраняет около 90% ложных совпадений и отбрасывает только 5% правильных совпадений.

### Резюме

- Детектор SIFT весьма специфичен, устойчив к изменениям освещения, небольшим сдвигам
- Вся схема SIFT (детектор, выбор окрестностей, дескриптор) оказалась очень эффективным инструментом для анализа изображений
- Очень широко используется



## SIFT in OpenCV

[https://opencv24-python-tutorials.readthedocs.io/en/latest/py\\_tutorials/py\\_feature2d/py\\_sift\\_intro/py\\_sift\\_intro.html](https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html)

So now let's see SIFT functionalities available in OpenCV. Let's start with keypoint detection and draw them. First we have to construct a SIFT object. We can pass different parameters to it which are optional and they are well explained in docs.

```
import cv2
import numpy as np

img = cv2.imread('home.jpg')
gray= cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

sift = cv2.SIFT()
kp = sift.detect(gray,None)

img=cv2.drawKeypoints(gray,kp)

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

**sift.detect()** function finds the keypoint in the images. You can pass a mask if you want to search only a part of image. Each keypoint is a special structure which has many attributes like its (x,y) coordinates, size of the meaningful neighbourhood, angle which specifies its orientation, response that specifies strength of keypoints etc.



OpenCV also provides **cv2.drawKeypoints()** function which draws the small circles on the locations of keypoints. If you pass a flag, **cv2.DRAW\_MATCHES\_FLAGS\_DRAW\_RICH\_KEYPOINTS** to it, it will draw a circle with size of keypoint and it will even show its orientation. See below example.

```
img=cv2.drawKeypoints(gray,kp,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)  
cv2.imwrite('sift_keypoints.jpg',img)
```

See the two results below:





# SURF (Speeded-Up Robust Features)

В прошлой главе мы видели SIFT для обнаружения и описания ключевых точек. Но это было сравнительно медленно, и людям нужна была более ускоренная версия. В 2006 году три человека, Бэй, Х., Туйтелаарс, Т. и Ван Гул, Л., опубликовали еще одну статью «SURF: ускоренные надежные функции», в которой был представлен новый алгоритм под названием SURF. Как следует из названия, это ускоренная версия SIFT.

```
>>> img = cv2.imread('fly.png',0)

# Create SURF object. You can specify params here or later.
# Here I set Hessian Threshold to 400
>>> surf = cv2.SURF(400)

# Find keypoints and descriptors directly
>>> kp, des = surf.detectAndCompute(img,None)

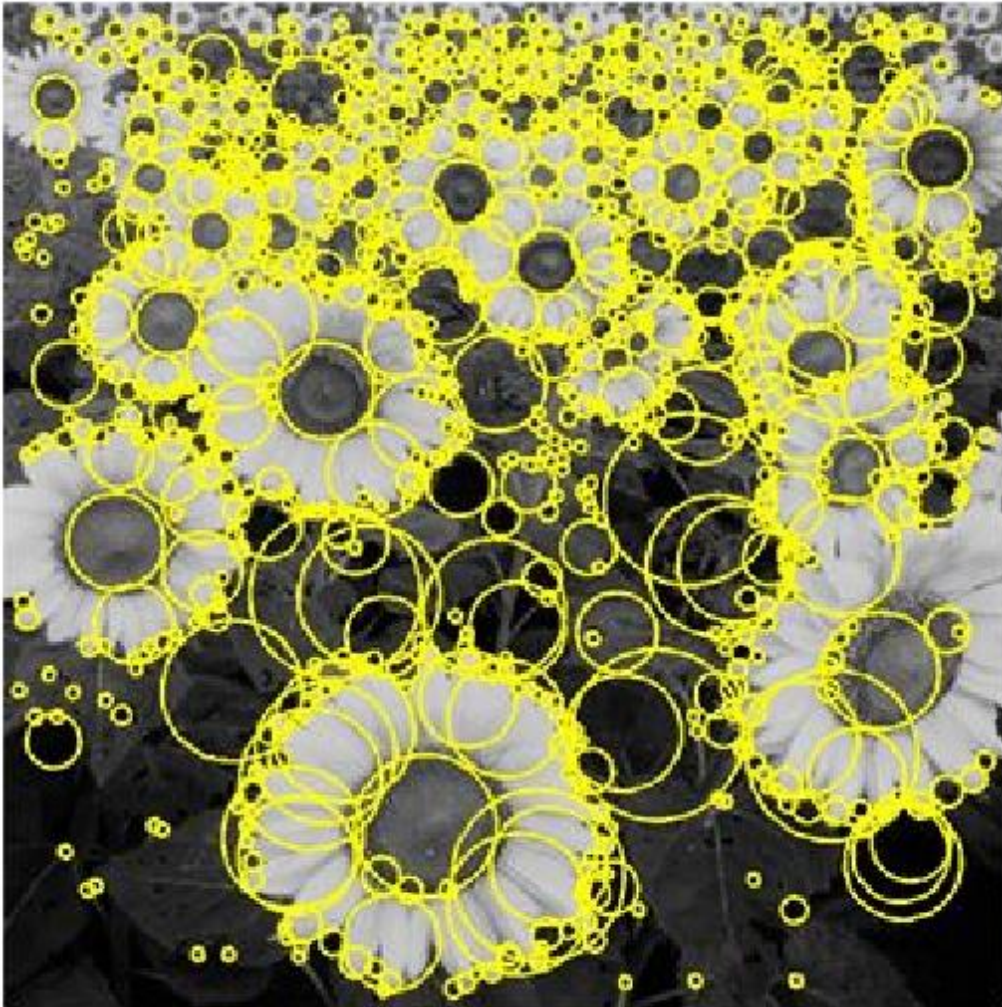
>>> len(kp)
699
```

[https://opencv24-python-tutorials.readthedocs.io/en/latest/py\\_tutorials/py\\_feature2d/py\\_surf\\_intro/py\\_surf\\_intro.html](https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html)

См. результат ниже. Вы можете видеть, что SURF больше похож на детектор блоков. Он обнаруживает белые пятна на крыльях бабочки. Вы можете проверить это с другими изображениями.

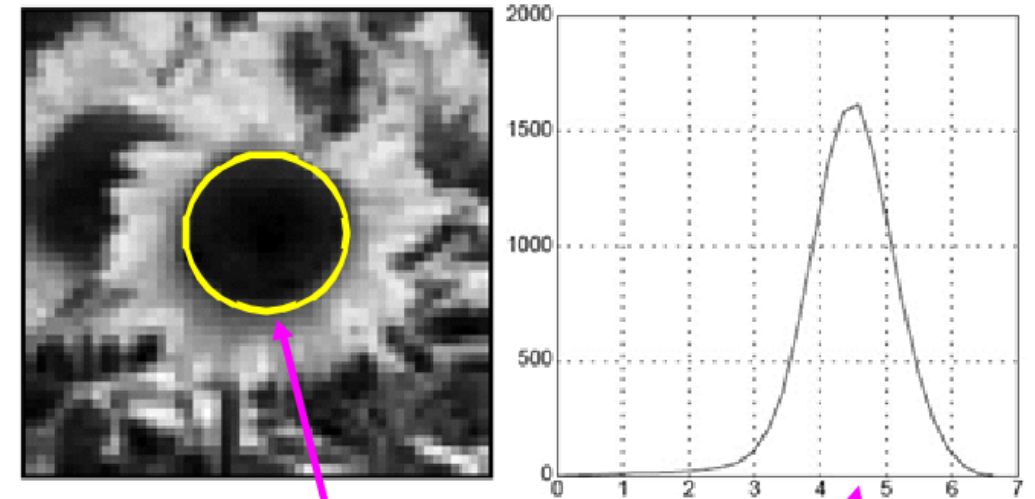


# Блобы



Первый такой метод нахождения блобов был предложен в конце 90-х гг.

Характеристический размер определяется как масштаб, на котором достигается максимум отклика Лапласиана



Характеристический масштаб

# Детекторы областей

---

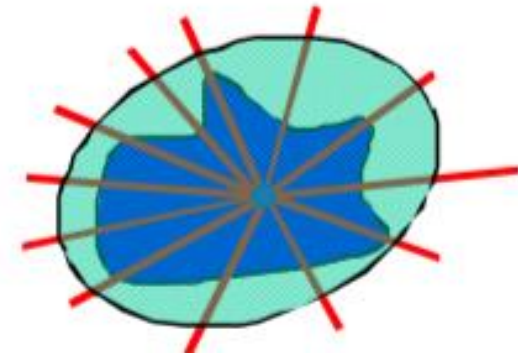
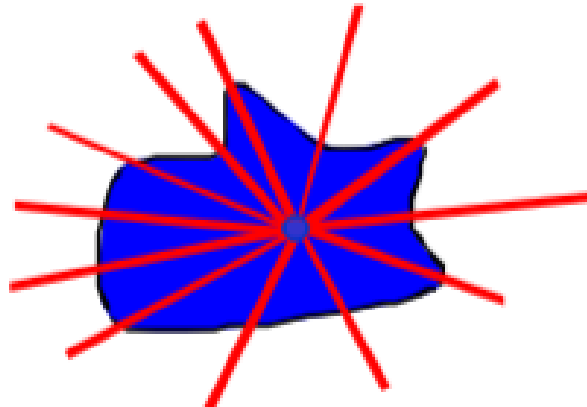
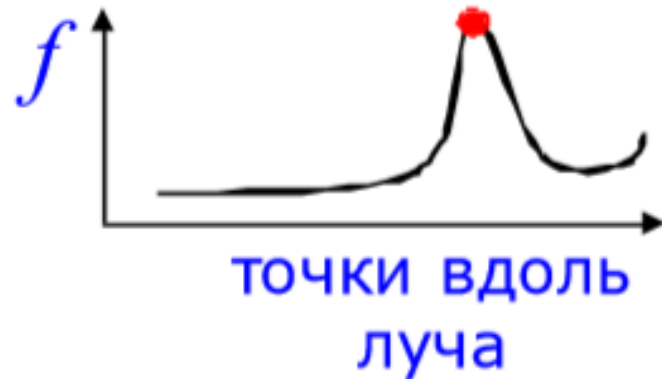
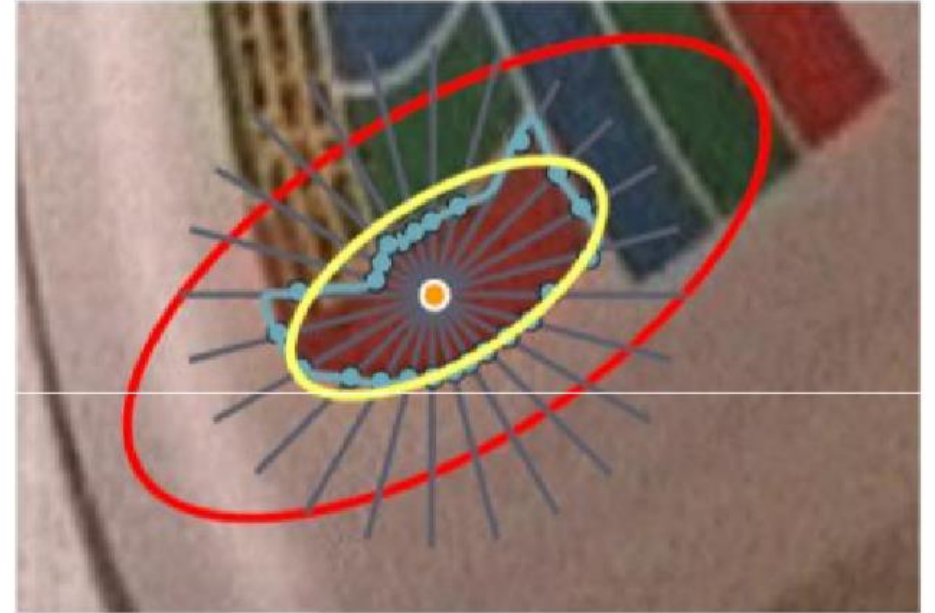
Можно попробовать работать с более уникальными характеристиками изображения – с **областями**. Экстремальных областей гораздо меньше, но они более точно характеризуют сцену или объект.

*У экстремальных (особых) областей* в этом контексте есть два важных свойства:

- непрерывное преобразование координат изображения. Это означает, что он является аффинно-инвариантным, и не имеет значения, будет ли изображение искажено или искажено.
- монотонное преобразование интенсивности изображения. Подход, конечно, чувствителен к естественным световым эффектам, таким как изменение дневного света или движущихся теней.

# Детектор областей IBR (Intensity-extrema based regions)

- Идти от локального экстремума яркости по лучам, считая некоторую величину  $f$
- Остановка при достижении пика  $f$



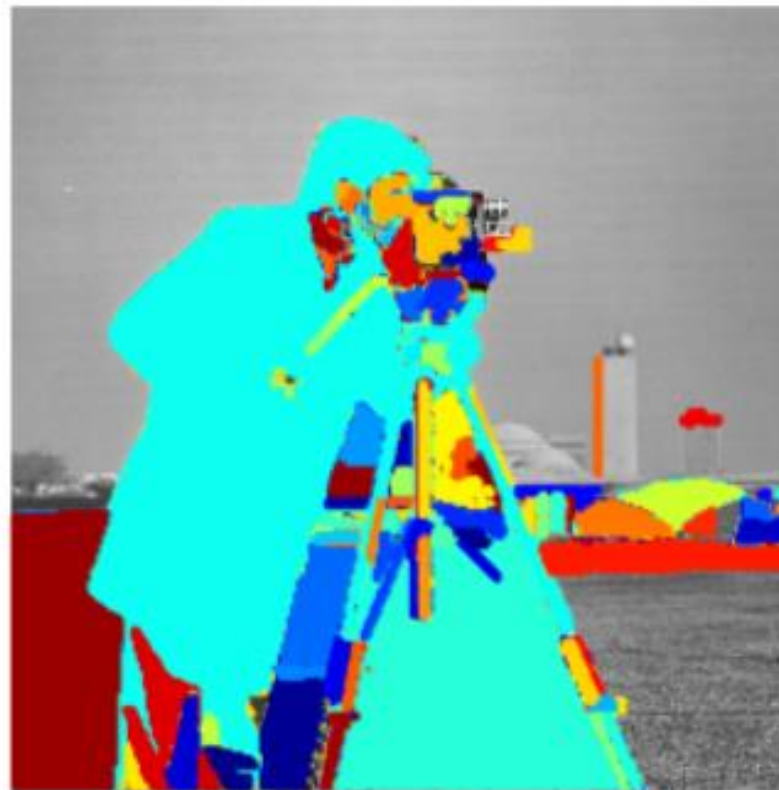


# MSER = Maximally Stable Extremal Regions

---

Метод находит соответствия между элементами изображения в двух изображениях с **разных точек зрения**.

- Задать порог яркости  $T$
- Провести сегментацию
- Извлечь области
- Для каждой области найти порог, при котором рост площади минимален



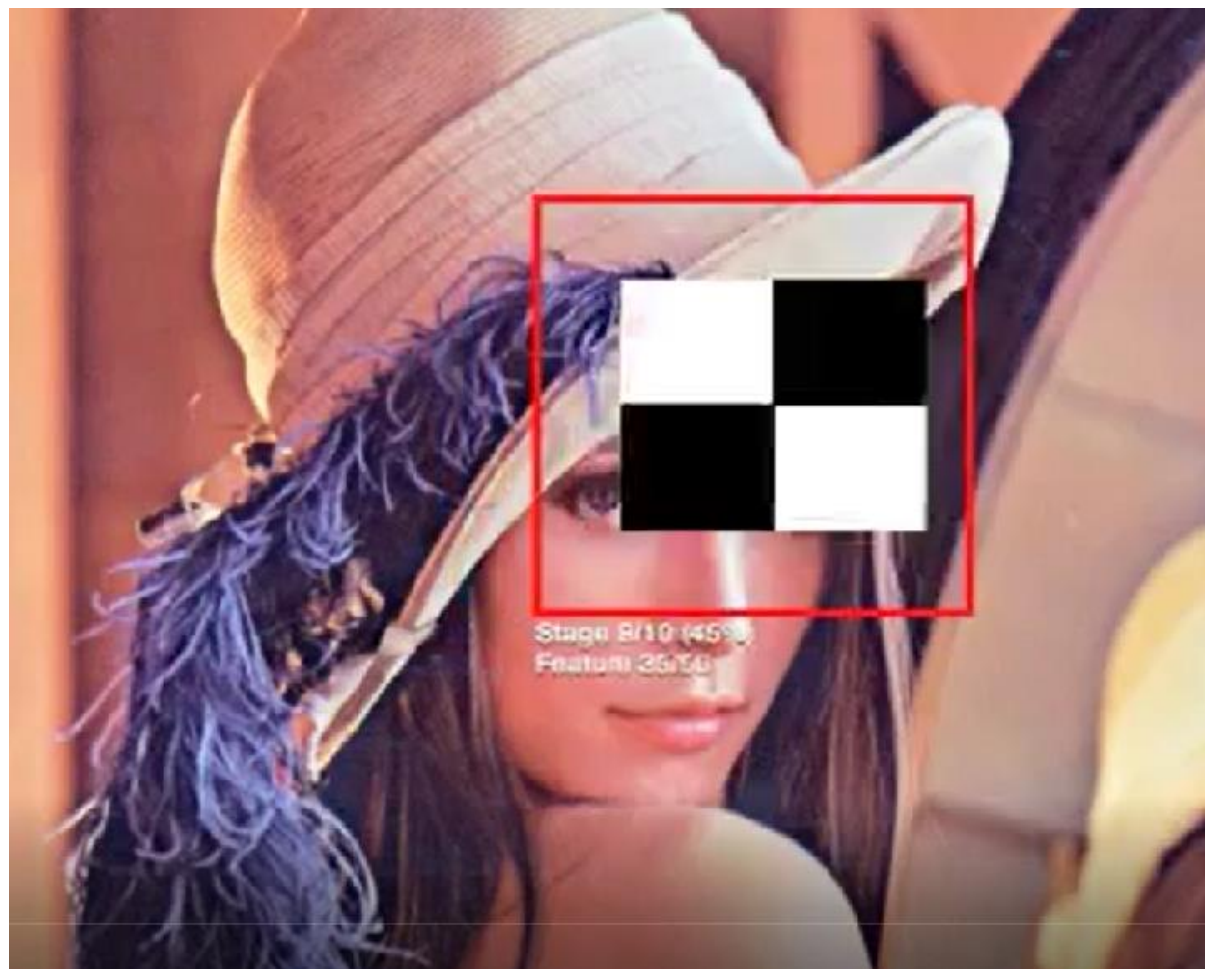
# Резюме

---

- Локальные особенности – один из основных инструментов анализа изображений
- Рассмотрели основные детекторы особенностей:
  - Детекторы углов: Harris (Forstner), Harris-Laplace)
  - Детекторы Блобов: LoG (Laplacian of Gaussian), DoG (Difference of Gaussians)
  - Детекторы областей: IBR (Intensity-extrema based regions), MSER (Maximally Stable Extremal Regions)



# Детектирование лиц



# Метод Виолы-Джонса (Viola-Jones)

---

**Метод Виолы—Джонса** (Viola—Jones object detection) — алгоритм, позволяющий обнаруживать объекты на изображениях в реальном времени.

- Разработан в 2001 году Полом Виолой и Майклом Джонсом.
- Используется в бюджетных цифровых фотоаппаратах.
- Является одним из лучших по соотношению показателей эффективности распознавания/скорость работы.
- Алгоритм находит лица с высокой точностью и низким количеством ложных срабатываний.
- Алгоритм хорошо работает и распознает черты лица под небольшим углом, примерно до 30 градусов.
- Реализован в составе библиотеки компьютерного зрения OpenCV.

# Метод Виолы-Джонса. Основные принципы.

---

Основные принципы, на которых основан метод, таковы:

- используются **изображения в интегральном представлении**, что позволяет вычислять быстро необходимые объекты;
- используются **признаки Хаара**, с помощью которых происходит поиск нужного объекта;
- используется **бустинг** (от англ. boost – улучшение, усиление) для выбора наиболее подходящих признаков для искомого объекта на данной части изображения;
- все признаки поступают на вход **классификатора**, который даёт результат «верно» либо «ложь»;
- используются **каскады признаков** для быстрого отбрасывания окон, где не найдено лицо.

# Интегральное представление изображения

**Интегральное представление изображения** — это матрица, размерность которой совпадает с размерностью исходного изображения.

Каждый элемент интегрального изображения содержит в себе сумму значений всех пикселей левее и выше данного пикселя  $(x, y)$ .

Original					Integral				
5	2	3	4	1	5	7	10	14	15
1	5	4	2	3	6	13	20	26	30
2	2	1	3	4	8	17	25	34	42
3	5	6	4	5	11	25	39	52	65
4	1	3	2	6	15	30	47	62	81

$$5 + 2 + 3 + 1 + 5 + 4 = 20$$

Original					Integral				
5	2	3	4	1	5	7	10	14	15
1	5	4	2	3	6	13	20	26	30
2	2	1	3	4	8	17	25	34	42
3	5	6	4	5	11	25	39	52	65
4	1	3	2	6	15	30	47	62	81

$$5 + 4 + 2 + 2 + 1 + 3 = 17$$

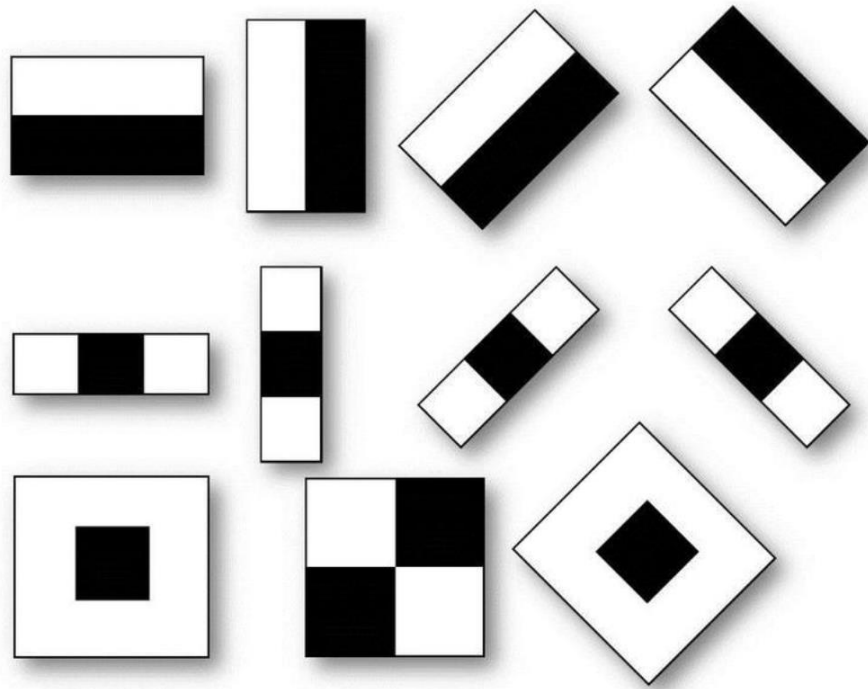
$$34 - 14 - 8 + 5 = 17$$

Интегральное представление позволяет **быстро вычислить сумму пикселей произвольного прямоугольника** с помощью четырех ссылок на массив.

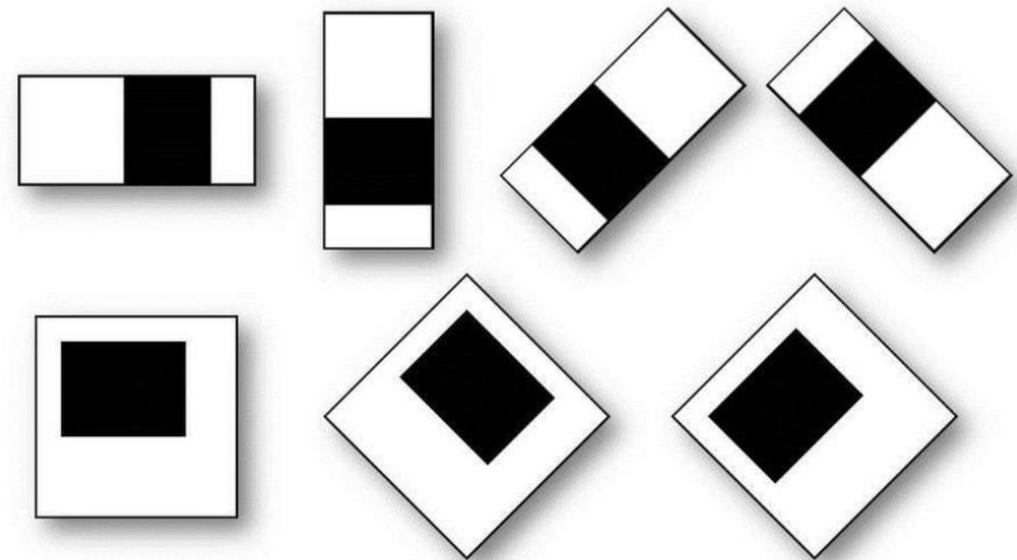
# Признаки Хаара

---

В методе Виолы-Джонса основу составляют примитивы Хаара, представляющие собой разбивку заданной прямоугольной области на наборы разнотипных прямоугольных подобластей:



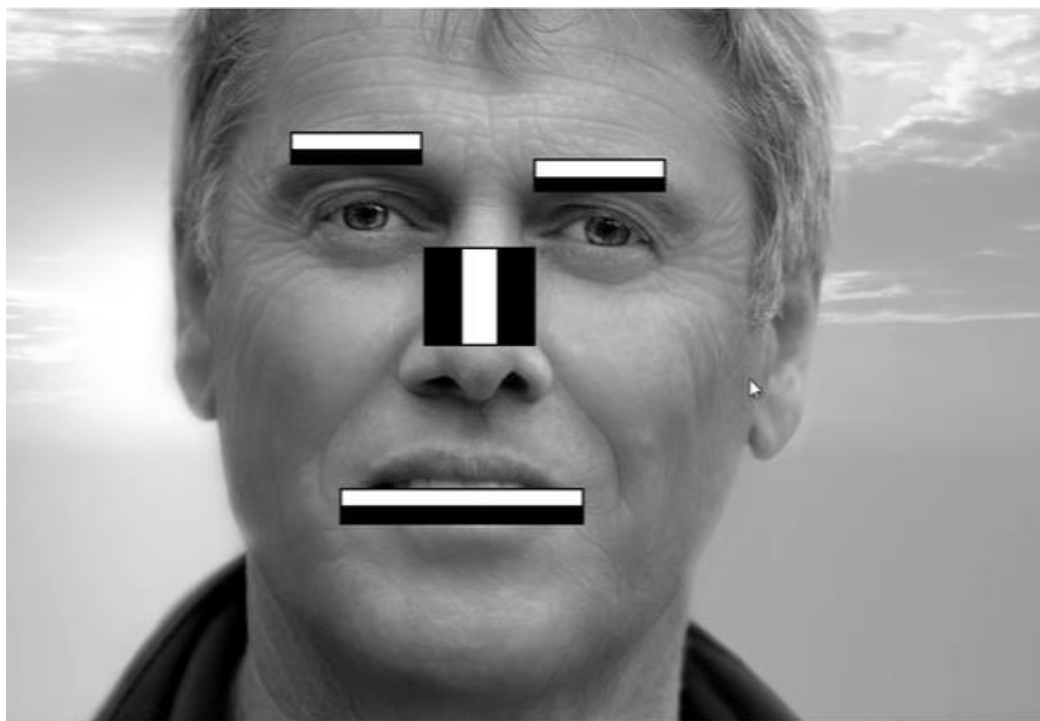
Стандартный метод



Расширенный метод

# Вычисление признаков Хаара

---

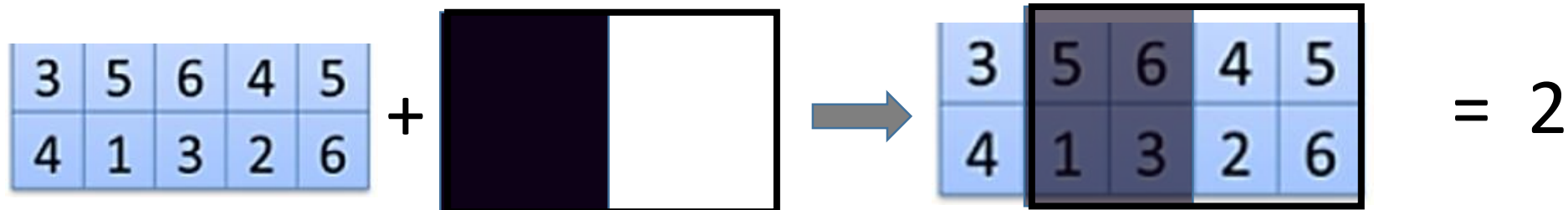


Вычисляемым значением такого признака будет

$$F = X - Y$$

где  $X$  – сумма значений яркостей точек закрываемых светлой частью признака, а  $Y$  – сумма значений яркостей точек закрываемых темной частью признака.

Для их вычисления используется понятие интегрального изображения, рассмотренное выше.





# Обучение классификатора в методе Виолы-Джонса

---

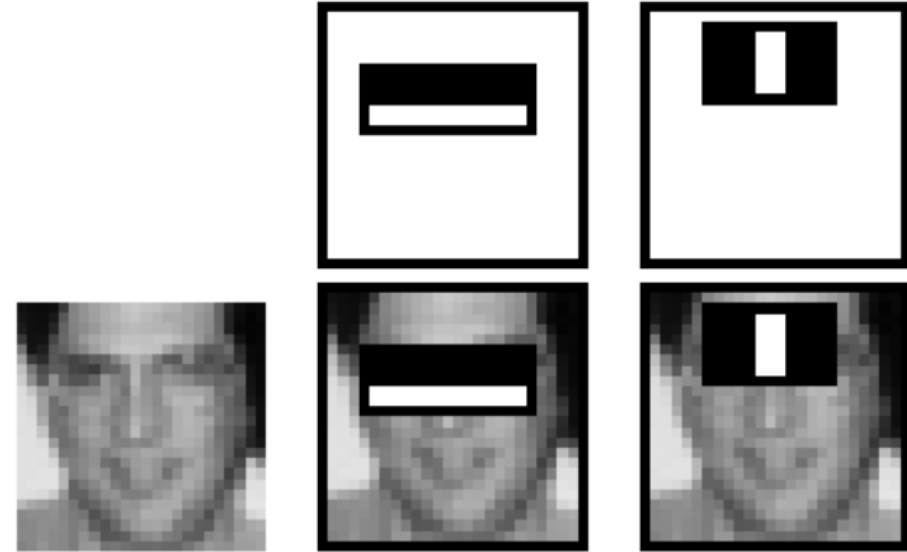
**Бустинг** означает дословно **«усиление» «слабых» моделей** – это процедура последовательного построения композиции алгоритмов машинного обучения, когда каждый следующий алгоритм стремится компенсировать недостатки композиции всех предыдущих алгоритмов.

*Предложена Робертом Шапиром (Scharire) в конце 90-х годов, когда надо было найти решение вопроса о том, чтобы имея множество плохих (незначительно отличающихся от случайных) алгоритмов обучения, получить один хороший. В основе такой идеи лежит построение *цепочки (ансамбля) классификаторов*, который называется **каскадом**, каждый из которых (кроме первого) *обучается на ошибках предыдущего*.*

**AdaBoost** (*adaptive boosting*) (1999) может использовать произвольное число классификаторов и производить обучение на одном наборе примеров, поочередно применяя их на различных шагах.

# Каскад признаков

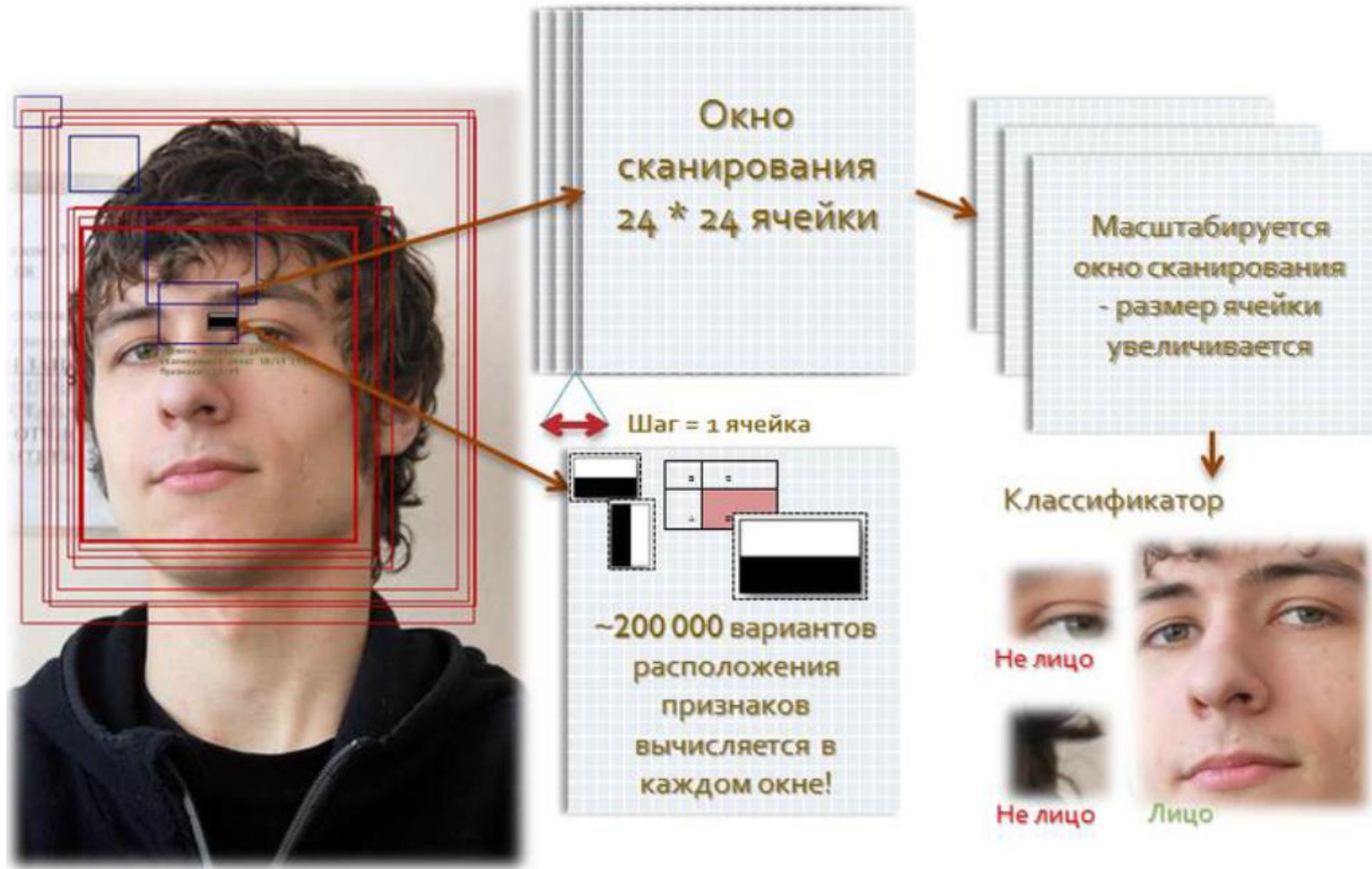
Из двух признаков Хаара строится первый каскад системы по распознаванию лиц, который имеет вполне осмысленную интерпретацию.



Для определения принадлежности к классу в каждом каскаде, находится сумма значений слабых классификаторов этого каскада. Каждый слабый классификатор выдает два значения в зависимости от того больше или меньше заданного порога значение признака, принадлежащего этому классификатору. В конце сумма значений слабых классификаторов сравнивается с порогом каскада и выносятся решения найден объект или нет данным каскадом.

# Алгоритм сканирования окна с признаками

<https://habr.com/ru/post/133826/>



А здесь можно посмотреть визуализацию метода <https://www.youtube.com/watch?v=hPCTwxFOqf4>