Лекция 5

Hough transform. Image segmentation. Eigenfaces. BoVW

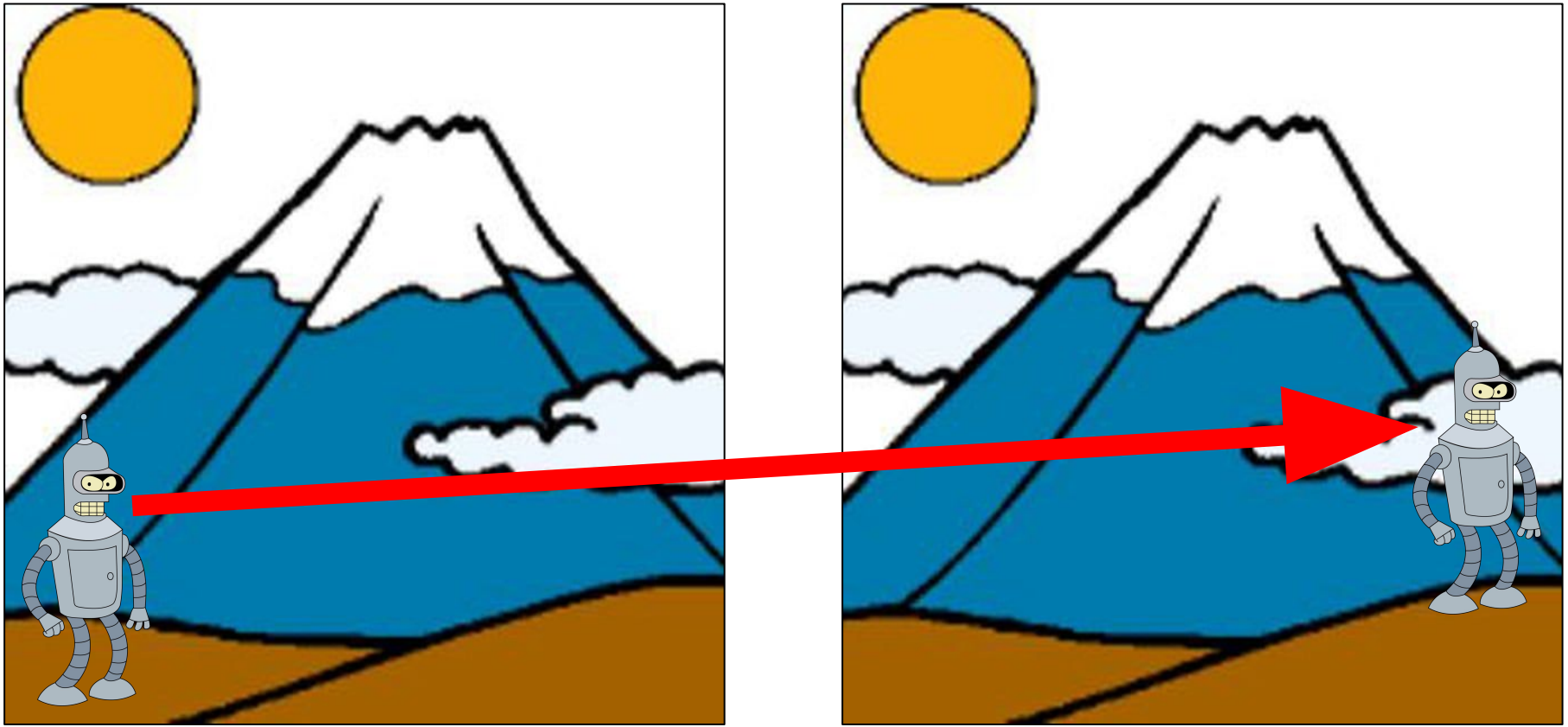
11.02.2020

Руслан Алиев

В предыдущих сериях...

Что такое Optical Flow?

Движение!

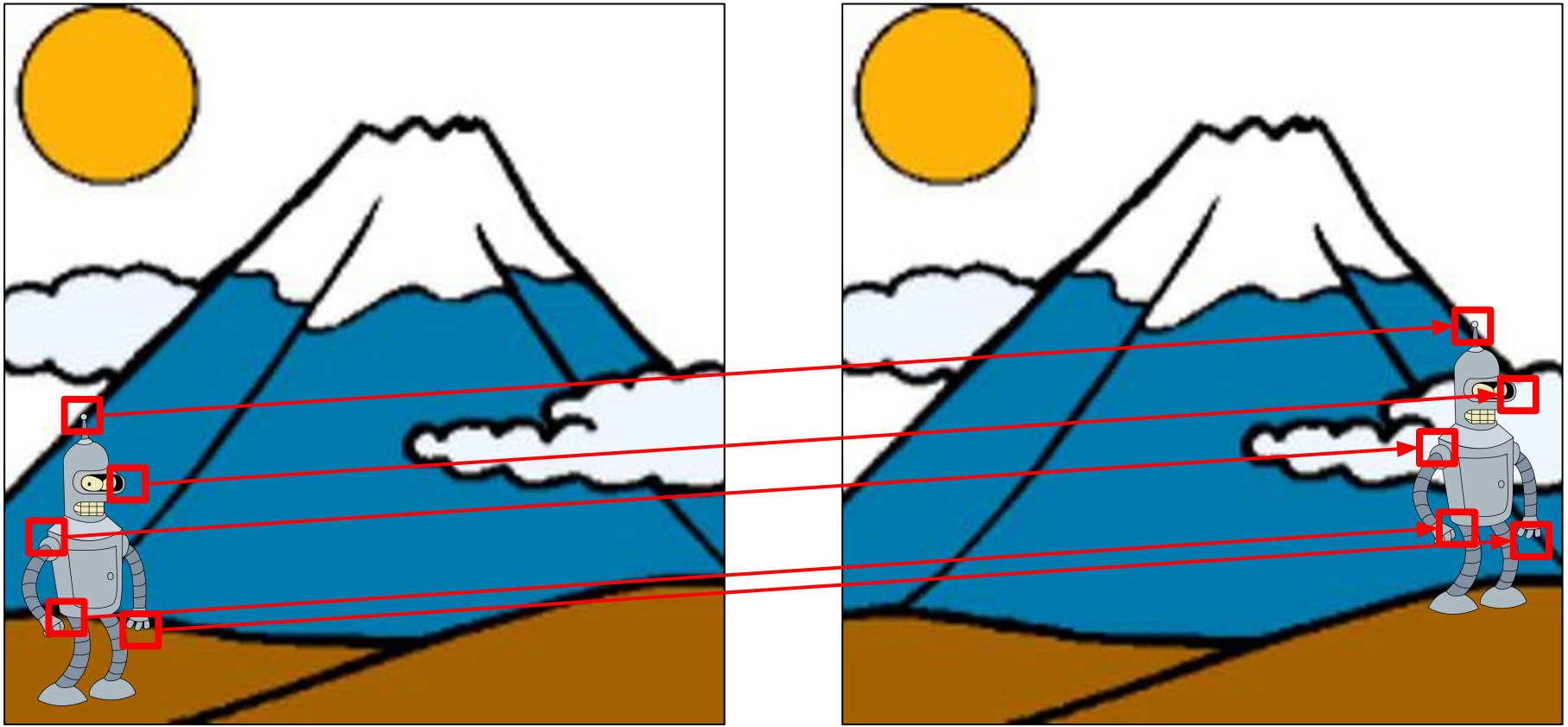


Sparse vs Dense

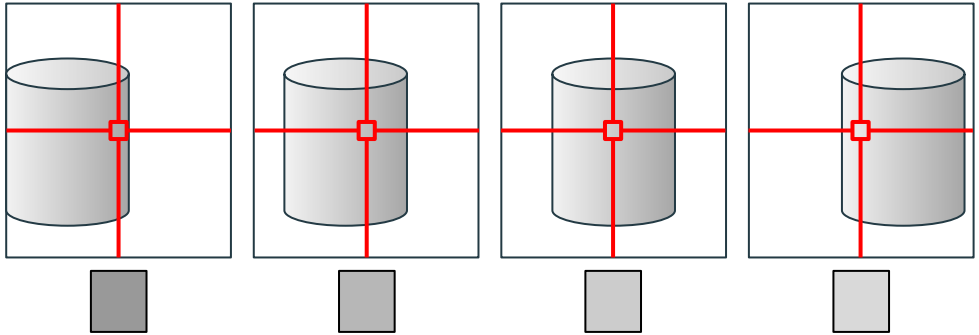
Считает OF для конкретных точек-фичей

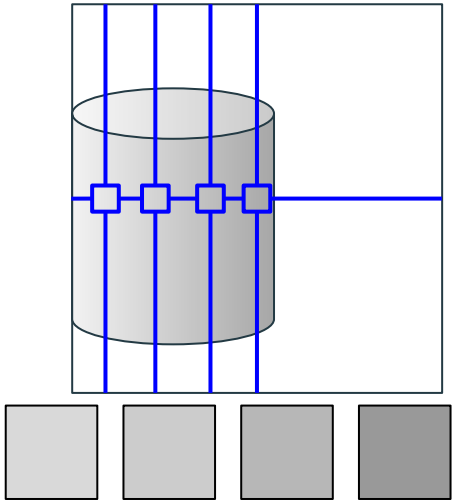
Считает OF для всех пикселей

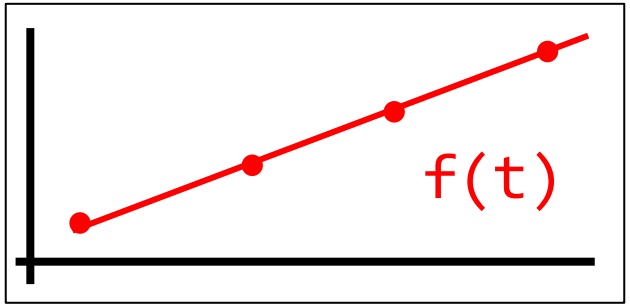
Матчинг признаков

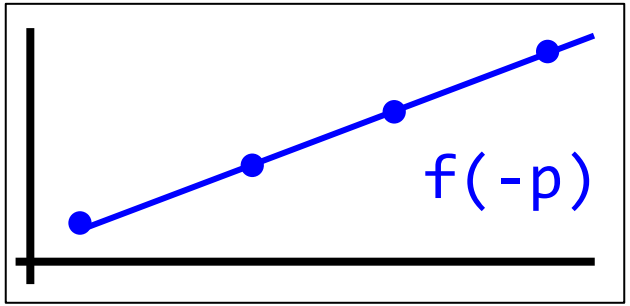


Наблюдение...

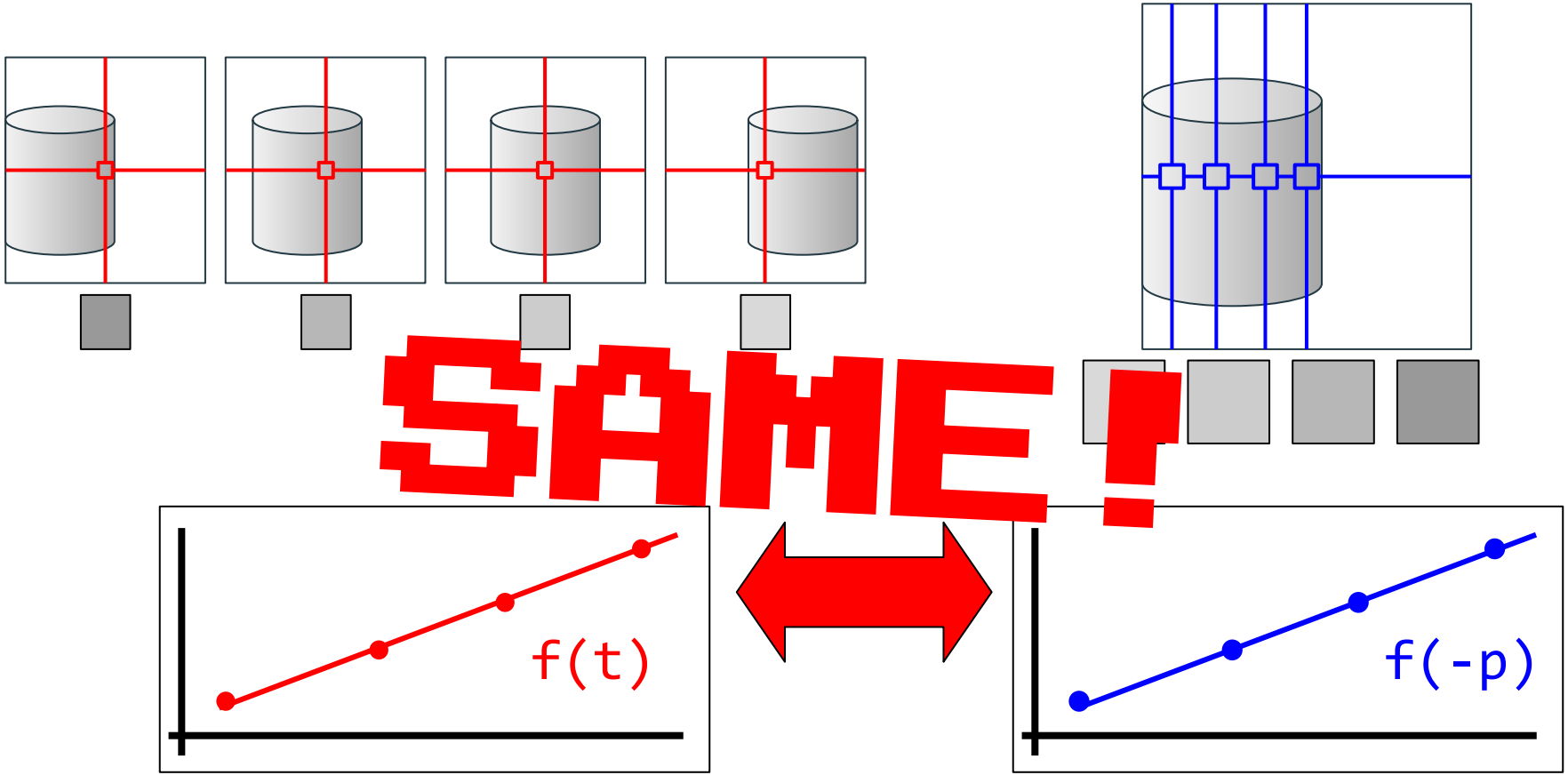








Наблюдение...



Lucas-Kanade Optical Flow

Предполагаем, что объект сместился на $\Delta\mathbf{p}$ за время Δt

$$f(\mathbf{p} - \Delta\mathbf{p}, t) = f(\mathbf{p}, t + \Delta t)$$

Раскладываем в ряд Тейлора до первого порядка

$$m(\mathbf{p} - \Delta\mathbf{p}) + b \approx f(\mathbf{p}, t + \Delta t)$$

Lucas-Kanade Optical Flow

$$m\mathbf{p} - m\Delta\mathbf{p} + b \approx f(\mathbf{p}, t + \Delta t)$$

$$m\mathbf{p} - m\Delta\mathbf{p} + b - f(\mathbf{p}, t) \approx f(\mathbf{p}, t + \Delta t) - f(\mathbf{p}, t)$$

Вспомним, что $f(x, t) = mx + b$

А что это??

$$-m\Delta\mathbf{p} \approx f(\mathbf{p}, t + \Delta t) - f(\mathbf{p}, t)$$

Это то что мы хотим узнать

Это мы знаем

Lucas-Kanade Optical Flow

$$-dx\Delta x + -dy\Delta y \approx f((x,y), t + \Delta t) - f((x,y), t)$$

Немножко изменим обозначения:

$$dx*u + dy*v = I_t[x,y] - I_{t+\Delta t}[x,y]$$

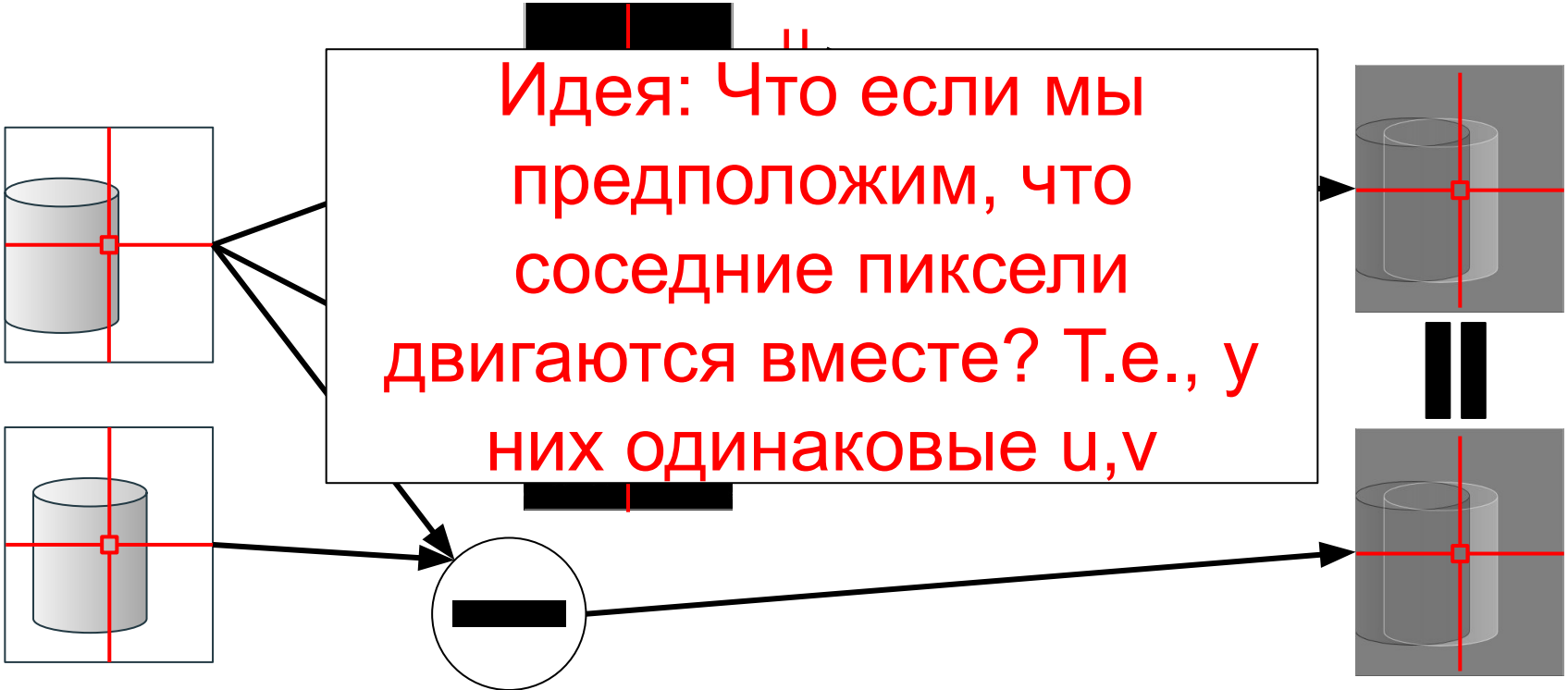
Решим относительно u,v

1 уравнение, 2 неизвестных

:(

Lucas-Kanade Optical Flow

$$dx*u + dy*v = I_t[x,y] - I_{t+\Delta t}[x,y]$$



Lucas-Kanade Optical Flow

$$dx_1 * u + dy_1 * v = I_t[x_1,y_1] - I_{t+\Delta t}[x_1,y_1]$$

$$dx_2 * u + dy_2 * v = I_t[x_2,y_2] - I_{t+\Delta t}[x_2,y_2]$$

$$\ldots$$

$$dx_9 * u + dy_9 * v = I_t[x_9,y_9] - I_{t+\Delta t}[x_9,y_9]$$

9 уравнений, 2 неизвестных -> Overdetermined

Lucas-Kanade Optical Flow

$$S = \begin{bmatrix} dx_1 & dy_1 \\ dx_2 & dy_2 \\ \cdots & \\ dx_9 & dy_9 \end{bmatrix} \qquad \Delta\mathbf{p} = \begin{bmatrix} u \\ v \end{bmatrix} \qquad T = \begin{bmatrix} I_t[x_1, y_1] - I_{t+\Delta t}[x_1, y_1] \\ I_t[x_2, y_2] - I_{t+\Delta t}[x_2, y_2] \\ \cdots \\ I_t[x_9, y_9] - I_{t+\Delta t}[x_9, y_9] \end{bmatrix}$$

$$S\Delta\mathbf{p} = T$$

Least-squares solution

$$||S\Delta\mathbf{p} - T||^2 = 0$$

$$\Delta\mathbf{p} = (S^T S)^{-1} S^T T$$

Lucas-Kanade Optical Flow

Когда алгоритм фейлится?

-Изменения освещенности

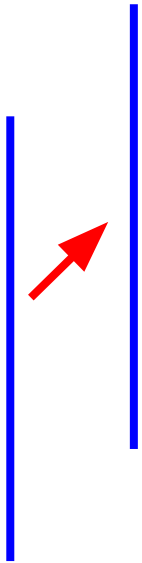
-Большие движения

-Нет хороших фичей

-Проблема апертуры

Lucas-Kanade Optical Flow

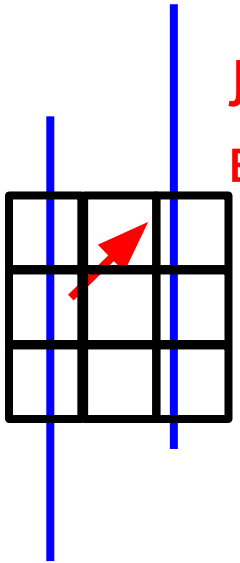
Проблема апертуры



Линия движется
вверх и направо

Lucas-Kanade Optical Flow

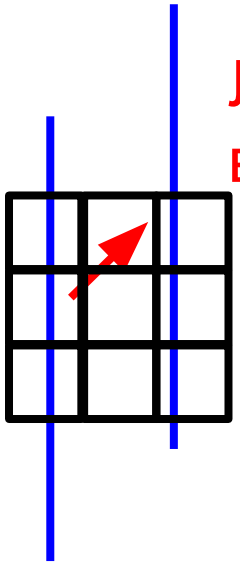
Проблема апертуры



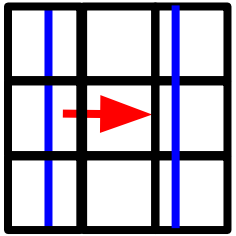
Линия движется
вверх и направо

Lucas-Kanade Optical Flow

Проблема апертуры



Линия движется
вверх и направо



Линия движется
только направо

Lucas-Kanade Optical Flow

Проблема апертуры - Barber pole illusion

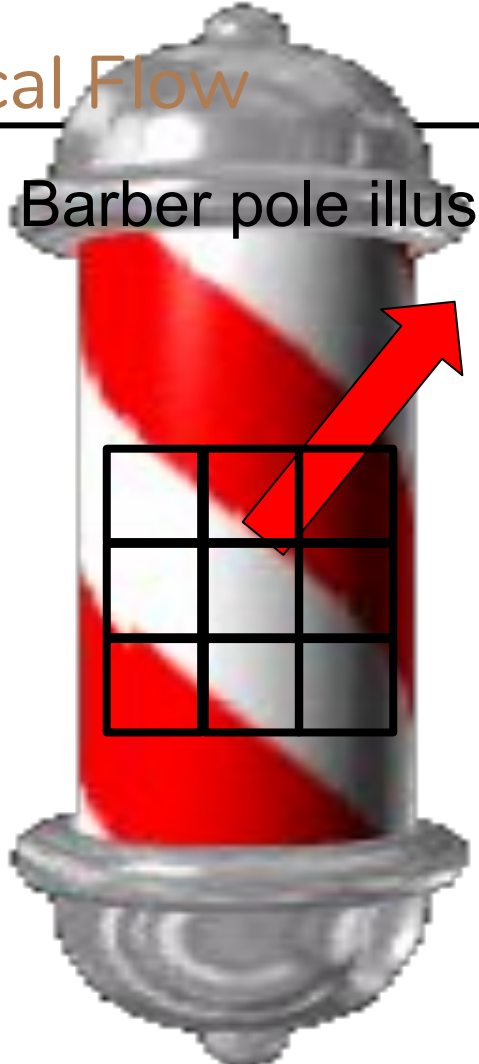
Lucas-Kanade Optical Flow

Проблема апертуры - Barber pole illusion

Движется направо

Lucas-Kanade Optical Flow

Проблема апертуры - Barber pole illusion



Смотрим через окошко - движется вверх и направо

Farnebäck Optical Flow

Мы предположили, объект смещается на $\Delta\mathbf{p}$ за время Δt

$$f(\mathbf{p} - \Delta\mathbf{p}, t) = f(\mathbf{p}, t + \Delta t)$$

давайте аппроксимировать рядом Тейлора 2 порядка

$$(\mathbf{p}-\Delta\mathbf{p})^T A_t(\mathbf{p}-\Delta\mathbf{p}) + b_t(\mathbf{p}-\Delta\mathbf{p}) + c_t = \mathbf{p}^T A_{t+\Delta t}\mathbf{p} + b_{t+\Delta t}\mathbf{p} + c_{t+\Delta t}$$

Параметры полинома из первого изображения

Параметры полинома из второго изображения

Farnebäck Optical Flow

$$b_{t+\Delta t} = b_t - 2A_t\Delta\mathbf{p}$$

Решение для $\Delta\mathbf{p}$

$$\Delta\mathbf{p} = -\tfrac{1}{2}(A_t)^{-1}(b_{t+\Delta t} - b_t)$$

A, b, и c могут быть найдены поточечно используя соседние пиксели как и в предыдущем алгоритма

$$S = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ & \cdots \\ x_k & y_k \end{bmatrix} \quad T = \begin{bmatrix} I[x_1,y_1] \\ I[x_2,y_2] \\ \cdots \\ I[x_k,y_k] \end{bmatrix}$$

$$SAS^T + bS^T + c = T$$

$$||SAS^T + bS^T + c - T||^2 = 0$$

Quadratic Regression

Farnebäck Optical Flow

$$b_{t+\Delta t} = b_t - 2A_t\Delta\mathbf{p}$$

Попробуем другой подход:

$$A\Delta\mathbf{p} = \Delta b$$

Предполагаем, что соседи двигаются вместе

$$A_i\Delta\mathbf{p} = \Delta b_i$$

МНК

$$\Sigma_i\, w_i ||A_i\Delta\mathbf{p} - \Delta b_i||^2$$

Гауссовы веса (мы больше учитываем центральные пиксели чем краевые)

Farnebäck Optical Flow

Все еще предполагаем, что $\Delta\mathbf{p}$ одинаковая для всех соседних пикселей, хотя на самом деле это не так...

ПРЕДПОЛОЖИМ: что $\Delta\mathbf{p}$ - преобразование которое выглядит как:

$$\Delta\mathbf{p} = \begin{bmatrix} a_1 + a_2x + a_3y + a_7x^2 + a_8xy \\ a_4 + a_5x + a_6y + a_7xy + a_8y^2 \end{bmatrix}$$

$$\Delta\mathbf{p} = S\mathbf{d}$$

$$S = \begin{bmatrix} 1 & x & y & 0 & 0 & 0 & x^2 & xy \\ 0 & 0 & 0 & 1 & x & y & xy & y^2 \end{bmatrix}$$

$$\mathbf{d} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \end{bmatrix}$$

Farnebäck Optical Flow

$$\Sigma_i\, w_i \,||A_i \Delta \mathbf{p} - \Delta b_i||^2$$

Подставлем S**d** вместо $\Delta\mathbf{p}$.

$$\Sigma_i\, w_i \,||A_i S\mathbf{d} - \Delta b_i||^2$$

Решаем относительно **d**

$$\mathbf{d} = \left(\Sigma_i w_i (S_i)^T (A_i)^T S_i\right)^{-1} \Sigma_i w_i (S_i)^T (A_i)^T \Delta b_i$$

Такая же проблема обратимости, но на практике проблем меньше

Как мы очущаем глубину?

-Бинокулярное схождение

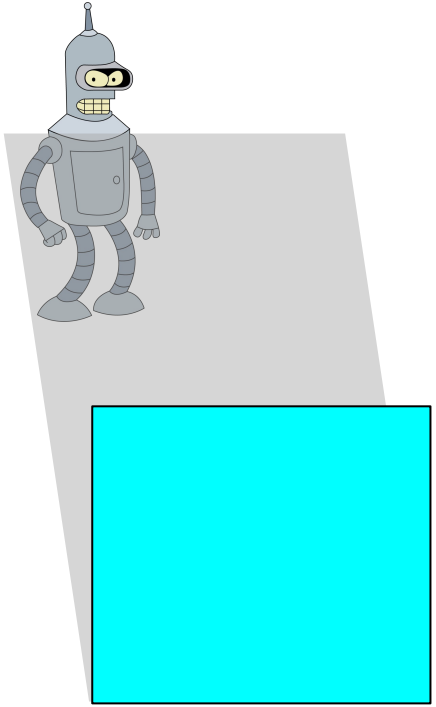
-Бинокулярный параллакс

-Монокулярный параллакс

-”Подсказки”

- Перекрытие

-Тени



Стерео

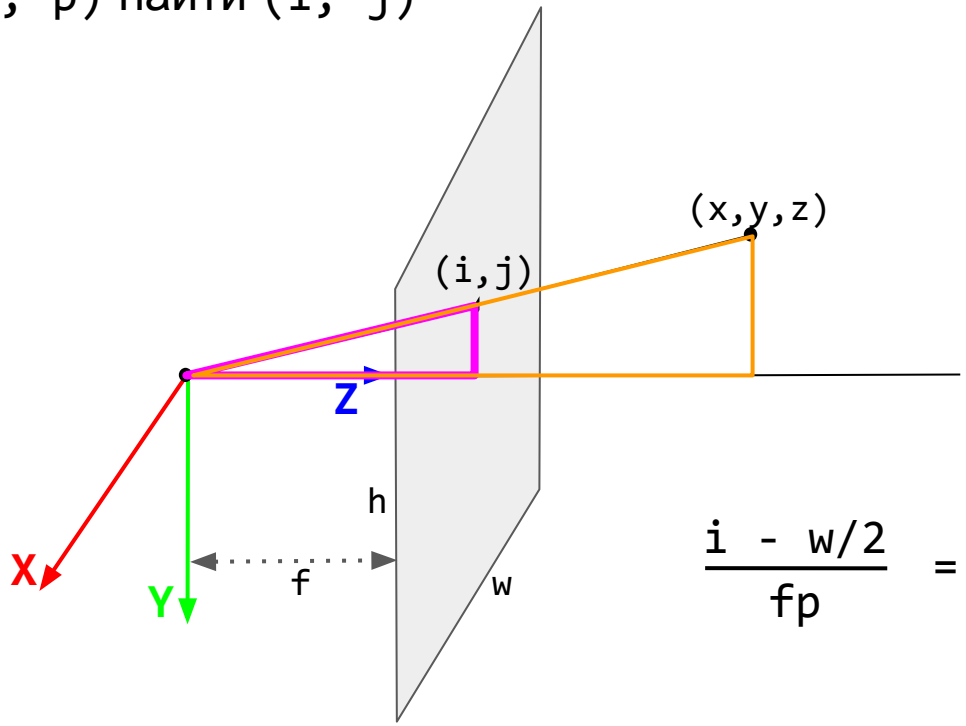
Смещение обратно пропорционально расстоянию от камеры

Проекция изображения

Имея (x, y, z) и (f, w, h, p) найти (i, j)

$$i = \frac{fpx}{z} + \frac{w}{2}$$

$$j = \frac{fpy}{z} + \frac{h}{2}$$



$$\frac{i - w/2}{fp} = \frac{x}{z}$$

Проекция изображения

Имея (x, y, z) и (f, w, h, p) найти (i, j)

$$i = \frac{fpx}{z} + \frac{w}{2}$$

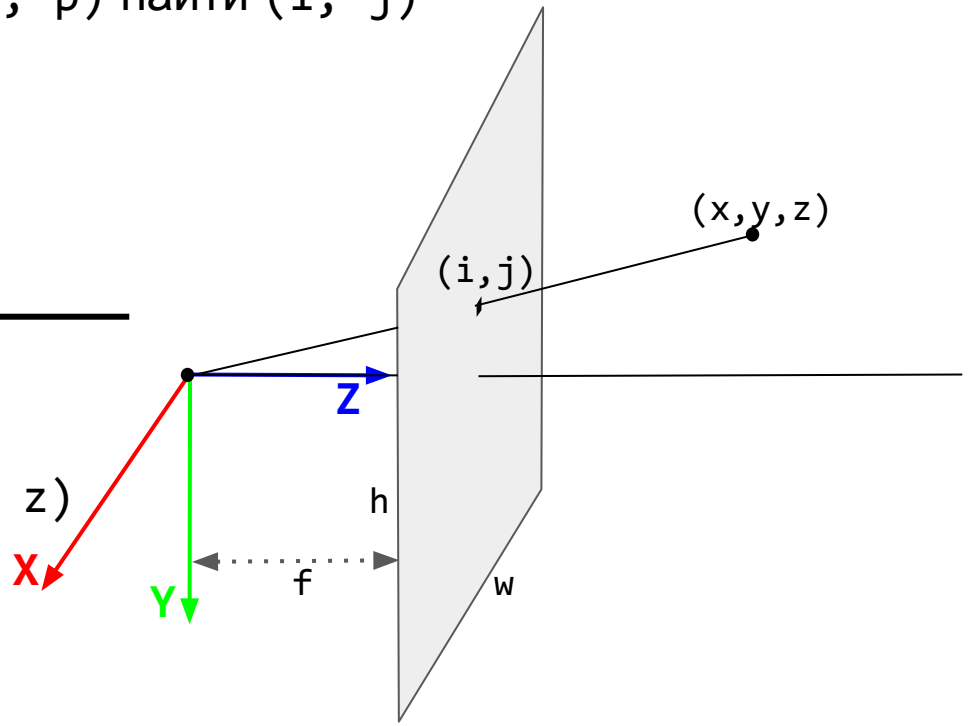
$$j = \frac{fpy}{z} + \frac{h}{2}$$

Имея (i, j) и (f, w, h, p) найти (x, y, z)

x = ?

y = ?

z = ?



Проекция изображения

Имея (x, y, z) и (f, w, h, p) найти (i, j)

$$i = \frac{fpx}{z} + \frac{w}{2}$$

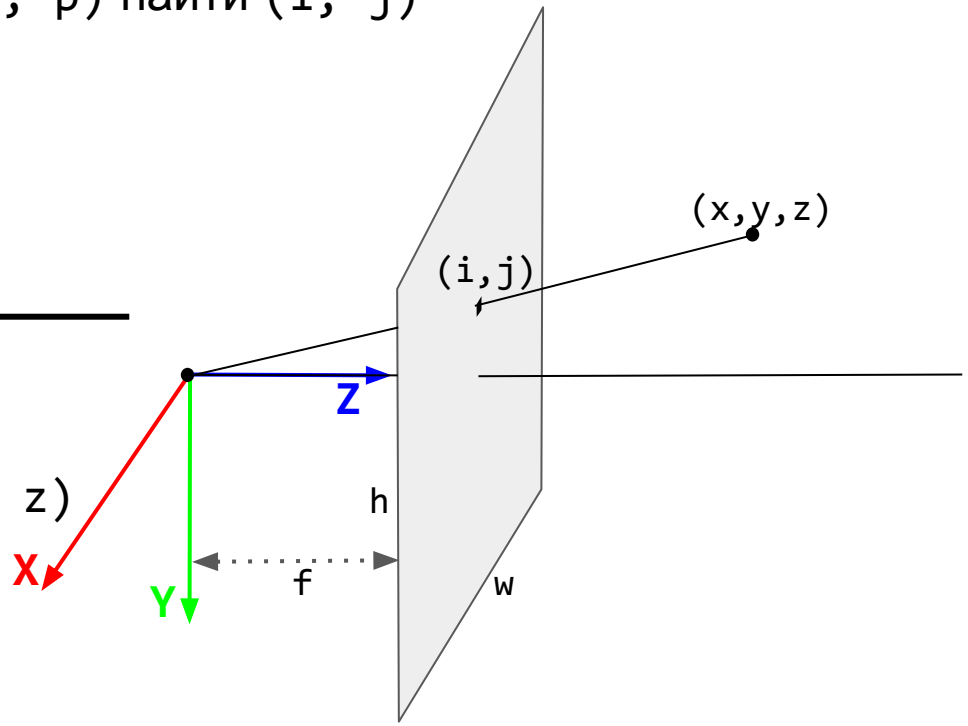
$$j = \frac{fpy}{z} + \frac{h}{2}$$

Имея (i, j, d) и (f, w, h, p) найти (x, y, z)

$$x = \frac{(i - w/2)d}{fp}$$

$$y = \frac{(j - h/2)d}{fp}$$

$$z = d$$



Нормализованные координаты изображения

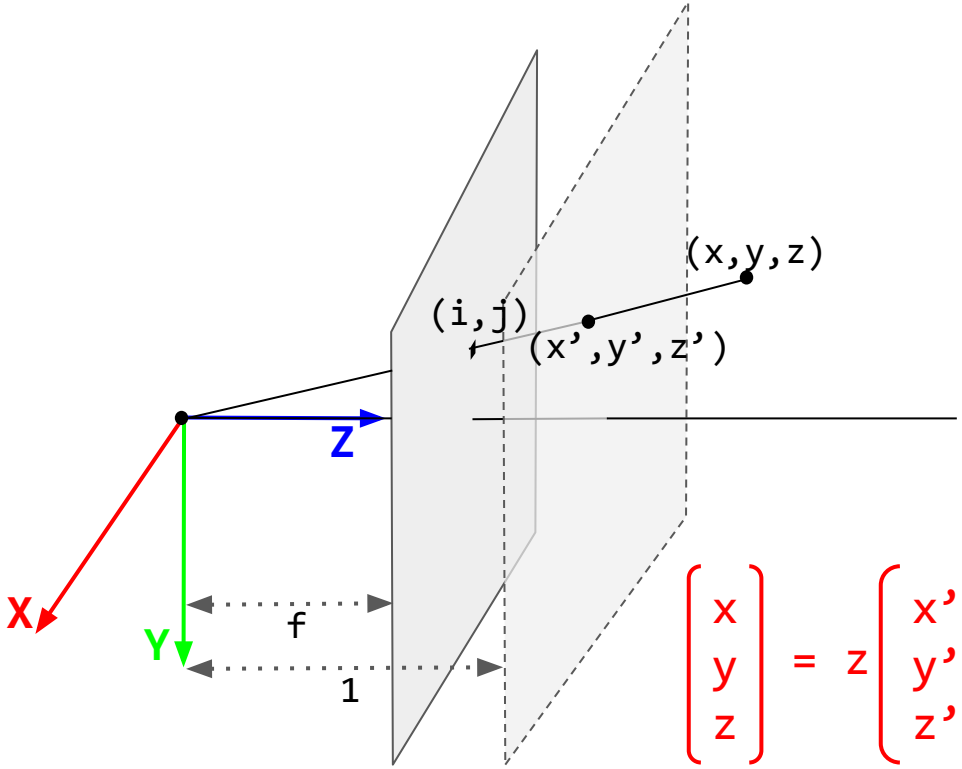
$$i = fpx' + \frac{w}{2}$$

$$j = fpy' + \frac{h}{2}$$

$$x' = \frac{x}{z}$$

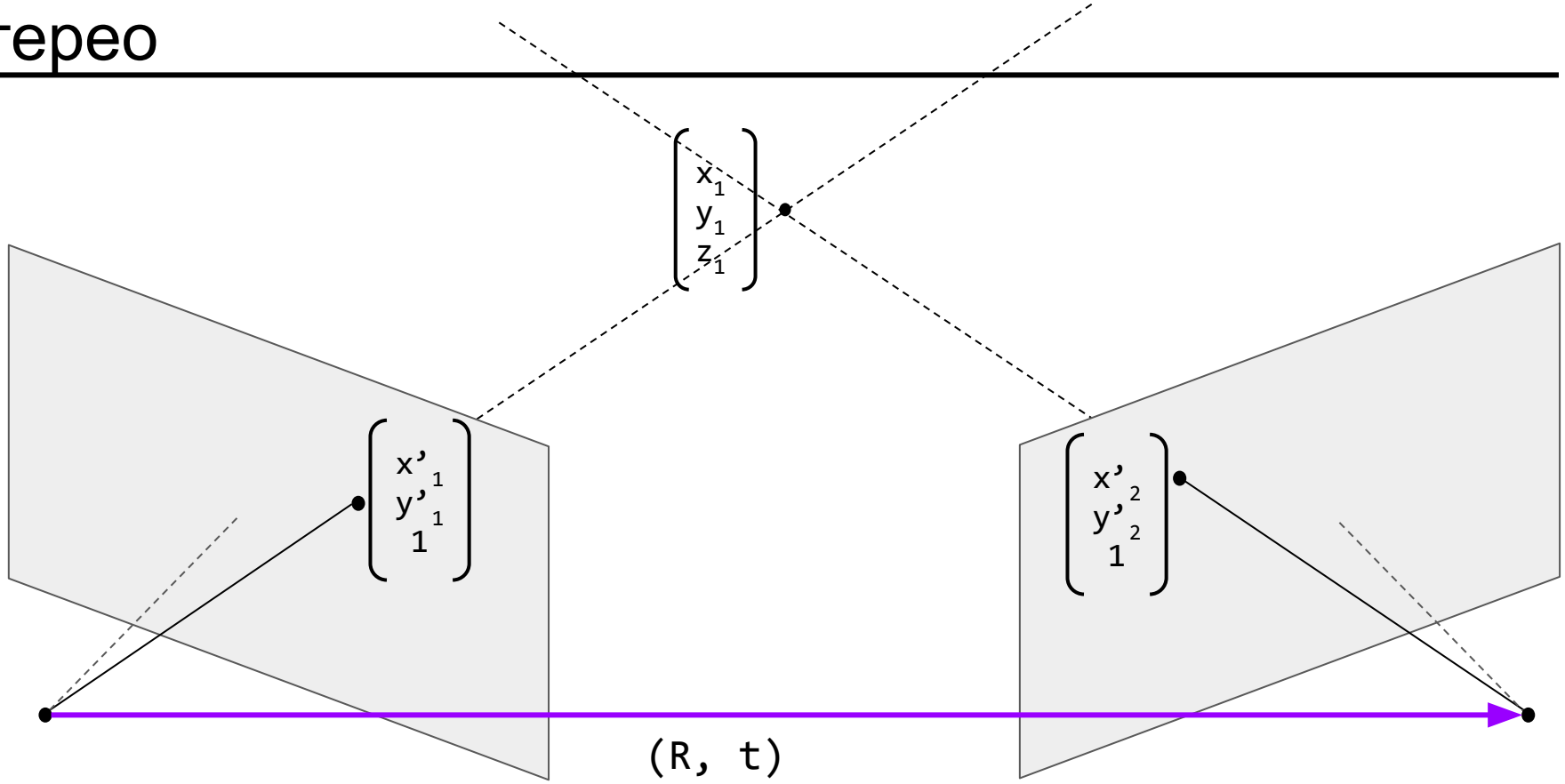
$$y' = \frac{y}{z}$$

$$z' = 1$$



$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = z \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$$

Стерео



Камера 1

Камера 2

Стерео

Линия в 3D может быть представлена как:

Линия 1

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + z_1 \begin{pmatrix} x'_1 \\ y'_1 \\ 1 \end{pmatrix}$$

Линия 2

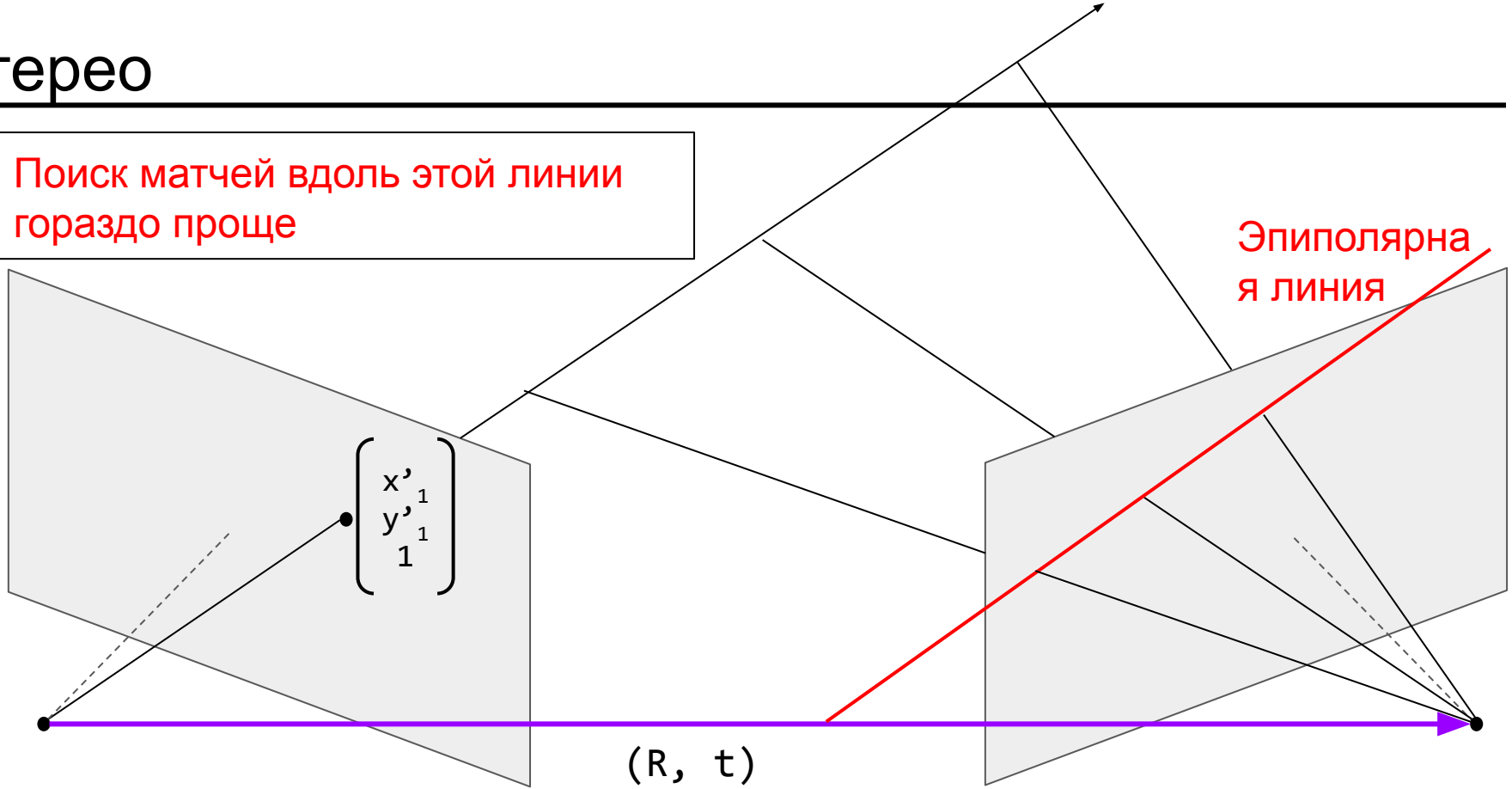
$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + z_2 \begin{pmatrix} x'_2 \\ y'_2 \\ 1 \end{pmatrix} \xrightarrow{\text{Преобразование к координатам Камеры 1}} -t + R^{-1} z_2 \begin{pmatrix} x'_2 \\ y'_2 \\ 1 \end{pmatrix}$$

Решаем системы линейных уравнений

$$z_1 \begin{pmatrix} x'_1 \\ y'_1 \\ 1 \end{pmatrix} = -t + R^{-1} z_2 \begin{pmatrix} x'_2 \\ y'_2 \\ 1 \end{pmatrix}$$

Стерео

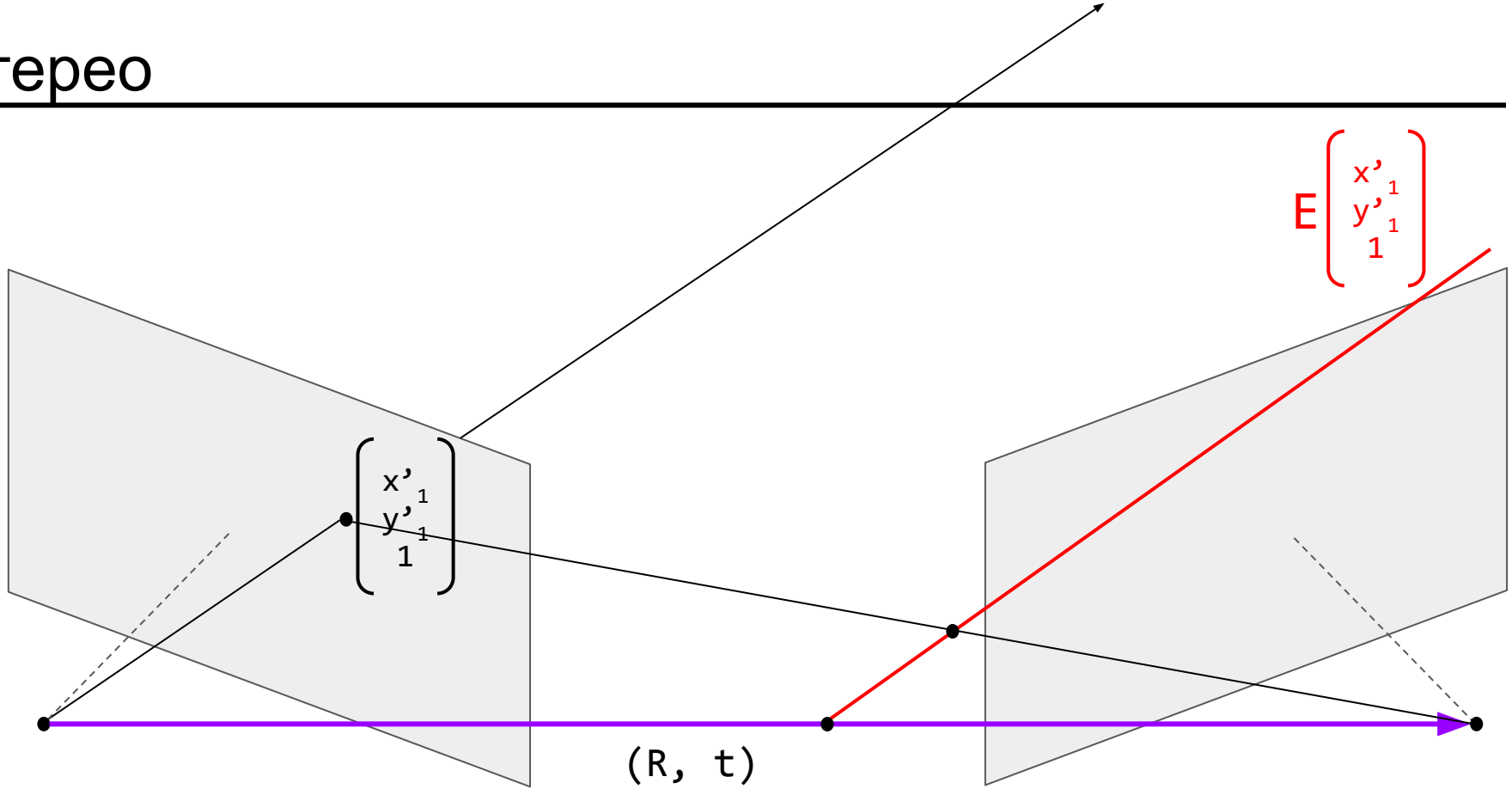
Поиск матчей вдоль этой линии гораздо проще



Камера 1

Камера 2

Стерео

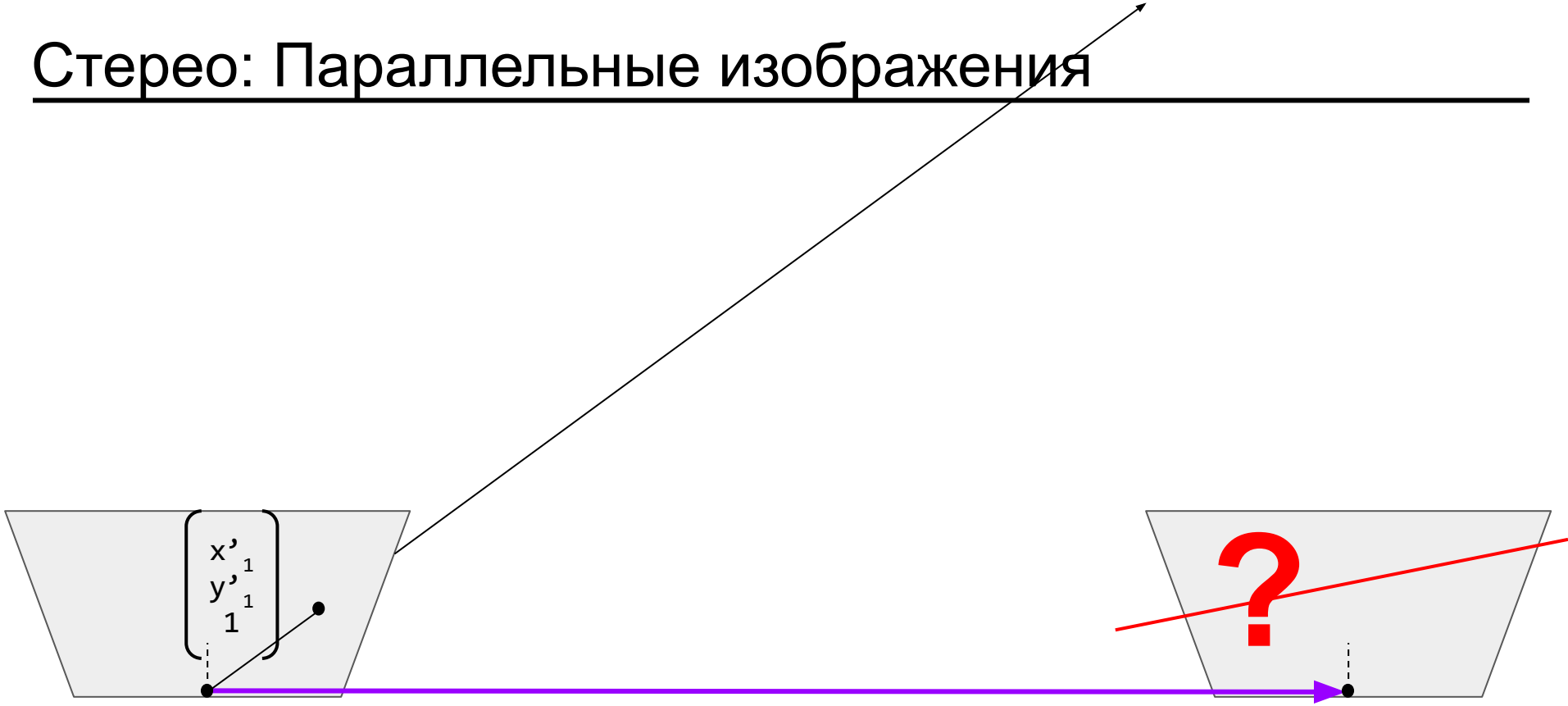


Камера 1

Камера 2

Стерео

Стерео: Параллельные изображения



(I, [Δx, 0, 0])

Камера 1

Камера 2

Стерео: Параллельные изображения

$$\begin{pmatrix} \Delta x \\ 0 \\ 0 \end{pmatrix} \times I \begin{pmatrix} x'_1 \\ y'_1 \\ 1 \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

$$\begin{pmatrix} \Delta x \\ 0 \\ 0 \end{pmatrix} \times \begin{pmatrix} x'_1 \\ y'_1 \\ 1 \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

$$\begin{pmatrix} 0 \\ -\Delta x \\ \Delta x y'_1 \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

$$(0)x + (-\Delta x)y + (\Delta x y'_1) = 0 \longrightarrow y = y'_1$$

Стерео: Параллельные изображения

$$\frac{x_1}{z} = \frac{x'_1}{1}$$

$$x_1 = x'_1 z$$

$$x'_1 z = x'_2 z + \Delta x$$

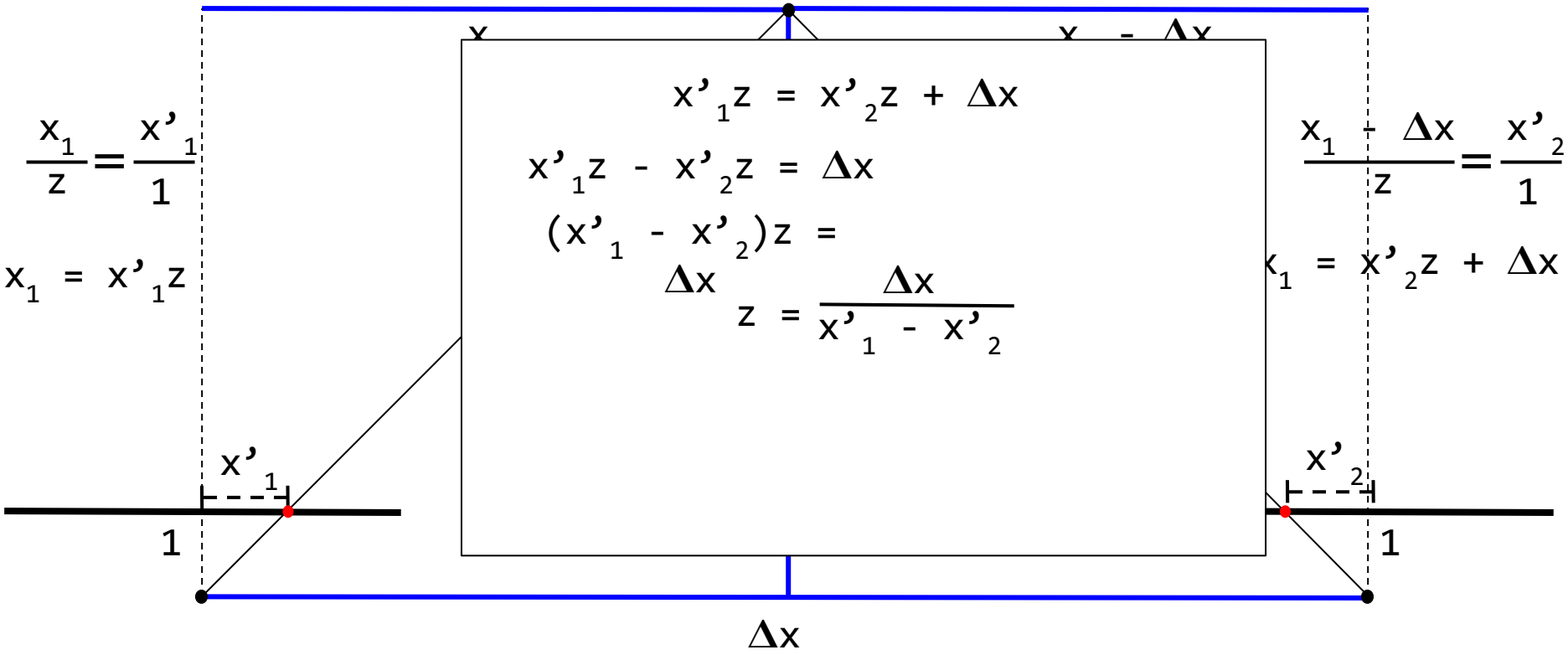
$$x'_1 z - x'_2 z = \Delta x$$

$$(x'_1 - x'_2) z = \Delta x$$

$$z = \frac{\Delta x}{x'_1 - x'_2}$$

$$\frac{x_1 + \Delta x}{z} = \frac{x'_2}{1}$$

$$x_1 = x'_2 z + \Delta x$$



Камера 1

Камера 2

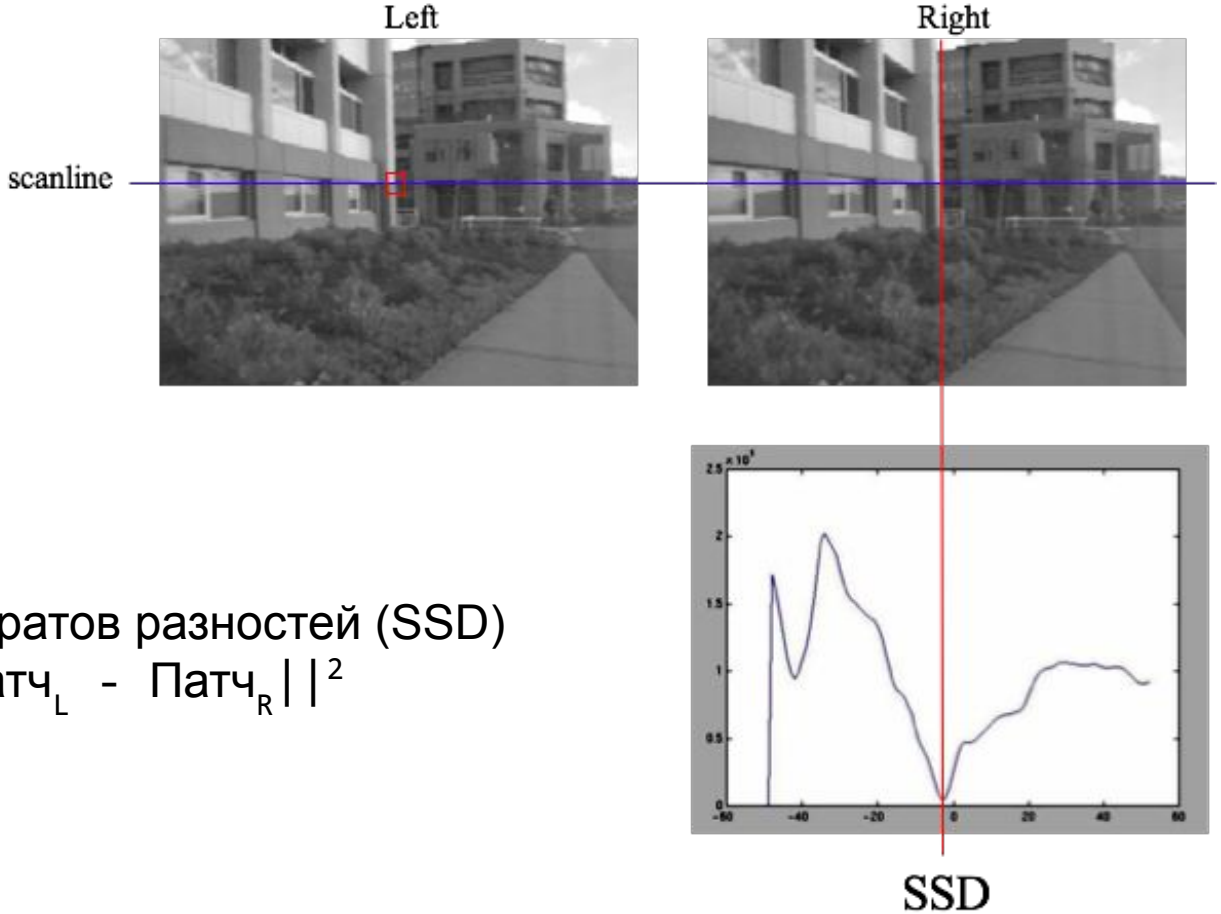
Стерео: Параллельные изображения

Алгоритм (одинаковые камеры)

```
compute_depth(I1, I2, Δx, f, p):
    depth <- new_image_like(I1)
    For every pixel i,j in I1:
        fij = get_feature_descriptor(I1, i, j)
        best_score <- INFINITY
        best_i <- NaN
        For i' in columns(I2):
            fi'j = get_feature_descriptor(I2, i', j)
            score = score_feature_match(fij, fi'j)
            If score < best_score:
                best_score = score
                best_i = i'
        depth[i, j] = fpΔx/(i - best_i)
    Return depth
```

Что это такое?

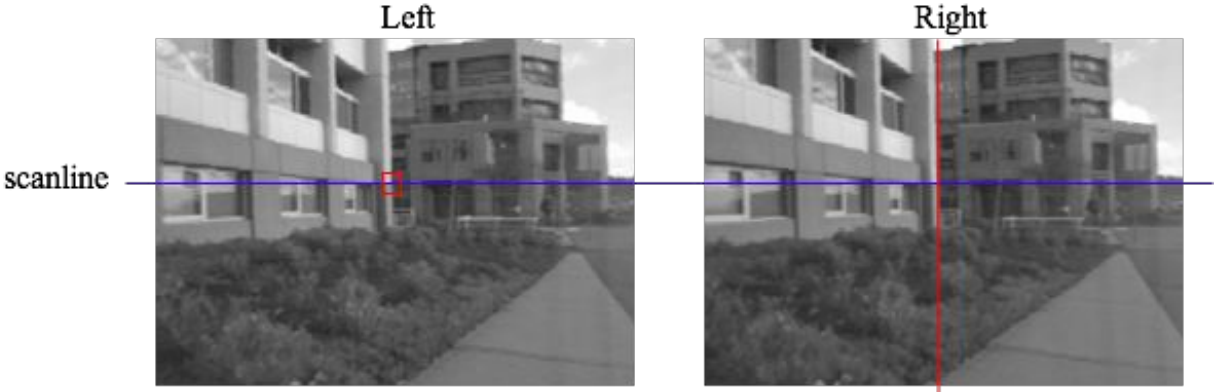
Матчинг признаков



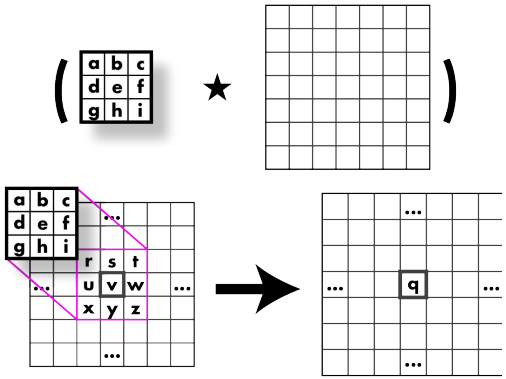
Сумма квадратов разностей (SSD)

$$||\text{Патч}_L - \text{Патч}_R||^2$$

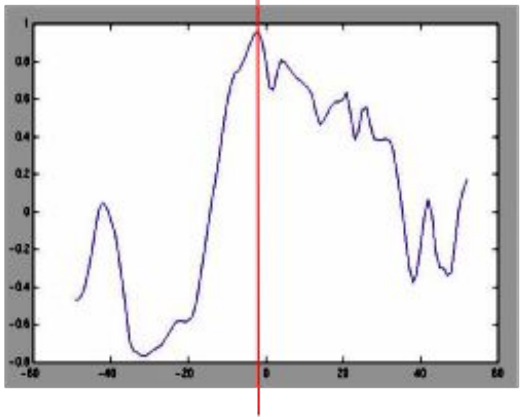
Матчинг признаков



Кросс корреляция



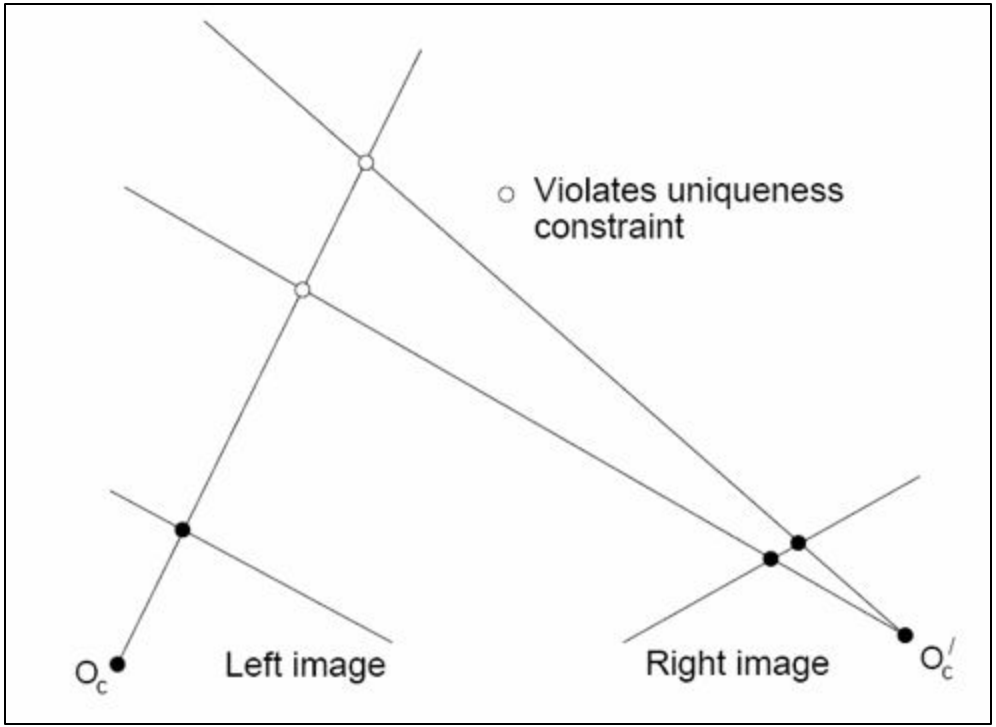
$q = a \times r + b \times s + c \times t + d \times u + e \times v + f \times w + g \times x + h \times y + i \times z$



Norm. corr

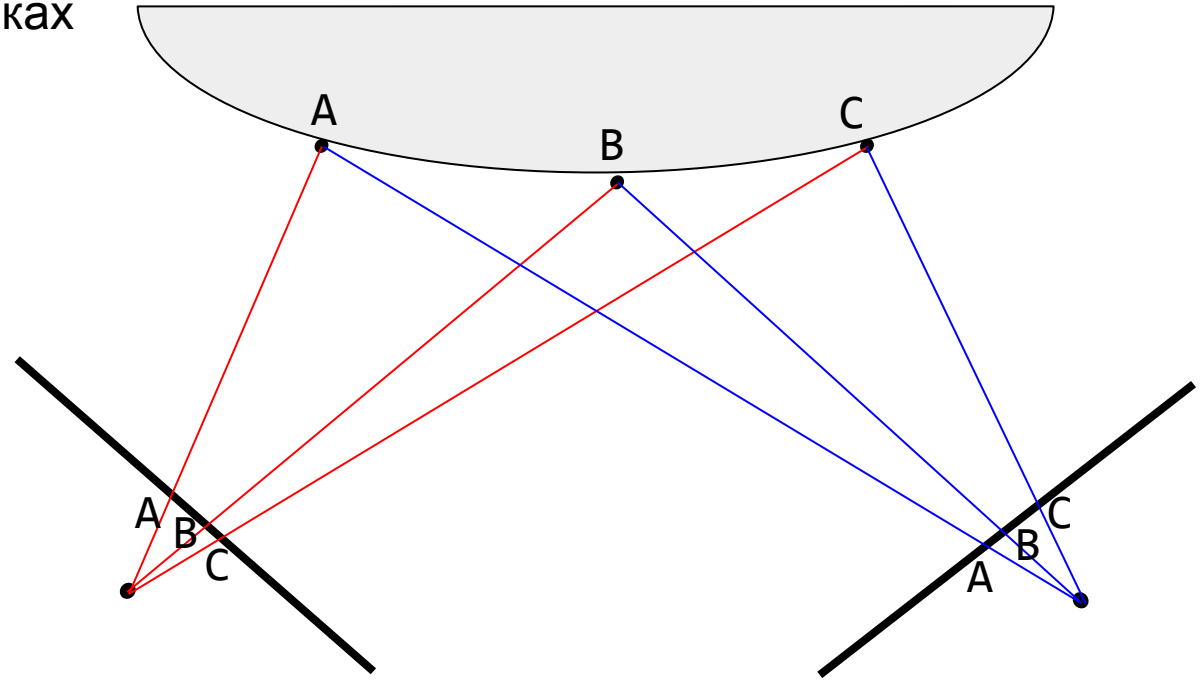
Стерео ограничения

- <u>Уникальность:</u> Для каждой точки одного изображения, может быть только одна точка-матч из другого изображения



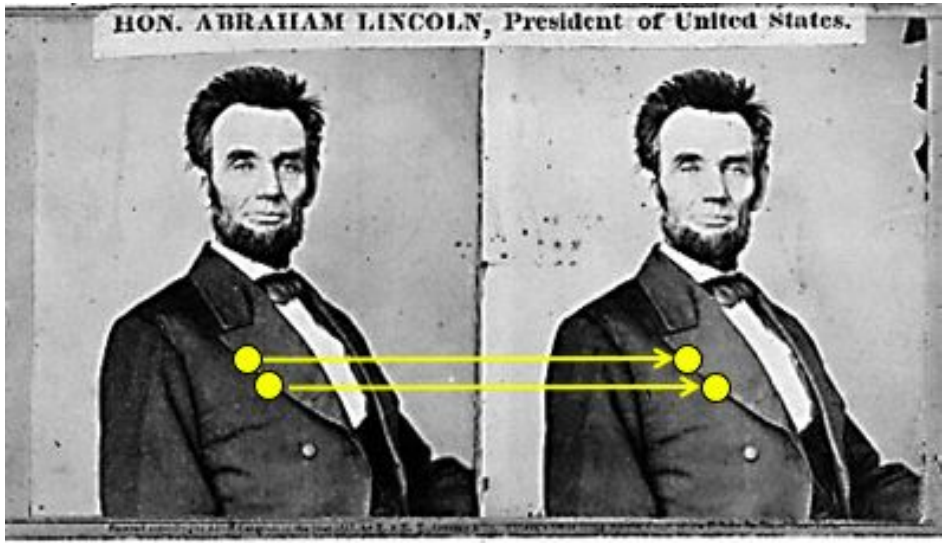
Стерео ограничения

- <u>Уникальность:</u> Для каждой точки одного изображения, может быть только одна точка-матч из другого изображения
- <u>Упорядочивание:</u> Точки должны быть в одном и том же порядке на обоих картинках



Стерео ограничения

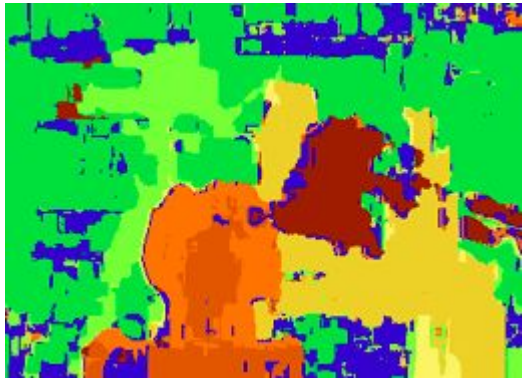
- <u>Уникальность:</u> Для каждой точки одного изображения, может быть только одна точка-матч из другого изображения
- <u>Упорядочивание:</u> Точки должны быть в одном и том же порядке на обоих картинках
- <u>Плавность</u>: Близкие пиксели смещают недалеко друг от друга



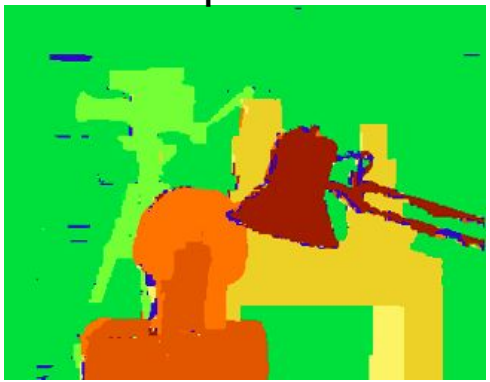
Разрез графа

Y. Boykov, O. Veksler, and R. Zabih, Fast Approximate Energy Minimization via Graph Cuts, PAMI 2001

Before



Graph Cuts



Ground Truth

Hough transform

Детектор границ Кэнни

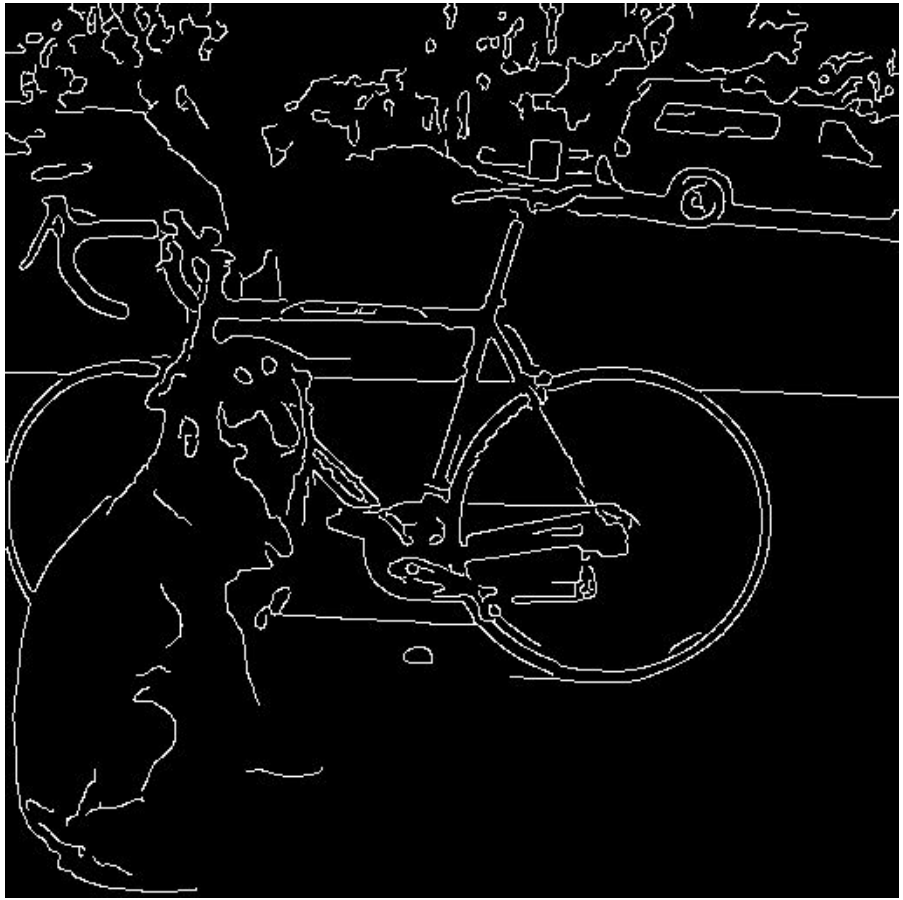
- Ваш первый image processing pipeline!
  - Олдскульное CV целиком состоит из пайплайнов

Алгоритм:

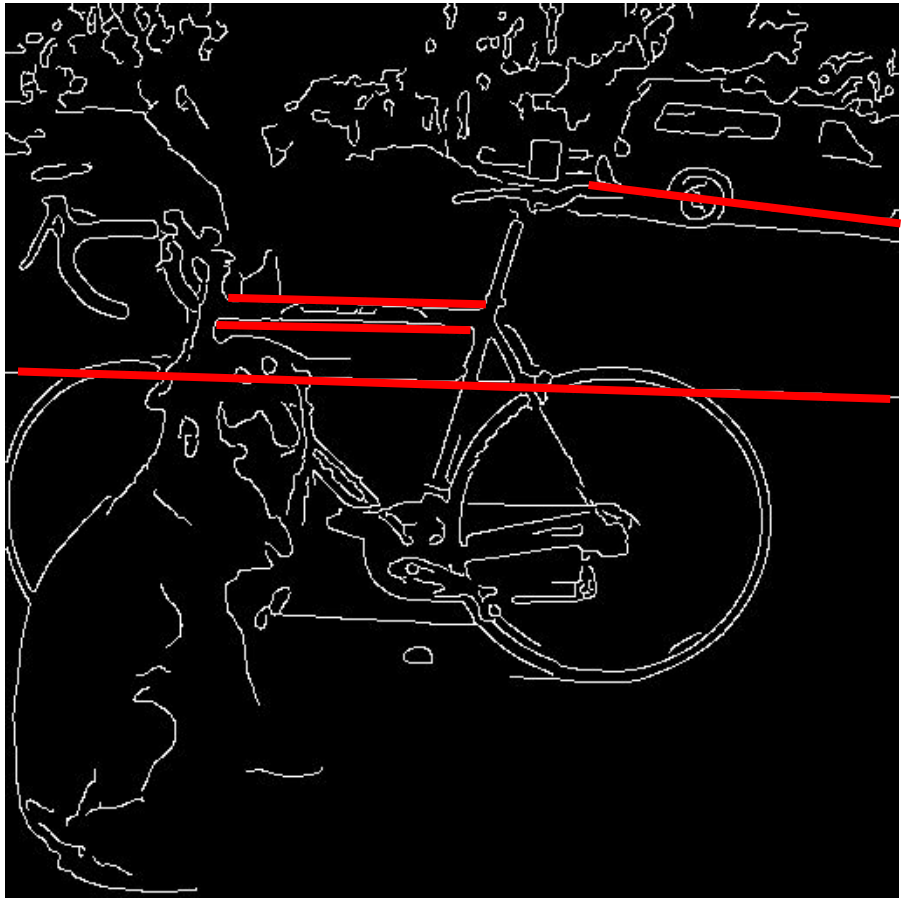
- Сгладить изображение
- Посчитать направление и магнитуду градиентов
- Non-maximum suppression
- Двойной трешхолдинг (strong, weak edges)
- Соединение компонент (гистерезис)

http://bigwww.epfl.ch/demo/ip/demos/edgeDetector/

Детектор Границ Кэнни

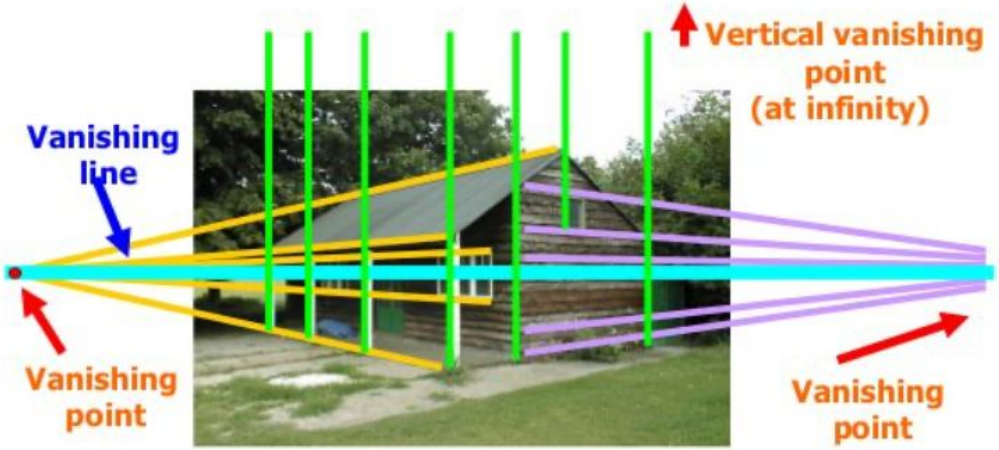


Детектор Границ Кэнни



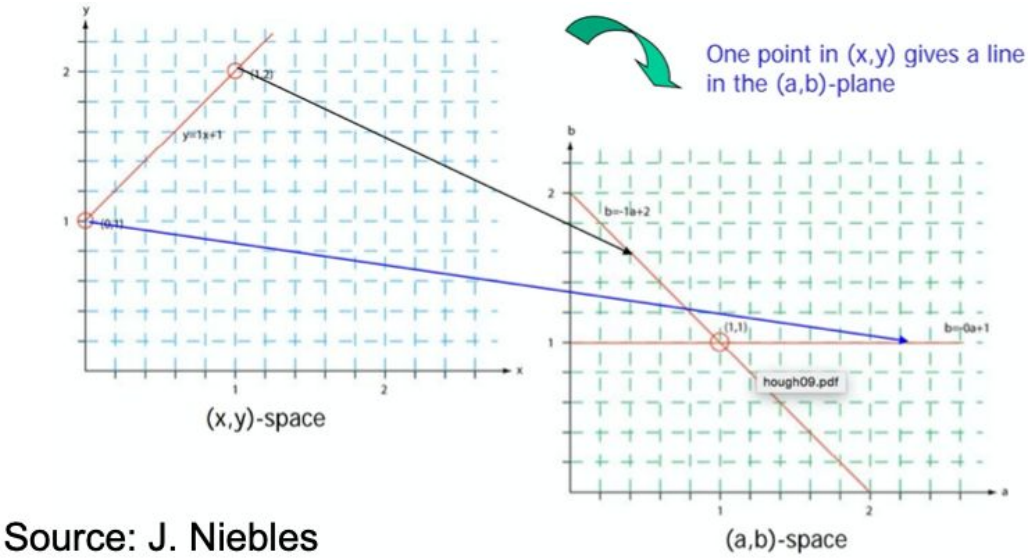
Hough transform

- Нахождений линий на изображении!
- Позволяет находить: ключевые точки (пересечение линий), геометрию изображения, 3D reconstruction
- Можно находить не только линии, но и любые другие параметрически задаваемые фигуры



Hough transform

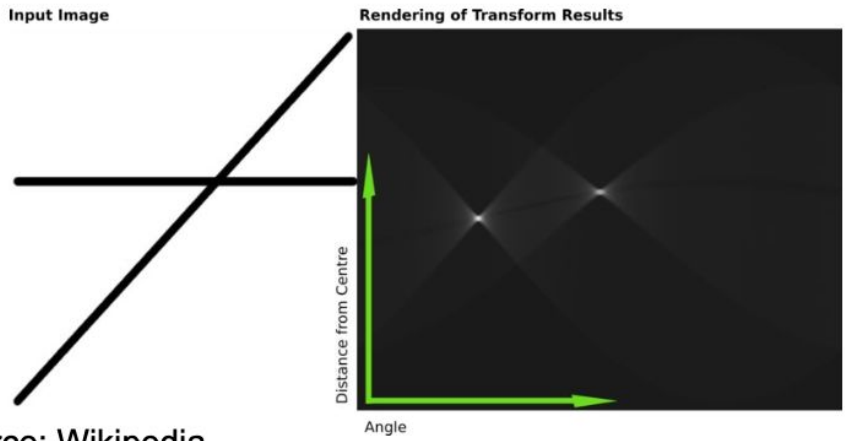
- Рассмотрим некоторую точку **(x_i; y_i) .** Прямые линии проходящие через эту точку удовлетворяют уравнению: **$y_i = a * x_i + b$**
- Уравнение может быть переписано как: : **$b = - a * x_i + y_i$**
- => Точка **(x_i; y_i)** задает линию в пространстве **(a, b)**



Source: J. Niebles

Hough transform

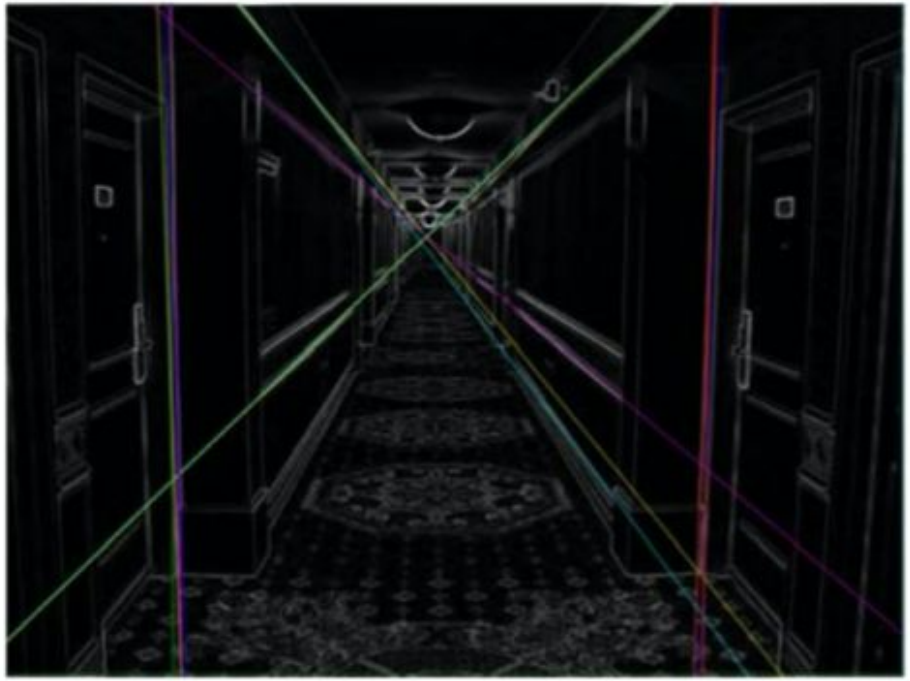
- Две точки **(x_1; y_1)** и **(x_2; y_2)** задают линию в пространстве **(x,y)**
- Эти две точки задают две различные линии в пространстве **(a,b)**
- В пространстве **(a,b)** эти линии пересекаются в точке **(A,B)**
- Все точки на линии задаваемой точками **(x_1; y_1)** и **(x_2; y_2)** порождают линии в пространстве **(a,b)**, пересекающиеся в точке **(A,B)**



Source: Wikipedia

Hough transform

Hough transform

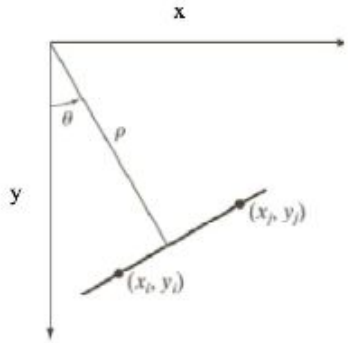


Hough transform

- Мы можем представить линию в полярных координатах вместо $y = a*x + b$
- Уравнение линии в полярных координатах: $x*\cos\theta + y*\sin\theta = \rho$
- Какое соотношение между – (x y) и $(\rho\ \theta)$?

Hough transform

- Линия в пространстве (x,y) - не линия в пространстве (ρ θ)
- Вертикальная линия: θ = 90, ρ = пересечение с осью x
- Горизонтальная линия: θ = 0, ρ = пересечение с осью y



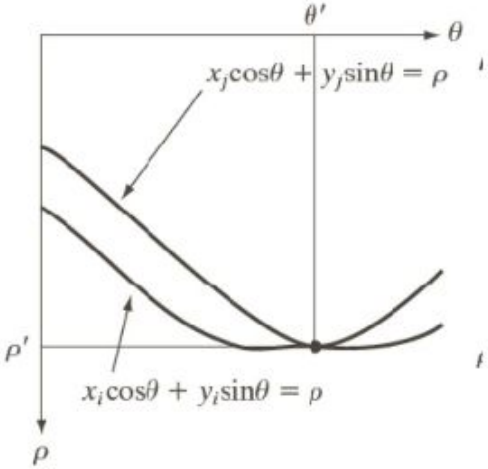
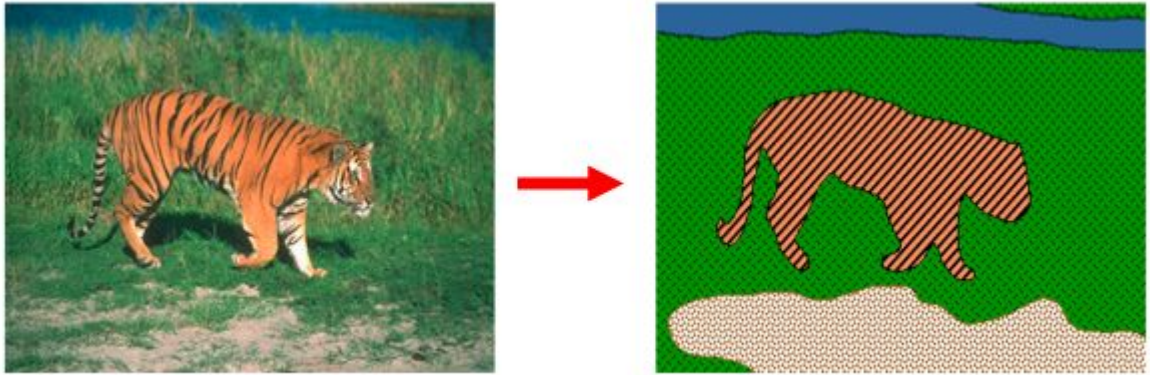


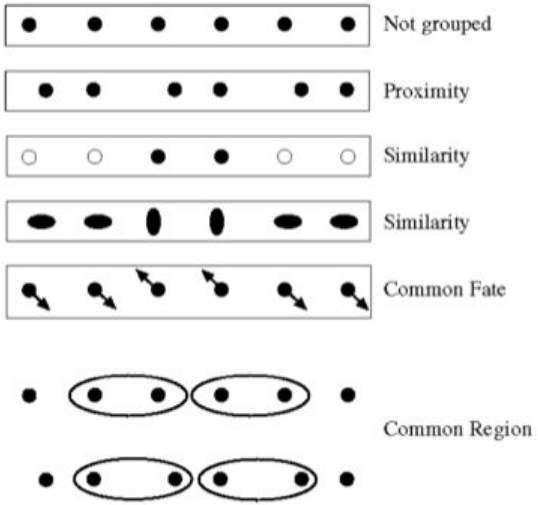
Image segmentation

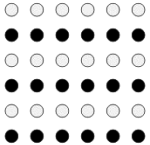
Image segmentation

Задача: разбить похожие пиксели на группы



Гештальт-психология





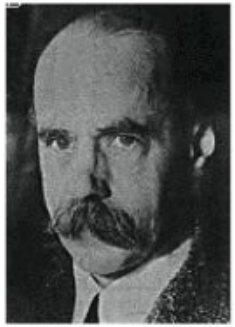
Принцип схожести. Рисунок воспринимается как строки, а не как колонки

Принцип близости. Правая часть рисунка воспринимается как три столбика.

Принцип замкнутости. Рисунок воспринимается не как отдельные отрезки, а как круг и прямоугольник

“I stand at the window and see a house, trees, sky. Theoretically I might say there were 327 brightnesses and nuances of colour. Do I have "327"? No. I have sky, house, and trees.”

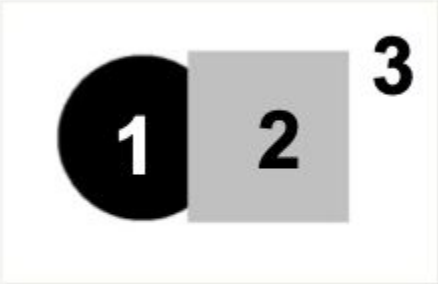
Max Wertheimer
(1880-1943)



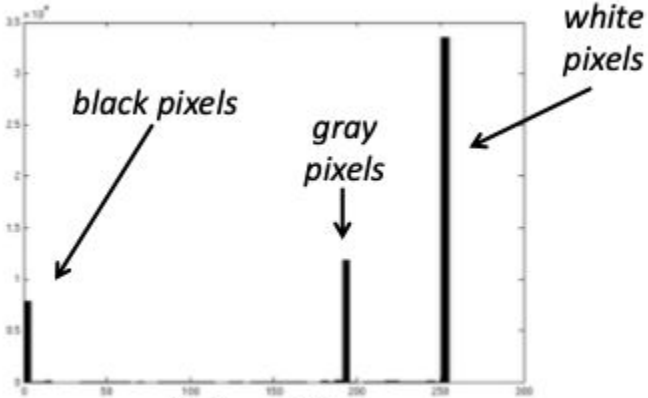
https://postnauka.ru/faq/69883

Image segmentation. Пример

- Интенсивности пикселей образуют 3 группы
- Мы можем отнести каждый пиксель к одной из этих групп в зависимости от интенсивности пикселя



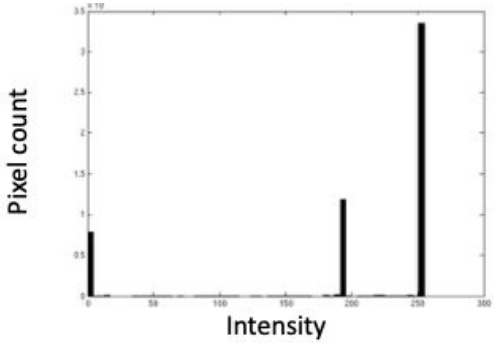
input image

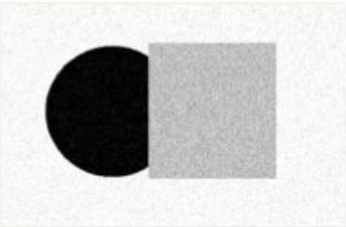


intensity

Image segmentation. Пример

Input image





Input image

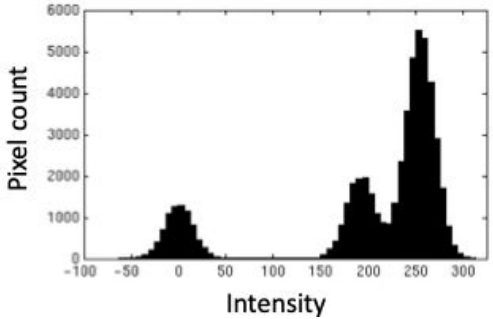
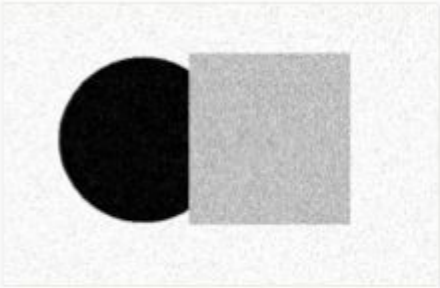


Image segmentation. Пример

- Как теперь определить к какой из групп относится каждый пиксель?
- Нужно кластеризовать!



Input image

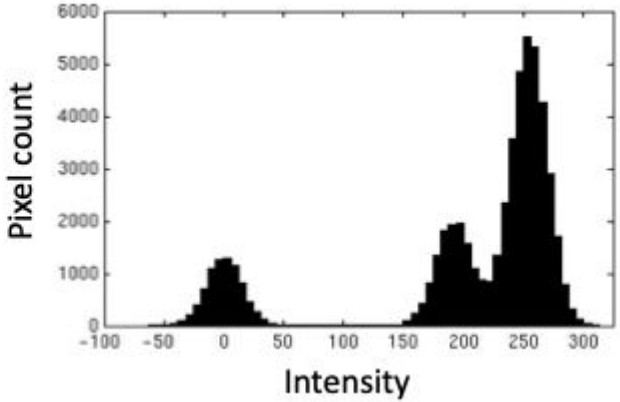
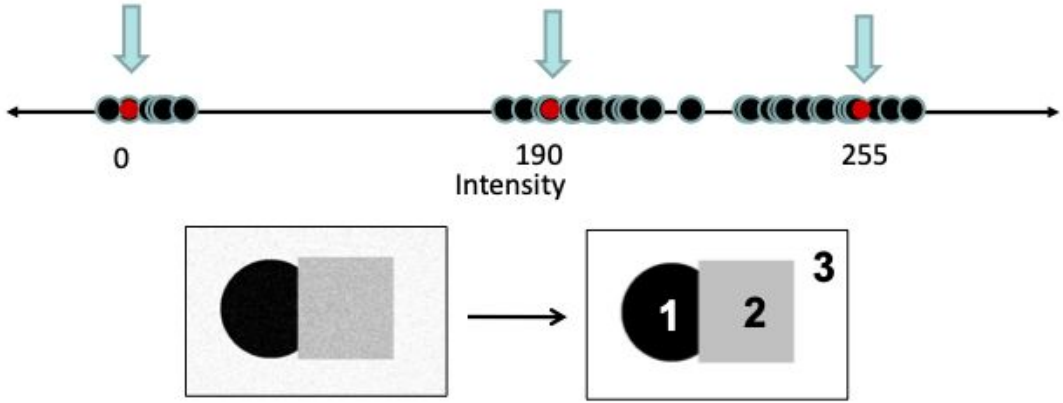


Image segmentation. Пример



Задача: выделить 3 центра кластеров, для каждого из пикселя, проверить к какому из центров он находится ближе всего

Лучшие центры кластеров - это те, которые минимизируют SSD (sum of squared distance)

$$SSD = \sum_{cluster\,i} \sum_{x \in cluster\,i} (x - c_i)^2$$

Минимизируем “разброс”

Задача: минимизировать дисперсию в получаемых кластеров

Cluster center Data

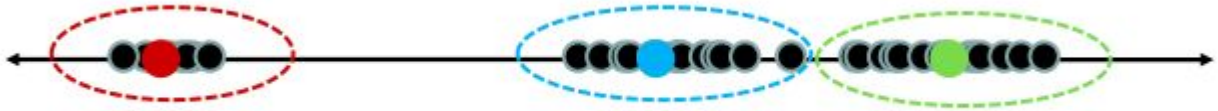
$$c^*,\ \delta^* = \arg\min_{c,\delta} \frac{1}{N} \sum_j^N \sum_i^K \delta_{ij} \left(c_i - x_j\right)^2$$

Whether x_j is assigned to c_i

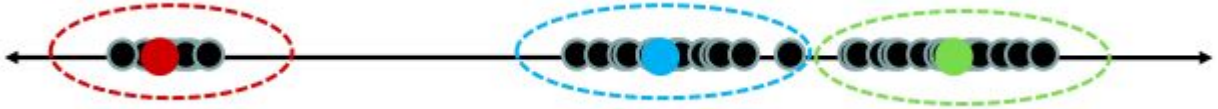
Кластеризация

“что было раньше курица или яйцо?”

- Если бы мы знали центры кластеров, мы могли бы легко разбить пиксели на группы



- Если бы мы знали, как пиксели разбиты на группы, мы могли бы легко посчитать центры кластеров



K-means

1. Инициализация (t=0): центры кластеров $c_1, \ldots, c_K$
2. Группируем пиксели в соответствии с текущими центрами кластеров

$$\delta^t = \underset{\delta}{\operatorname{argmin}} \frac{1}{N} \sum_j^N \sum_i^K \delta_{ij}^{t-1} \left(c_i^{t-1} - x_j \right)^2$$

3. Пересчитываем центры кластеров

$$c^t = \underset{c}{\operatorname{argmin}} \frac{1}{N} \sum_j^N \sum_i^K \delta_{ij}^t \left(c_i^{t-1} - x_j \right)^2$$

4. Возвращаемся к шагу 1

K-means

1. Инициализация (t=0): центры кластеров $c_1, \ldots, c_K$
   a. Обычно: рандомная инициализация
   b. Или жадным алгоритмом

2. Группируем пиксели в соответствии с текущими центрами кластеров
   a. Метрика: евклидова, косинусная

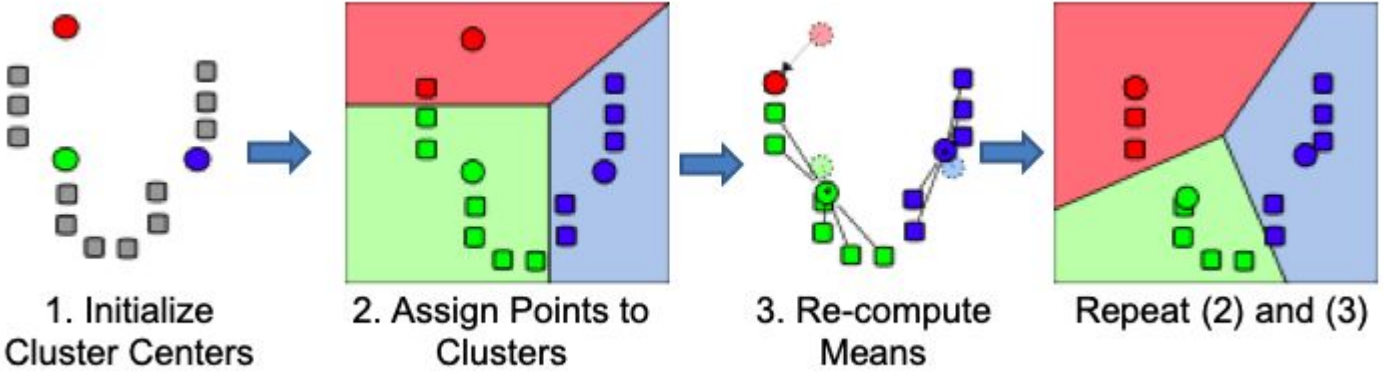
$$\delta^t = \underset{\delta}{\operatorname{argmin}} \frac{1}{N} \sum_j^N \sum_i^K \delta_{ij}^{t-1} \left(c_i^{t-1} - x_j \right)^2$$

3. Пересчитываем центры кластеров

$$c^t = \underset{c}{\operatorname{argmin}} \frac{1}{N} \sum_j^N \sum_i^K \delta_{ij}^t \left(c_i^{t-1} - x_j \right)^2$$

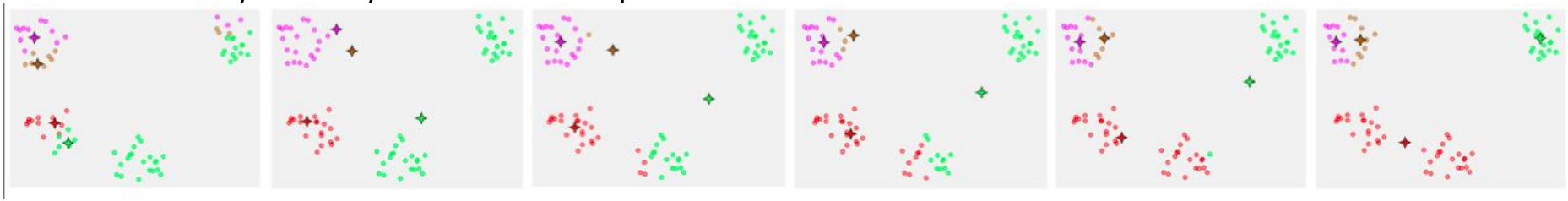
4. Возвращаемся к шагу 1
   a. Пока центры не перестают меняться

K-means

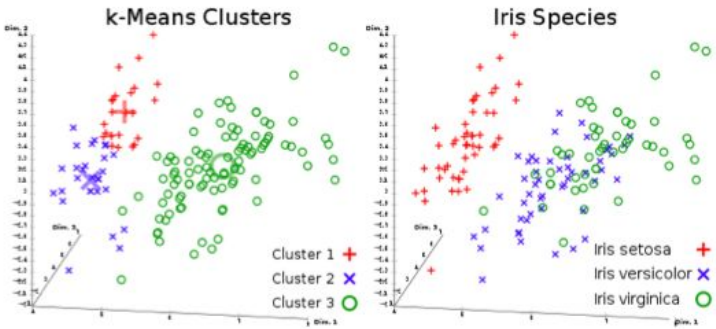

1. Initialize Cluster Centers
2. Assign Points to Clusters
3. Re-compute Means
Repeat (2) and (3)

K-means

- Сходится к локальному минимуму
  - Нужно запускать несколько раз



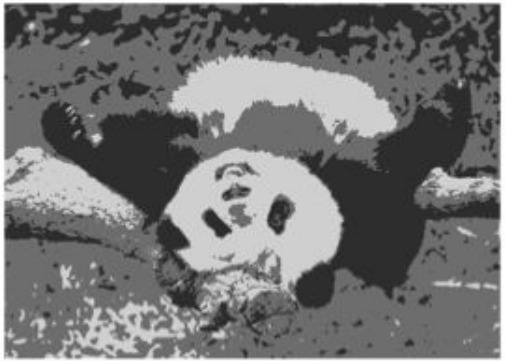
- Релевантен только для “шарообразных данных”
- Требует выбора K (количества кластеров)
- Вычислительно затратен при большом количестве данных



K-means

Original image

2 clusters



3 clusters

K-means++

- Как можно улучшить инициализацию?

1. Рандомно выбираем первый центр
2. Выбираем новый центр с вероятностью пропорциональной $(x - c_i)^2$
   a. Чем дальше x, тем большую ошибку он вносит, и тем с большей вероятностью его выберут как новый центр кластера
3. Повторить пока не наберем К центров

Пространство признаков

В зависимости от того, что мы выберем как признаки, можно по-разному кластеризовать пиксели

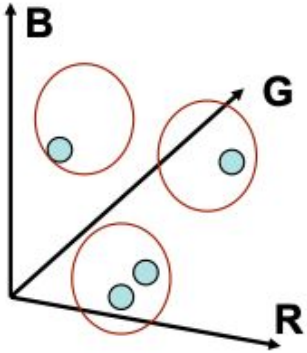
Простейший случай: ч/б картинка группировка по интенсивности

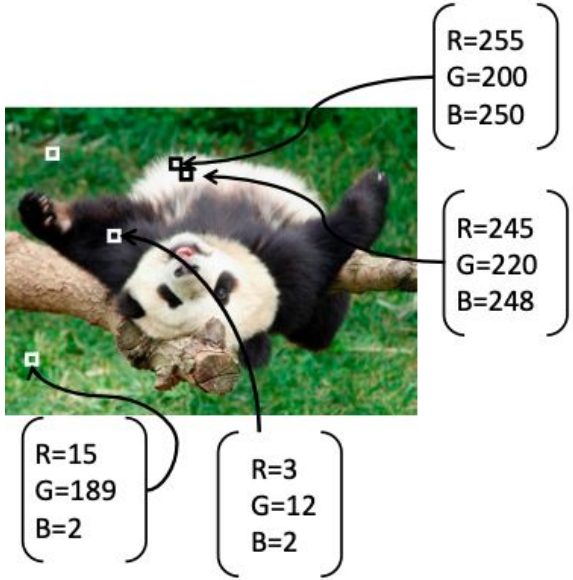
Признаки: интенсивность (1D)

Пространство признаков

В зависимости от того, что мы выберем как признаки, можно по-разному кластеризовать пиксели

Группировка пикселя по схожести цвета (RGB)



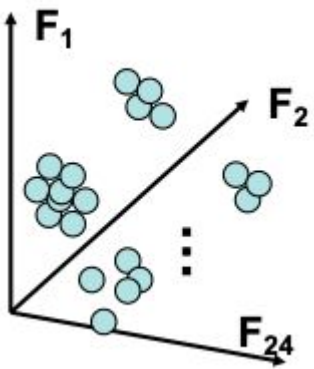


Признаки: значение цвета (3D)

Пространство признаков

В зависимости от того, что мы выберем как признаки, можно по-разному кластеризовать пиксели

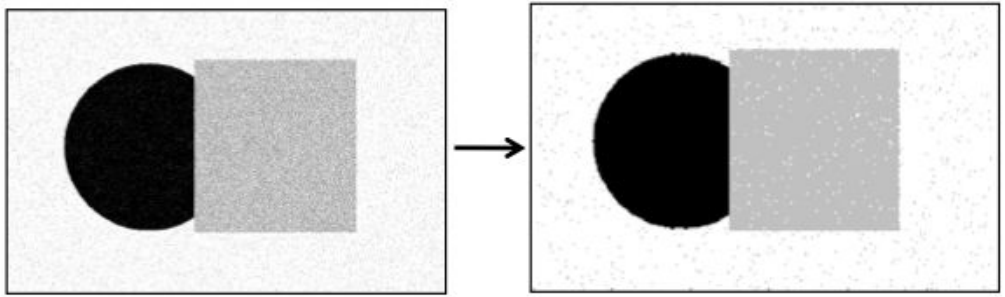
Группировка пикселя по схожести текстур



Признаки: отклики на различные фильтры (24D)

Пространство признаков

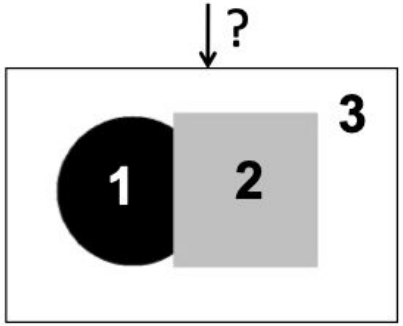
Попиксельная кластеризация почти наверняка приведет к большому количеству выбросов



Original

Labeled by cluster center’s intensity

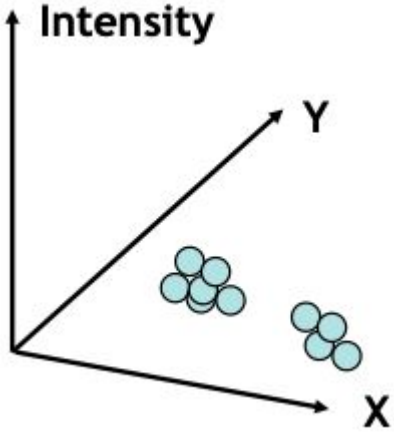
Как учитывать пространственную информацию при кластеризации?

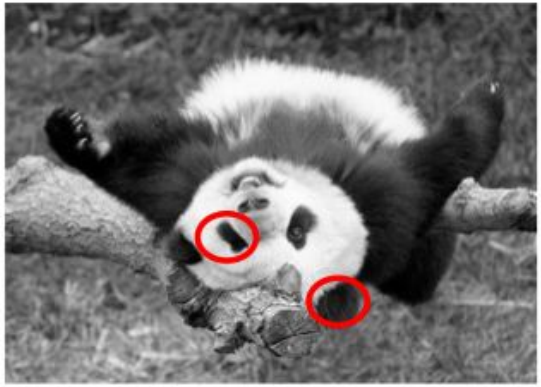


Пространство признаков

В зависимости от того, что мы выберем как признаки, можно по-разному кластеризовать пиксели

Группировка пикселя по интенсивности+позиции

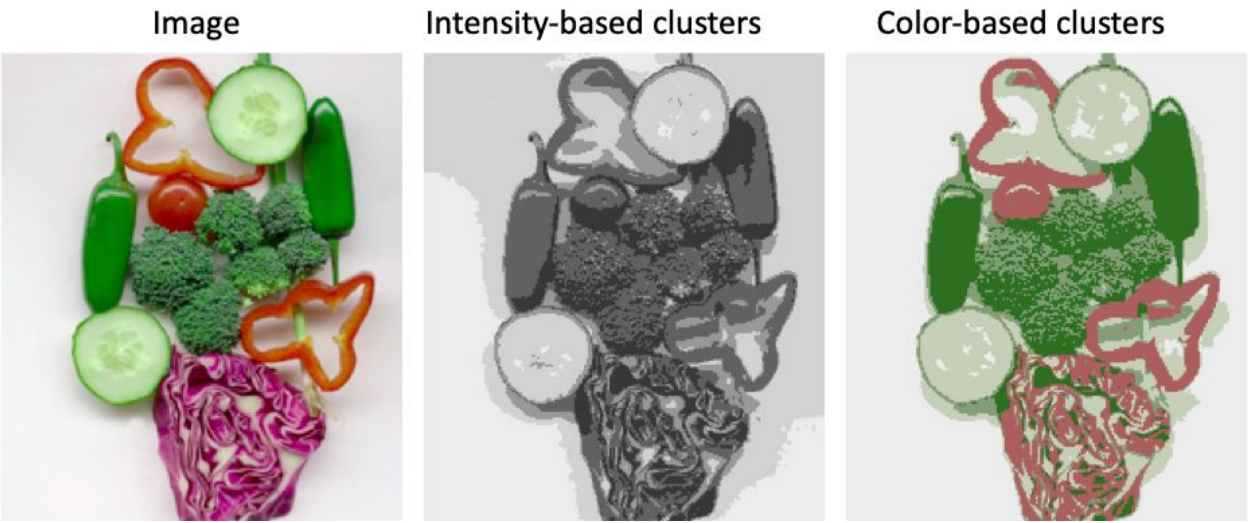




K-means

K-means на основе цвета или интенсивности - это своего рода квантование (сжатие) изображения

Такие кластеры не будут пространственно согласованными



K-means

K-means на основе цвета или интенсивности - это своего рода квантование (сжатие) изображения

Такие кластеры не будут пространственно согласованными

Кластеры значений (r, g, b, x, y) будут пространственно согласованными

Метрики качества

Как оценивать кластеры?

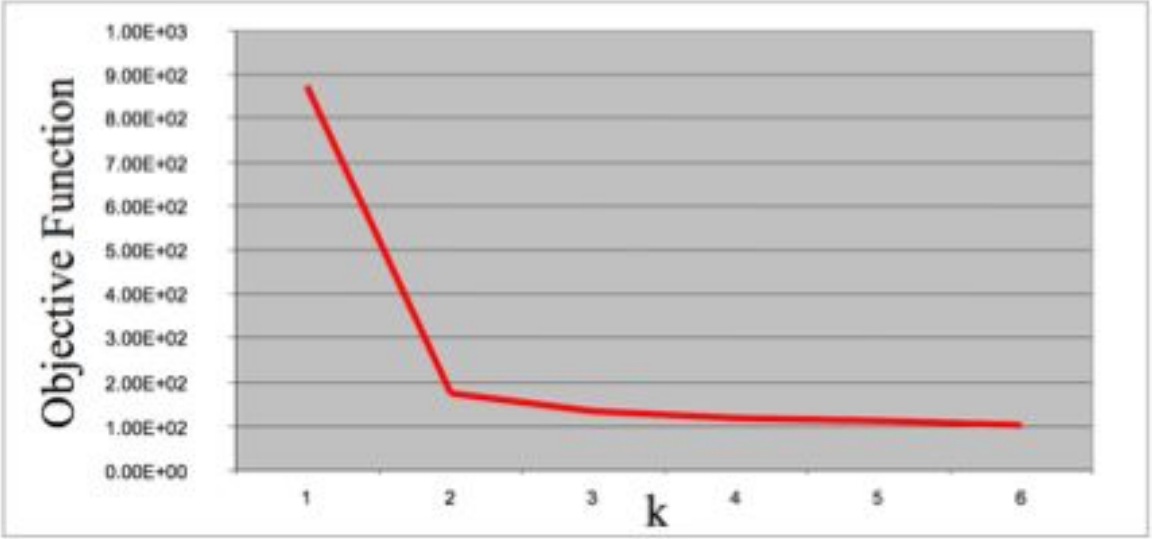
- Генеративные метрики

  Насколько хорошо кластеры соотносятся с точками?
- Дискриминативные метрики

  Насколько хорошо мы предсказали класс для каждого пикселя
- Обучения без учителя не подразумевает дискриминативных метрик, поскольку у нас нет истинных значений классов пикселей

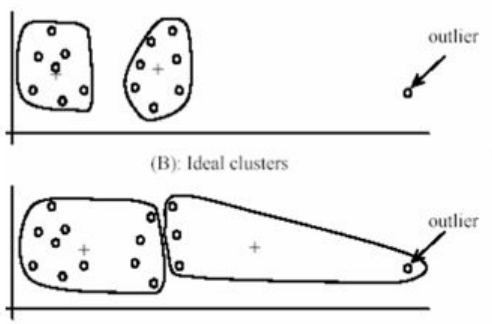
Как выбрать количество кластеров?

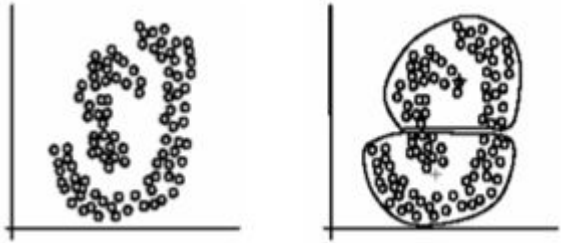
Попробовать различное количество кластеров, и посмотреть значение метрики качества для каждого из них



K-means: pros and cons

- Pros:
  - Находит центры кластеров, минимизирующих эмпирическую дисперсию (хорошо описывает закономерности в данных)
  - Быстро и просто, легко заимплементить
- Cons:
  - Необходимо выбирать количество K
  - Чувствителен к выбросами
  - Сходится к локальному минимуму
  - Не универсальная метрика расстояния
  - Может быть медленным: каждая итерация стоит O(KNd) для N точек размерности d
- Usage:
  - Кластеризация без учителя
  - Сегментация пикселей

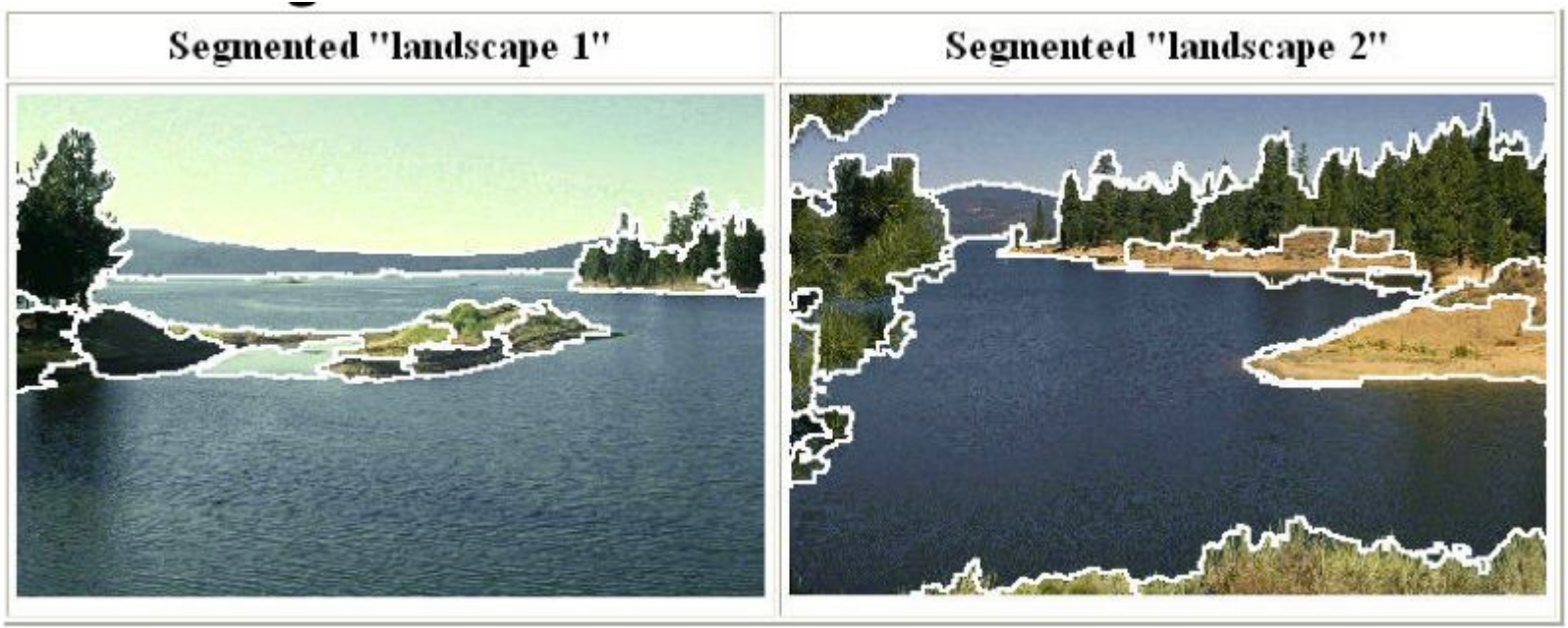




Mean-shift clustering

Mean-shift clustering

Гибкий алгоритм для cluster-based сегментации

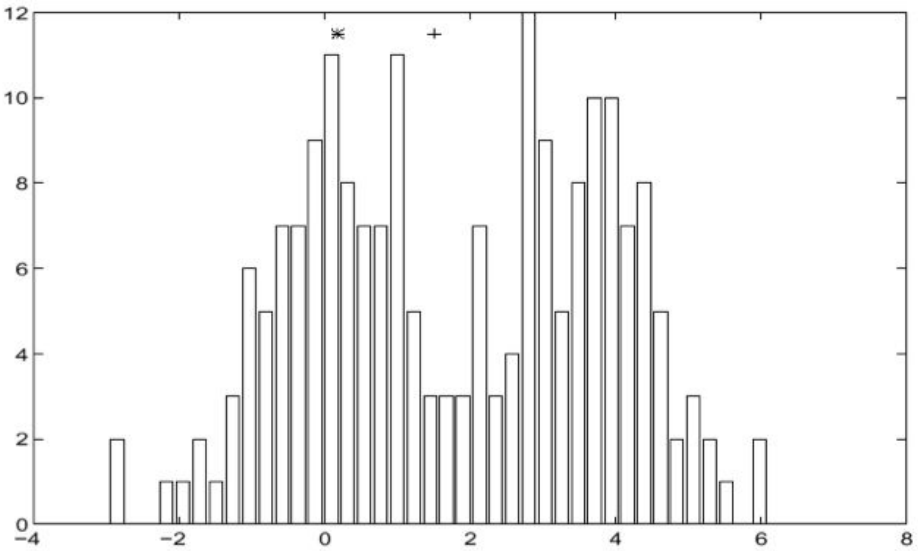


D. Comaniciu and P. Meer, Mean Shift: A Robust Approach toward Feature Space Analysis, PAMI 2002.

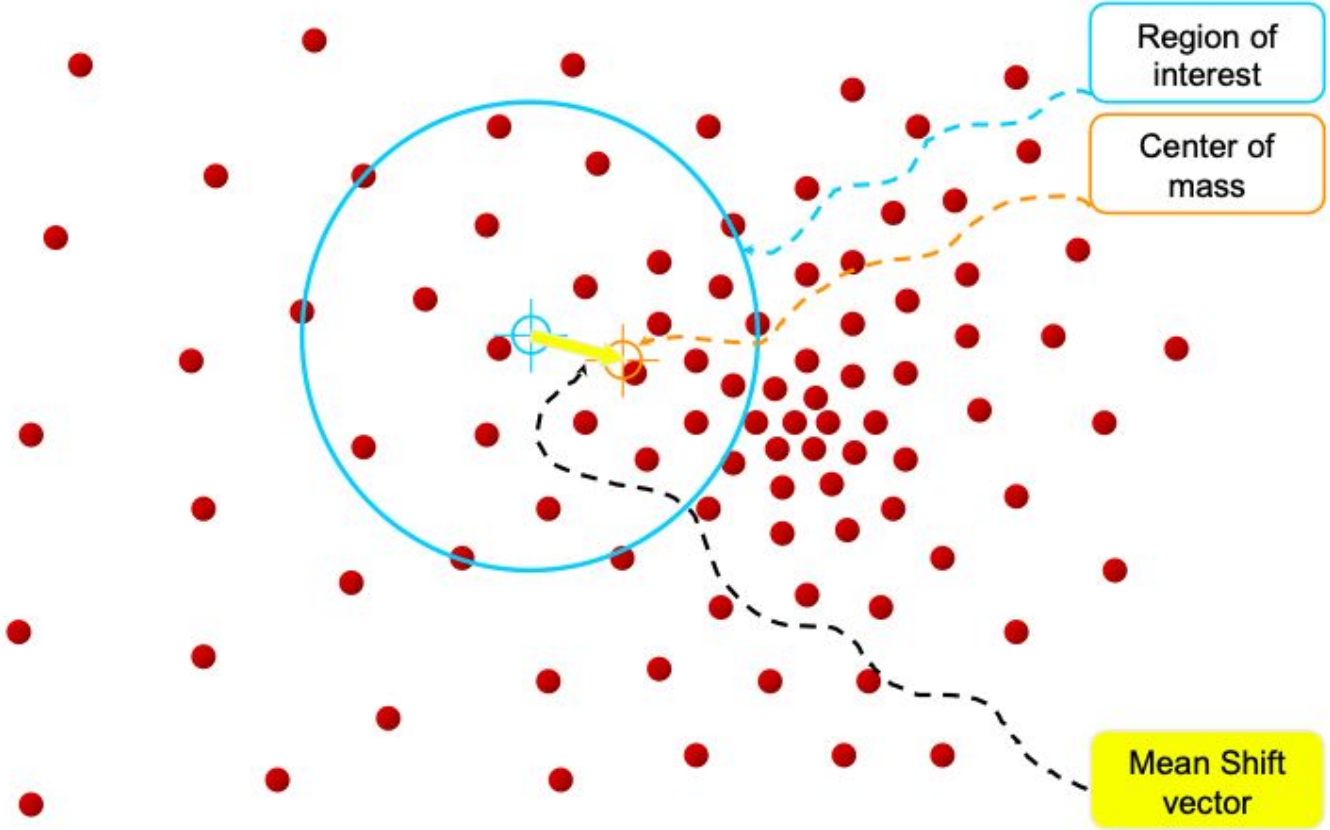
Mean-shift clustering

Итеративный алгоритм:

1. Инициализируем несколько точек и окно W
2. Подсчитываем центроиду окна (mean of window) $\sum_{x \in W} x H(x)$
3. Сместить окно к центроиде
4. Вернуться к шагу 2, пока не сойдется

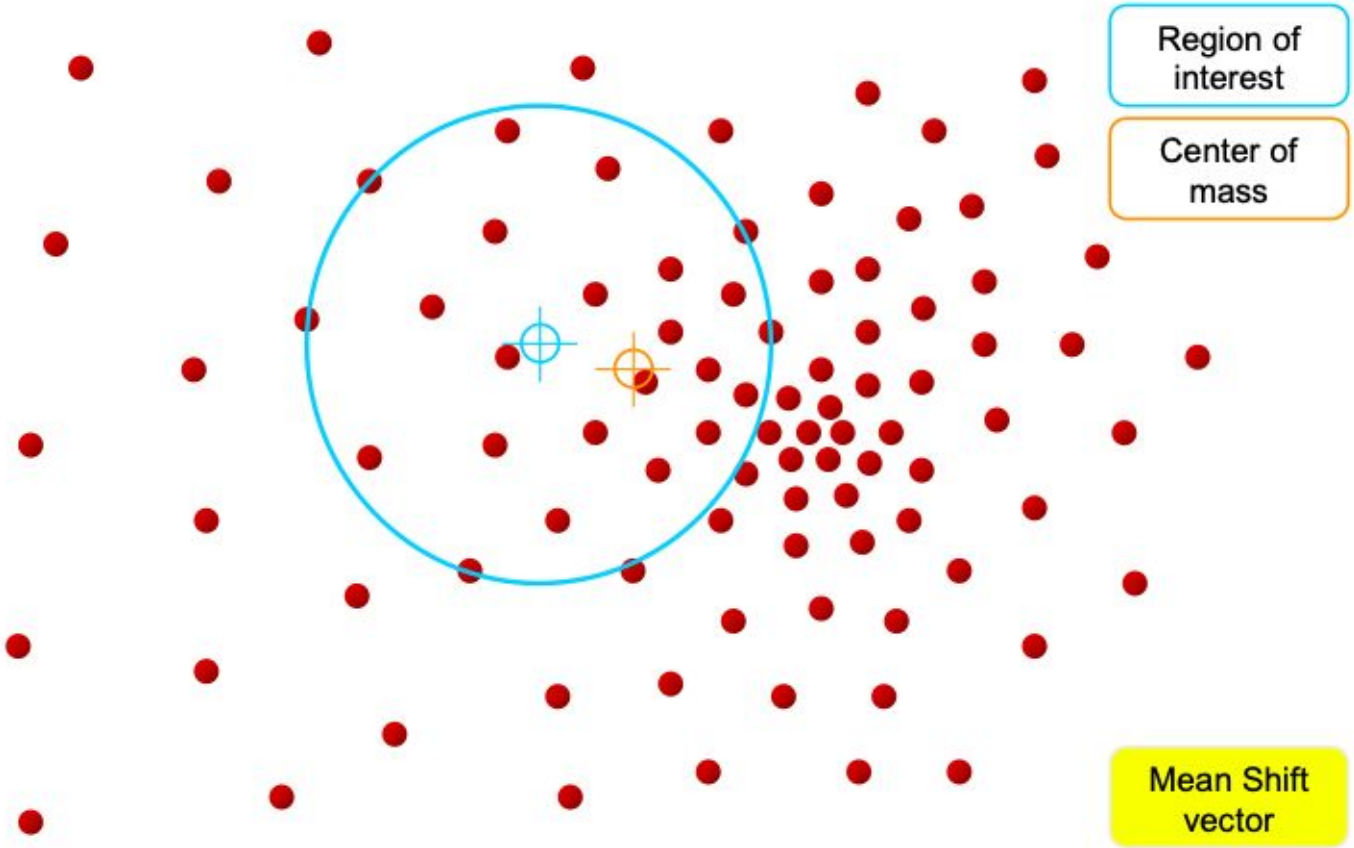


Mean-shift clustering



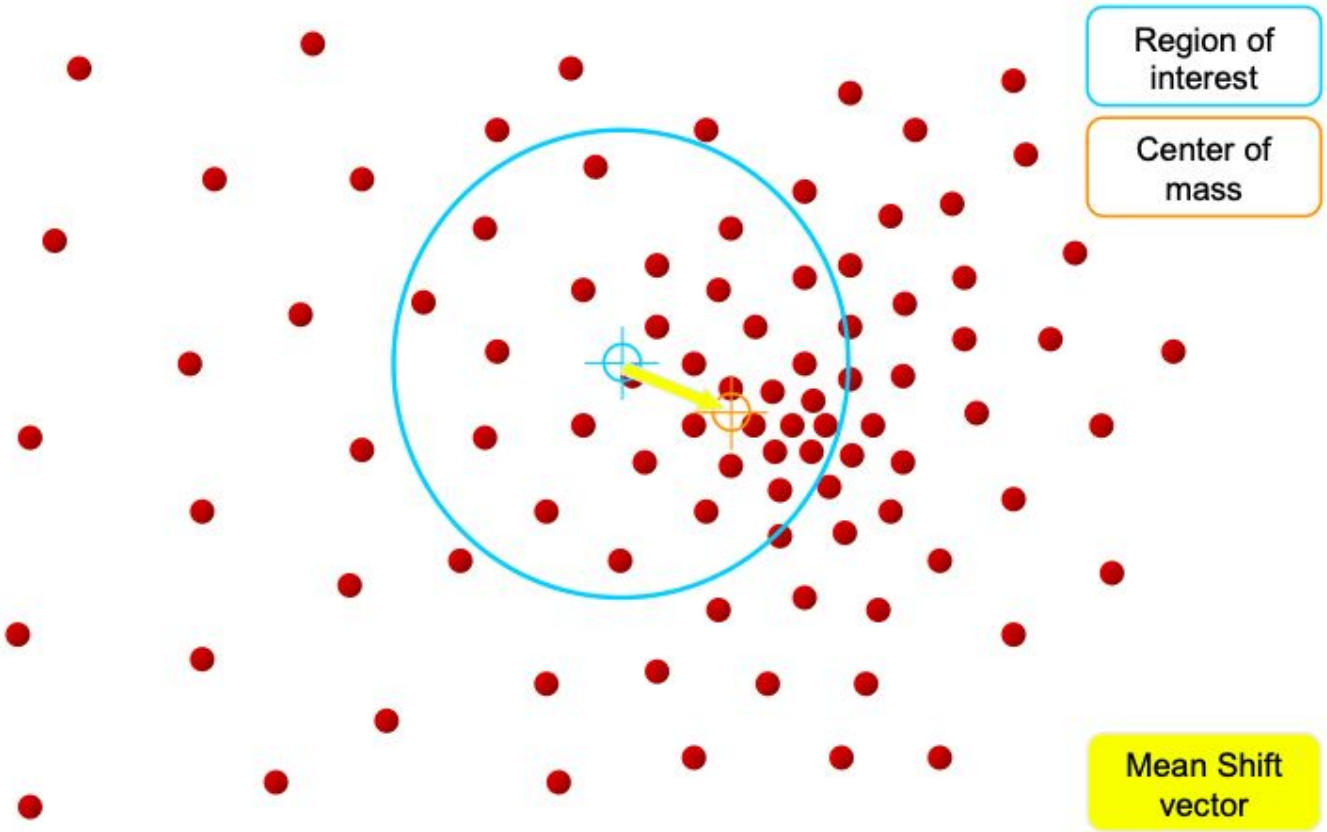
Slide by Y. Ukrainitz & B. Sarel

Mean-shift clustering



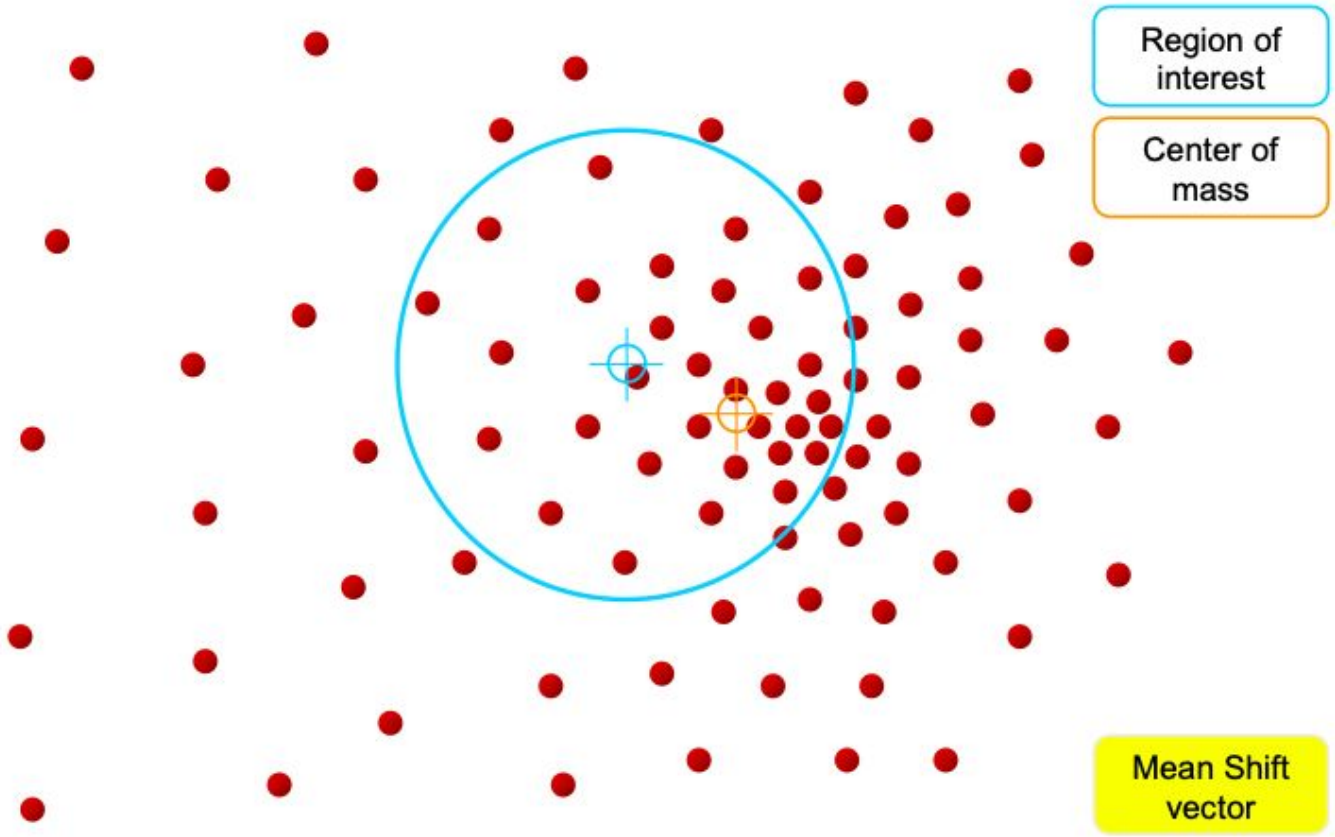
Slide by Y. Ukrainitz & B. Sarel

Mean-shift clustering



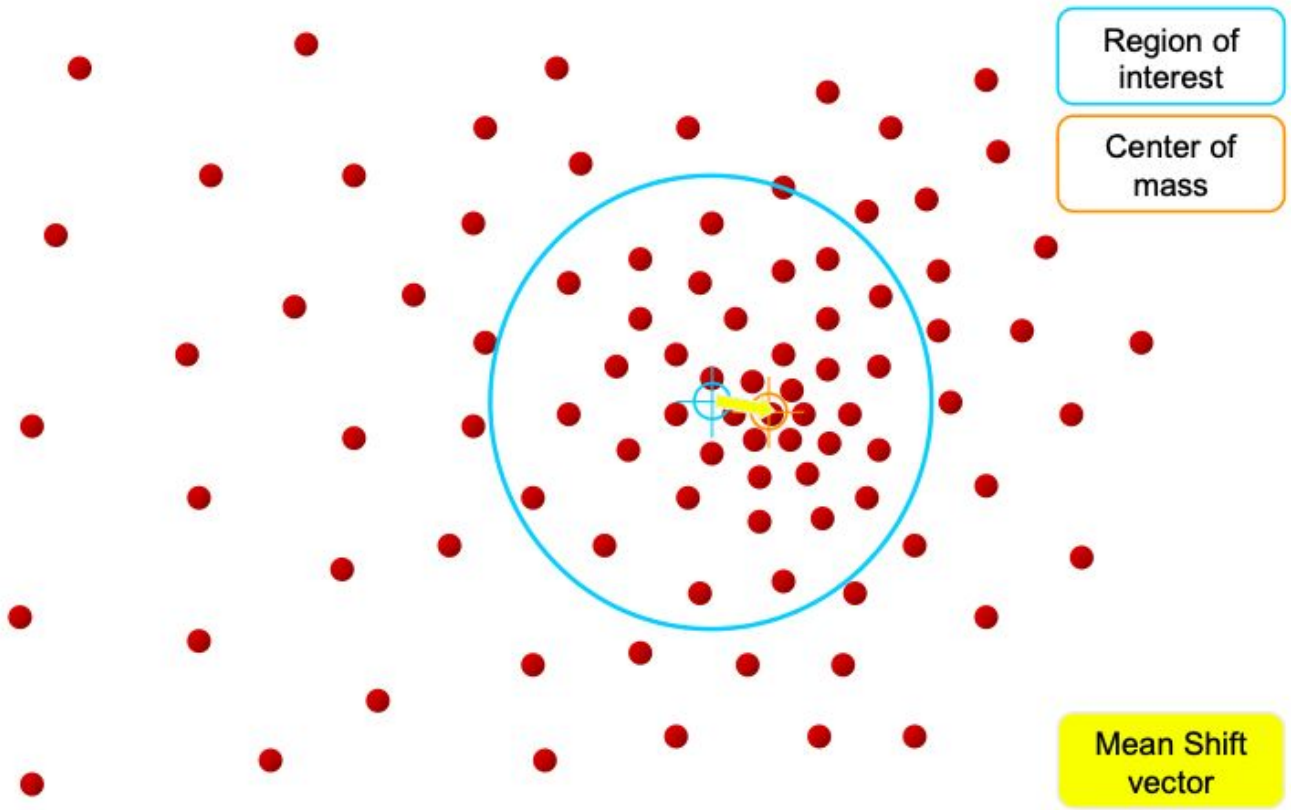
Slide by Y. Ukrainitz & B. Sarel

Mean-shift clustering



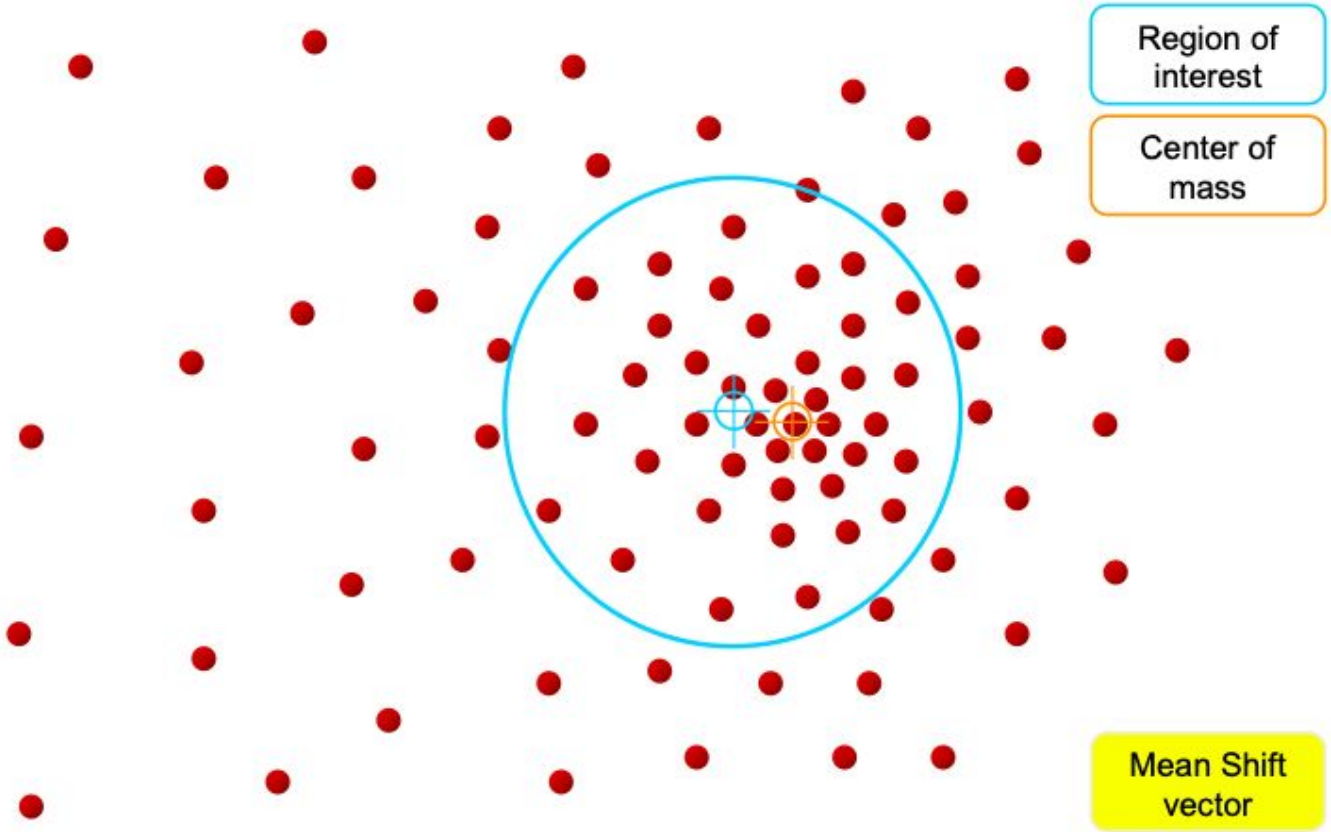
Slide by Y. Ukrainitz & B. Sarel

Mean-shift clustering



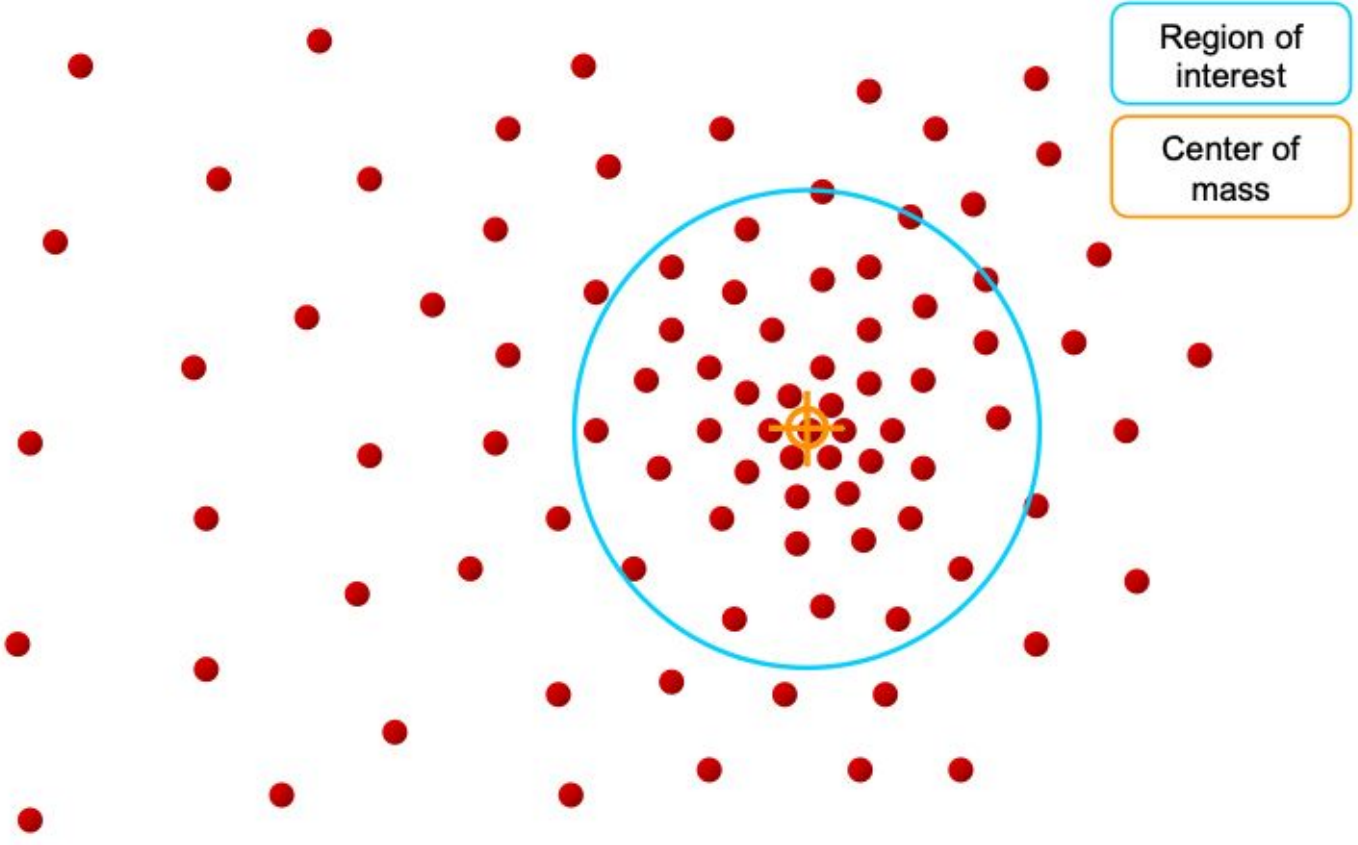
Slide by Y. Ukrainitz & B. Sarel

Mean-shift clustering



Slide by Y. Ukrainitz & B. Sarel

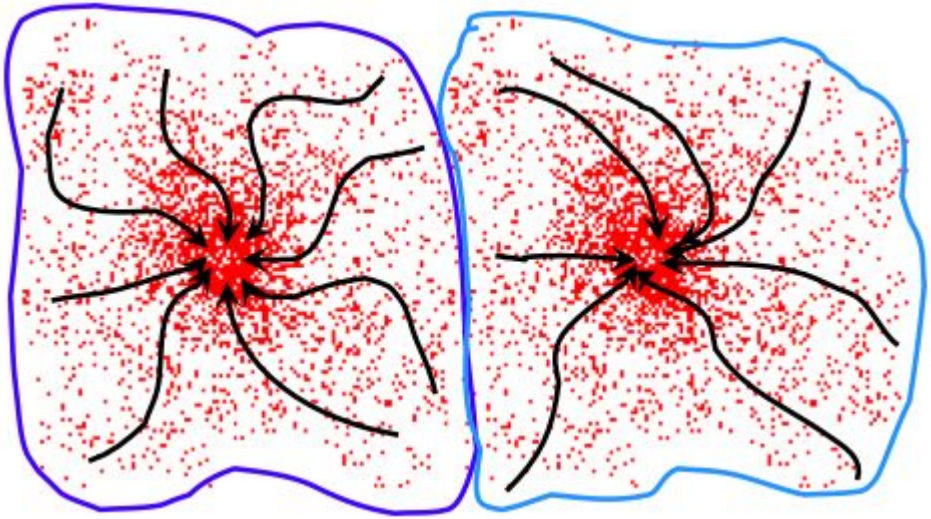
Mean-shift clustering



Slide by Y. Ukrainitz & B. Sarel

Mean-shift clustering

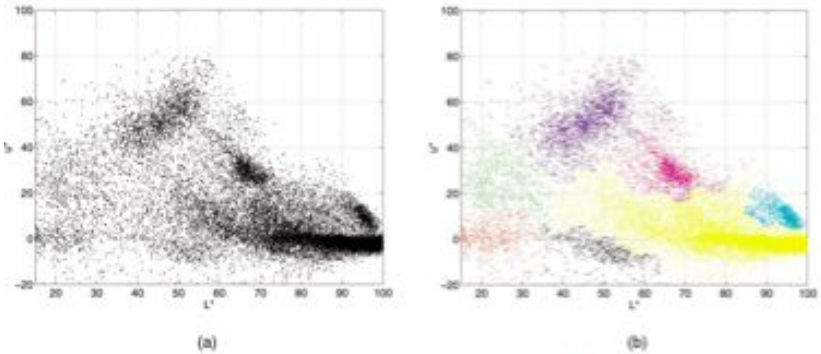
- Кластер: место куда “стекаются” окна
- Множество притяжения: область, в которой все траектории окон ведут в одну точку

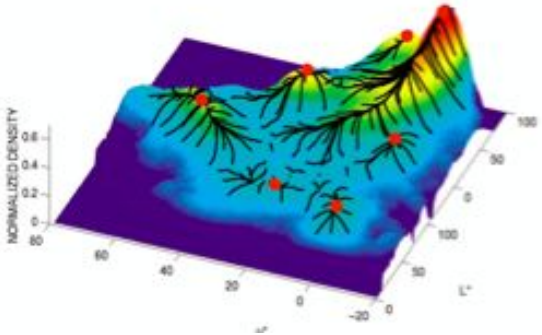


Mean-shift clustering

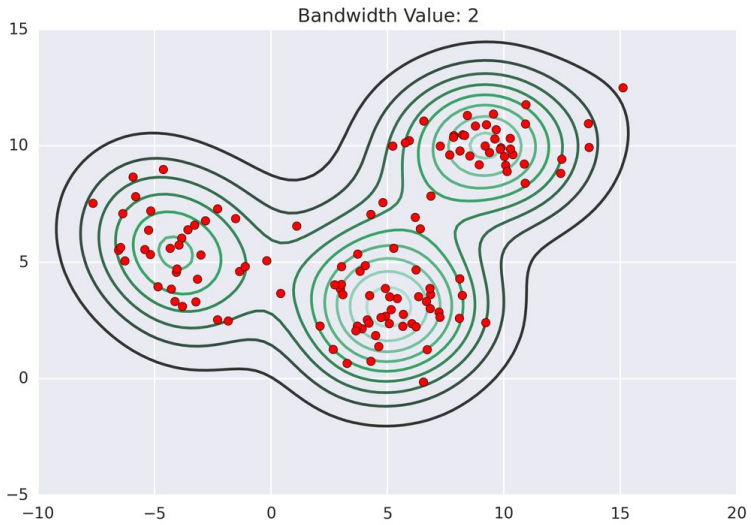
Алгоритм:

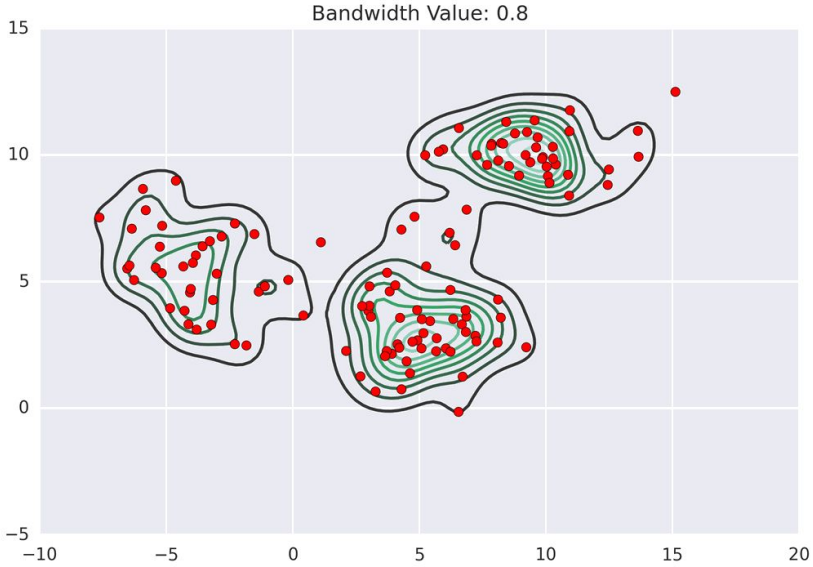
1. Определить признаки для каждого из пикселей (цвет, градиент, местоположение)
2. Инициализировать окна на месте каждого пикселя
3. Пересчитать центры окон, пока алгоритм не сойдется
4. Смерджить окна, которые сошлись в одну точку



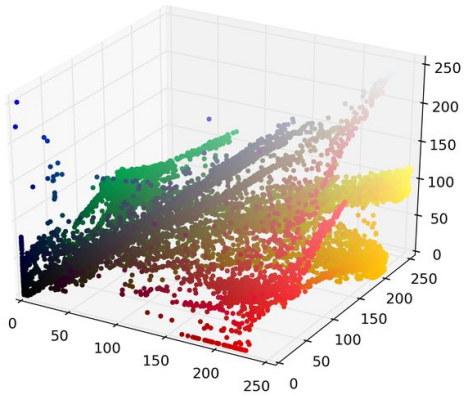


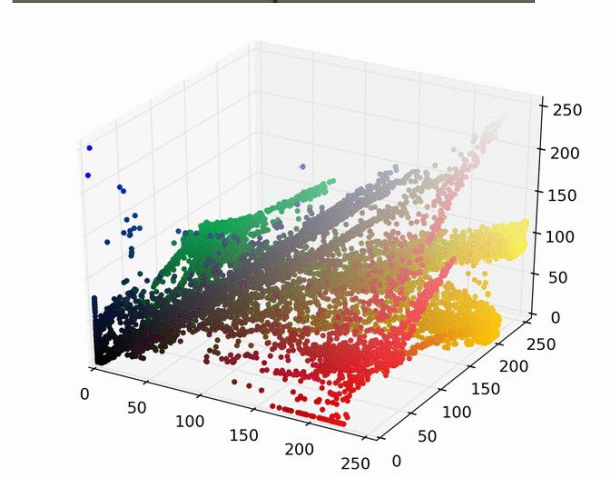
Mean-shift algorithm





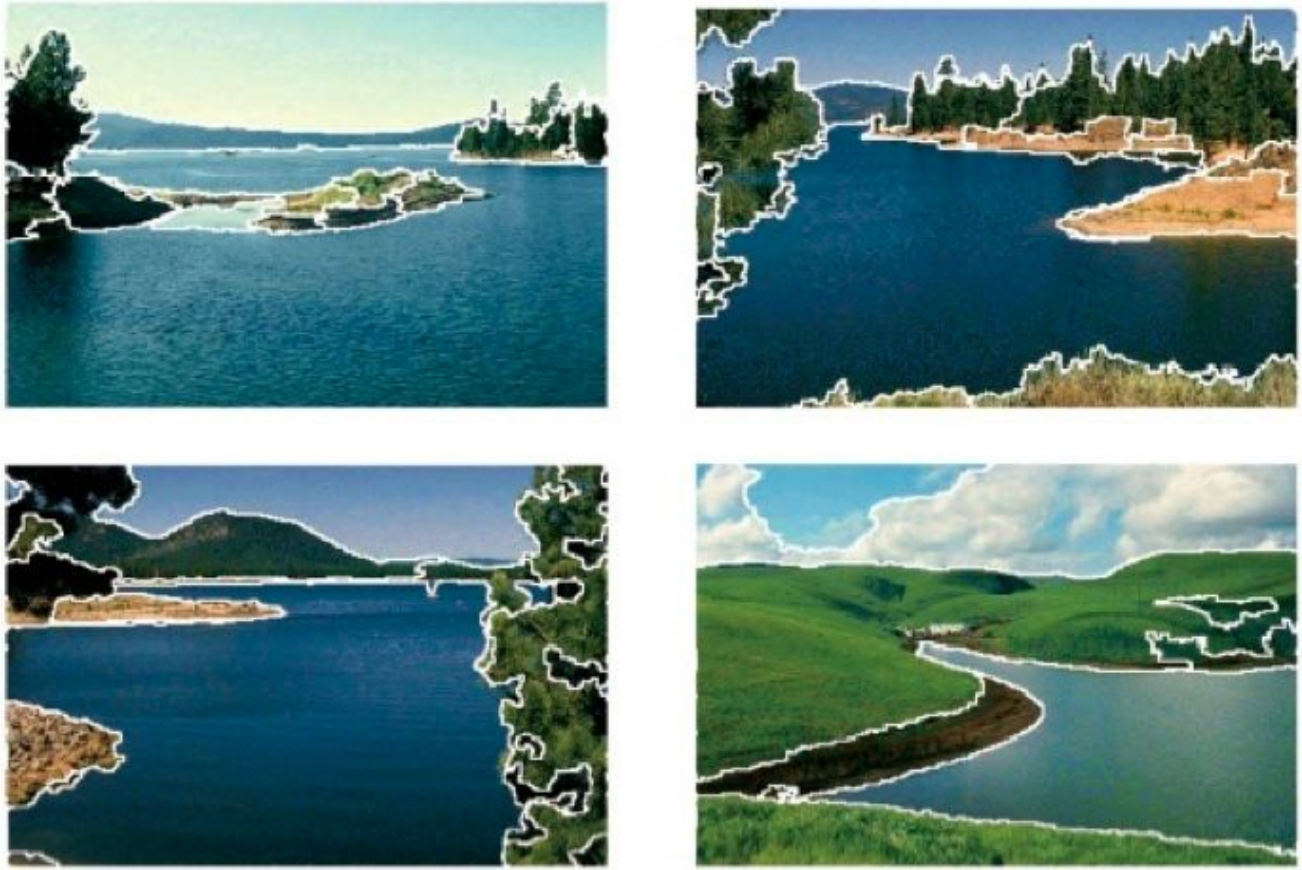
Mean-shift algorithm





Примеры работы

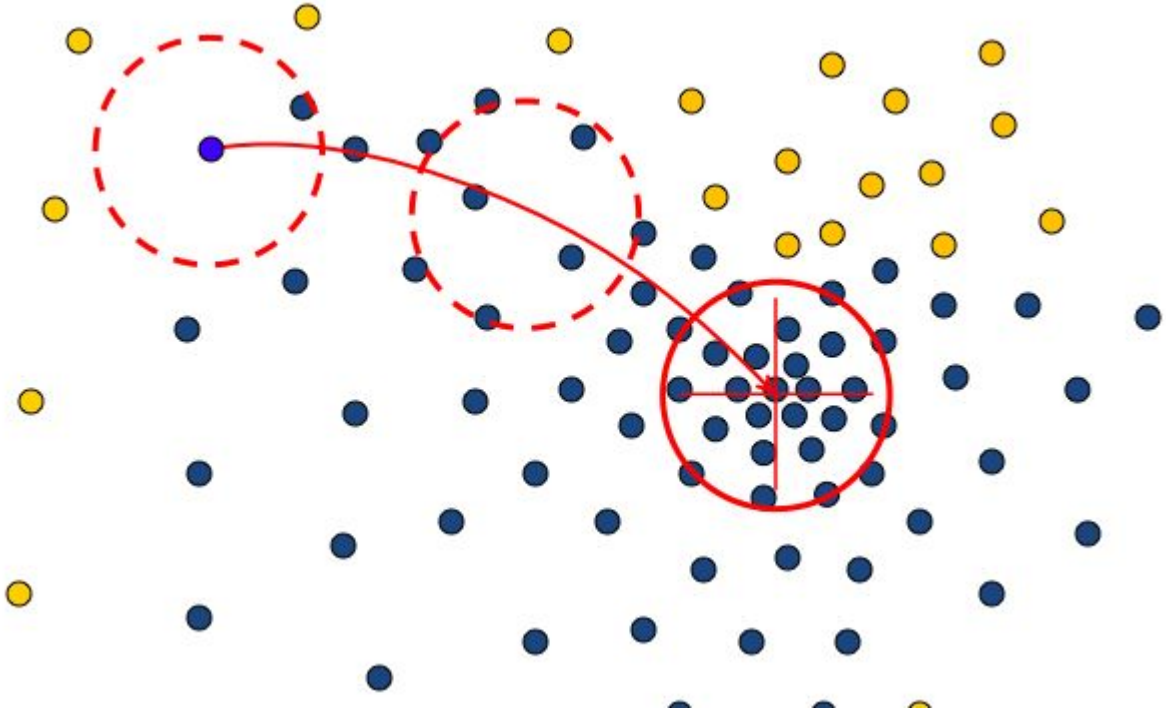
Примеры работы



Примеры работы

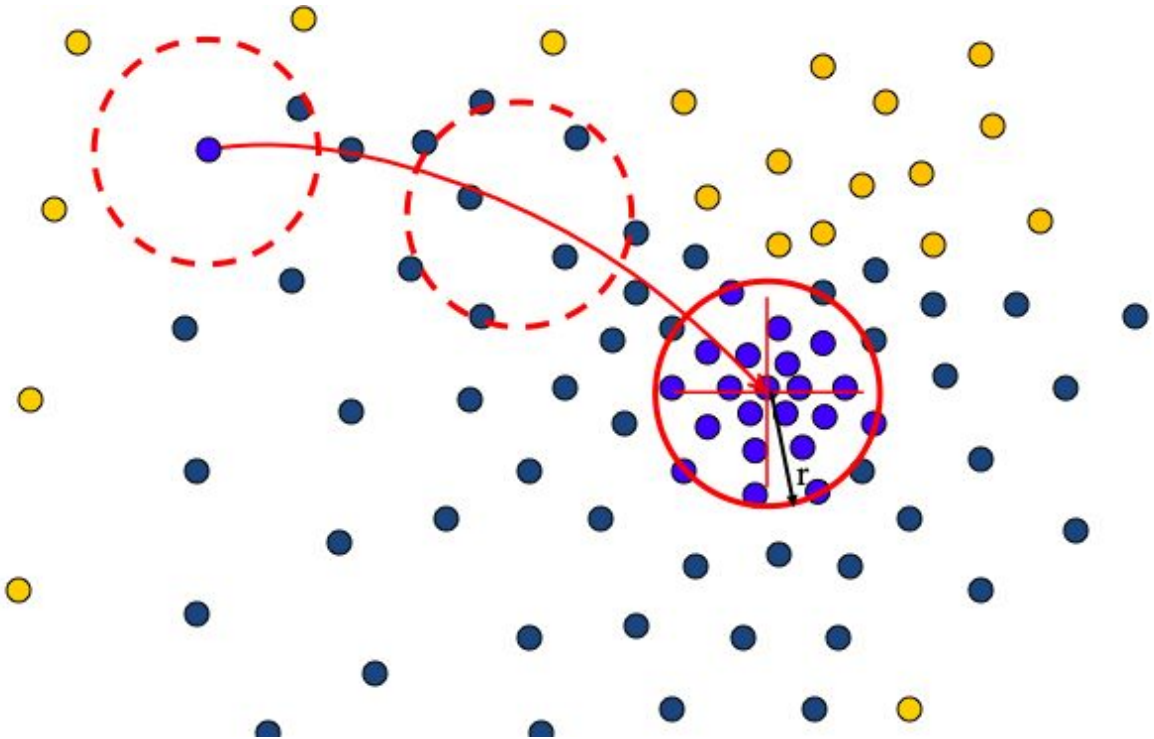
Проблема: вычислительно затратно

- Нужно рассматривать каждый пиксель...
- Многие вычисления будут избыточными



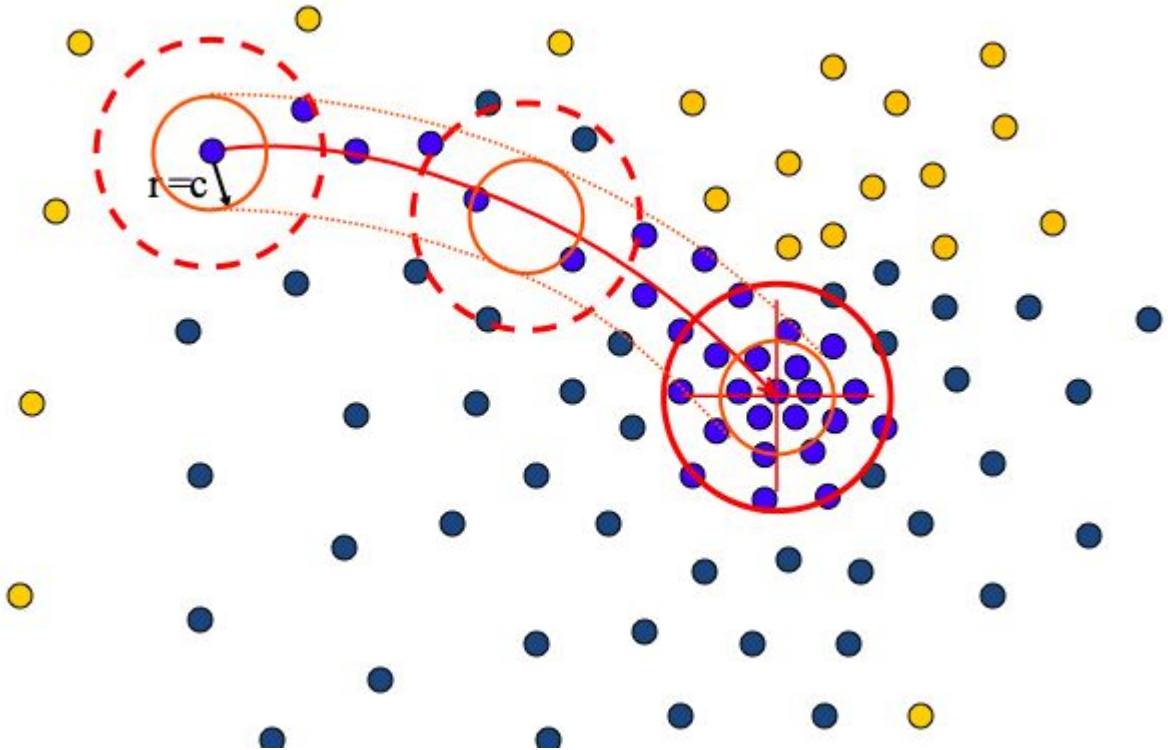
Проблема: вычислительно затратно

1. Когда окно пришло в конечную точку, назначить все точки в радиусе r к этому же кластеру



Проблема: вычислительно затратно

2. Все точки на расстоянии c<r от центра окон в каждый момент прохождения траектории присвоить к тому же кластеру-> уменьшает количество рассматриваемых точек



Резюме: mean-shift алгоритм

- Pros
  - Гибкий алгоритм, подходит для многих задач
  - Подходит для данных любой “формы” (сферические, эллиптические, ...)
  - Всего один параметр (размер окна h)
    - Понятная статистическая интерпретация
  - Устойчиво к выбросам
- Cons
  - Результат сильно зависит от выбора параметра
  - h трудно выбрать
  - Вычислительно затратно
  - Плохо масштабируется с увеличением количества признаков

Image compression

Matrix factorization

- Есть множество алгоритмов, факторизующих матрицы - раскладывают на сомножители
- Самый популярный/крутой/полезный - SVD
- Представляет любую матрицу A в виде произведения трех матриц: A = $U\Sigma V^T$
- Python: np.linalg.svd(A)

SVD

- U, V - ортогональные матрицы, Σ диагональная матрица

$$U\Sigma V^T = A$$

$$\underset{U}{\begin{bmatrix} -.40 & .916 \\ .916 & .40 \end{bmatrix}} \times \underset{\Sigma}{\begin{bmatrix} 5.39 & 0 \\ 0 & 3.154 \end{bmatrix}} \times \underset{V^T}{\begin{bmatrix} -.05 & .999 \\ .999 & .05 \end{bmatrix}} = \underset{A}{\begin{bmatrix} 3 & -2 \\ 1 & 5 \end{bmatrix}}$$

SVD

В общем случае, если A - MxN матрица, тогда

U- MxM матрица, Σ - MxN матрица, V - NxN матрица

$$\overset{U}{\begin{bmatrix} -.40 & .916 \\ .916 & .40 \end{bmatrix}} \times \overset{\Sigma}{\begin{bmatrix} 5.39 & 0 \\ 0 & 3.154 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.05 & .999 \\ .999 & .05 \end{bmatrix}} = \overset{A}{\begin{bmatrix} 3 & -2 \\ 1 & 5 \end{bmatrix}}$$

SVD

- SVD можно рассматривать как разбиение на 3 преобразования - поворот, масштабирование, и снова поворот
- Также SVD может быть очень полезно для матриц изображений
- Будем смотреть на SVD в менее геометрическом ключе

SVD

$$\overset{U}{\begin{bmatrix} -.39 & -.92 \\ -.92 & .39 \end{bmatrix}} \times \overset{\Sigma}{\begin{bmatrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} = \overset{A}{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}}$$

$$\overset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \quad \overset{A_{partial}}{\begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}}$$

$$+ \overset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \quad \overset{A_{partial}}{\begin{bmatrix} -.6 & -.1 & .4 \\ .2 & 0 & -.2 \end{bmatrix}}$$

$$= \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

SVD

$$
\begin{array}{cccc}
 & U\Sigma & V^T & A_{partial} \\
 & \begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix} \times & \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} & \begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix} \\
+ & \begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix} \times & \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} & \begin{bmatrix} -.6 & -.1 & .4 \\ .2 & 0 & -.2 \end{bmatrix} \\
= & & & \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}
\end{array}
$$

Каждая компонента (i-ый столбец U)*(i-ое значение Sigma)*(i-ая строка V^T) формирует финальную матрицу A

SVD

$$
\begin{array}{c}
U\Sigma \\
\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}
\end{array}
\times
\begin{array}{c}
V^T \\
\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}
\end{array}
\quad
\begin{array}{c}
A_{partial} \\
\begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}
\end{array}
$$

$$
+
\begin{array}{c}
U\Sigma \\
\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}
\end{array}
\times
\begin{array}{c}
V^T \\
\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}
\end{array}
\quad
\begin{array}{c}
A_{partial} \\
\begin{bmatrix} -.6 & -.1 & .4 \\ .2 & 0 & -.2 \end{bmatrix}
\end{array}
\qquad
\begin{array}{c}
A \\
\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}
\end{array}
$$

$$
= \quad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}
$$

- Мы восстанавливаем A как линейную комбинацию столбцов U
- Используя все столбцы U, мы полностью восстановим исходную матрицу A
- Но мы можем забить и использовать только первые несколько столбцов, и все равно получим довольно близкий результат

SVD

$$
\begin{array}{c}
U\Sigma \\
\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}
\end{array}
\times
\begin{array}{c}
V^T \\
\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}
\end{array}
\quad
\begin{array}{c}
A_{partial} \\
\begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}
\end{array}
$$

$$
+
\begin{array}{c}
U\Sigma \\
\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}
\end{array}
\times
\begin{array}{c}
V^T \\
\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}
\end{array}
\quad
\begin{array}{c}
A_{partial} \\
\begin{bmatrix} -.6 & -.1 & .4 \\ .2 & 0 & -.2 \end{bmatrix}
\end{array}
\qquad
\begin{array}{c}
A \\
\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}
\end{array}
$$

$$
= \quad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}
$$

- Будем называть эти первые столбцы матрицы U **главными компонентами**
- Они содержат основные “паттерны” исходной матрицы A
- Строки матрицы V^T показывают, какие коэффициенты нужно ставить перед главными компонентами, чтобы восстановить исходную матрицу

SVD для сжатия изображений

- Используя только 10 из 300 главных компонент, можно добиться достаточно хорошего качества
- SVD можно использовать для сжатия изображений

SVD

$$\underset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \underset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \quad \underset{A_{partial}}{\begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}}$$

- Столбцы U - главные компоненты данных, содержат основные паттерны, их линейная комбинация позволяет восстановить стобцы исходной матрица
- Построим матрицу, где каждый столбец - отдельная “точка”
- Факторизовать с помощью SVD, посмотреть на первые столбцы U, чтобы увидеть паттерны, общие среди столбцов
- Это называется метод главных компонент (PCA)

SVD

$$\underset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \underset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \quad \underset{A_{partial}}{\begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}}$$

- Зачастую, сырые данные избыточны, и содержат много общих паттернов
- PCA позволяет представить данные как веса векторов главных компонент
- С помощью такого представления, можно получить явную разницу между различными изображений
- Позволяет делать многие алгоритмы машинного обучения гораздо более эффективными

Image compression

- Разбить оригинальное изображение 372x492 на патчи
  - Каждый патч - размера 12x12, патчи не пересекаются
- Будем рассматривать патч, как вектор размерности 144D
- Всего - около 1300 патчей

Image compression

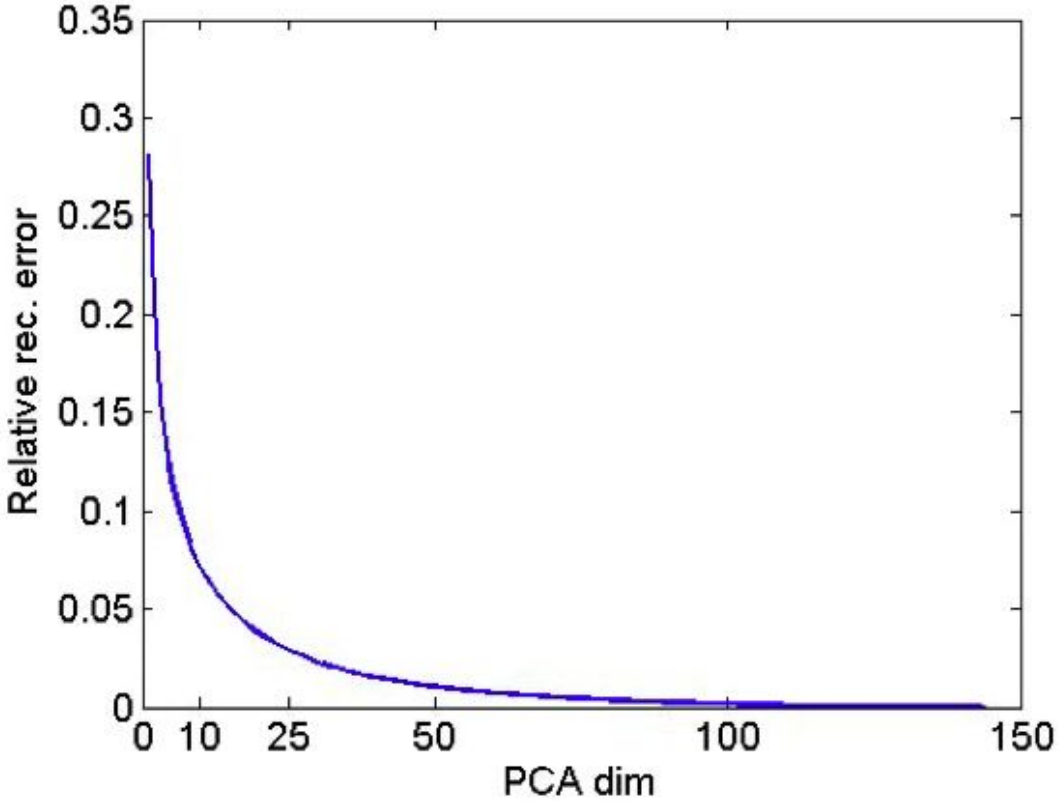


Image compression (60/144)

Image compression (16/144)

Image compression (16/144)

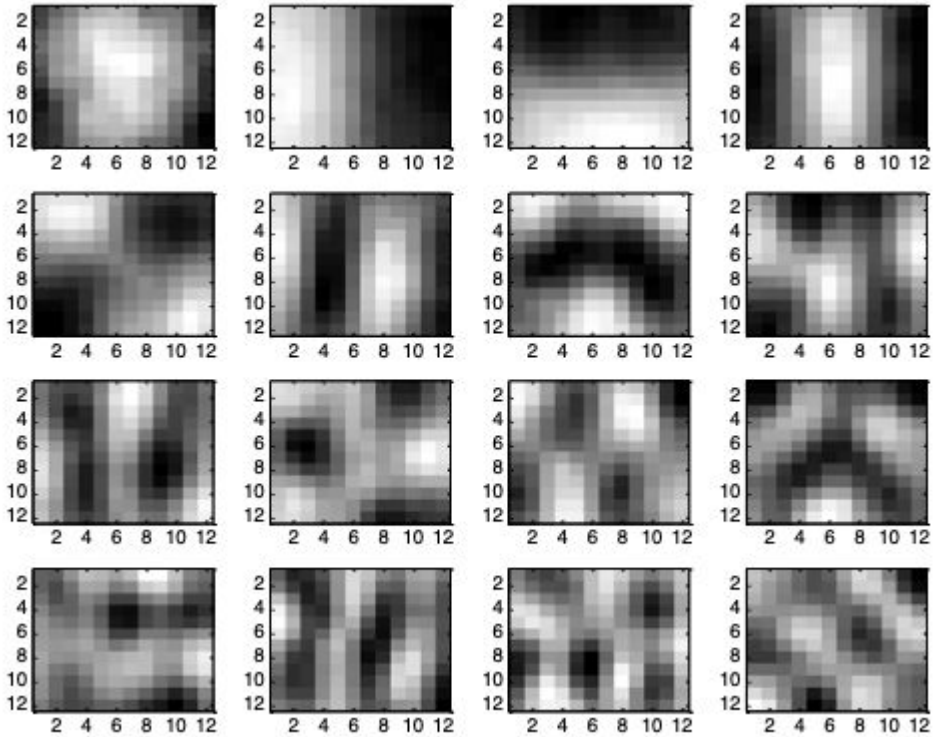


Image compression (6/144)

Image compression (6/144)

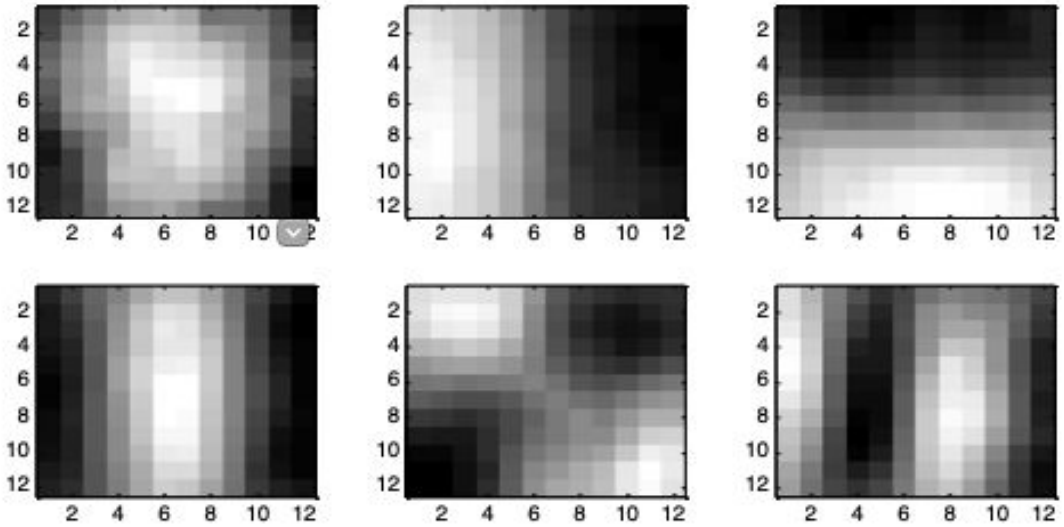


Image compression (3/144)

Image compression (3/144)

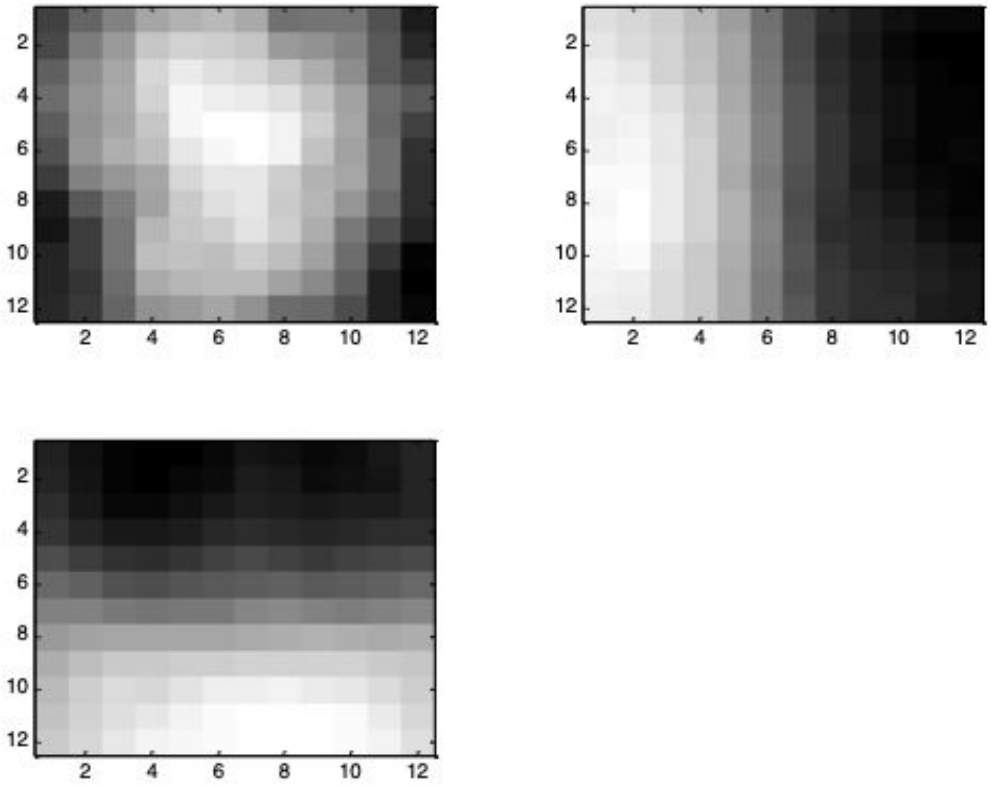
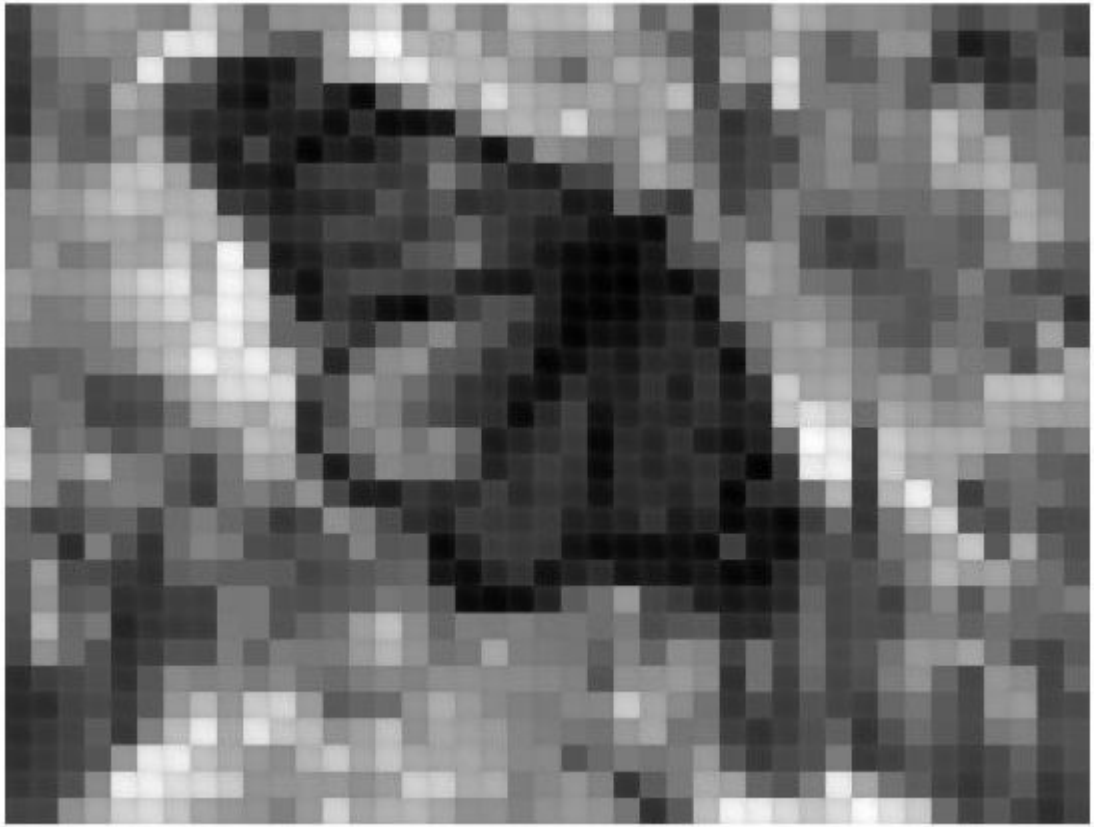


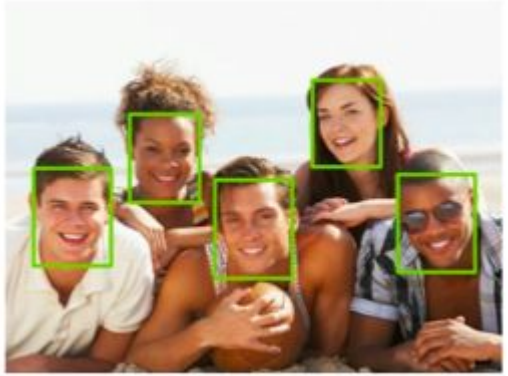
Image compression (1/144)

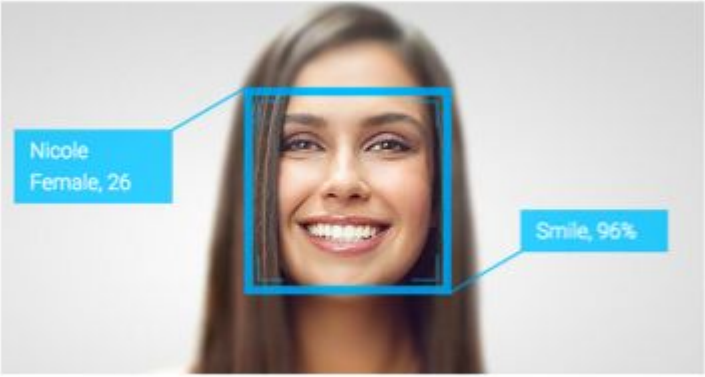


Face recognition

Face recognition

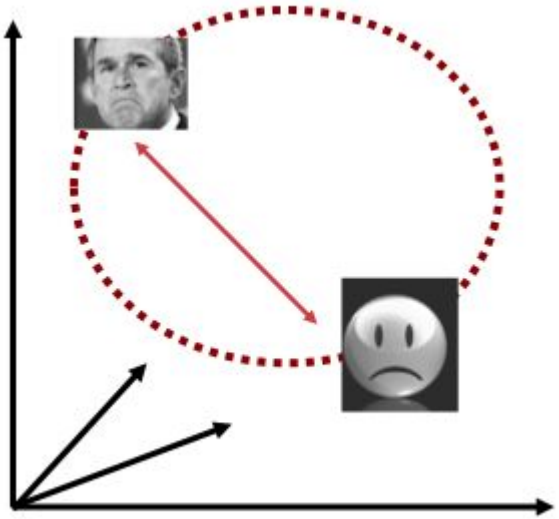
- Цифровая фотография
- Наблюдение
- Верификация
- Безопасность
- Memoji
- Организация альбомов
- ...





Пространство лиц (!)

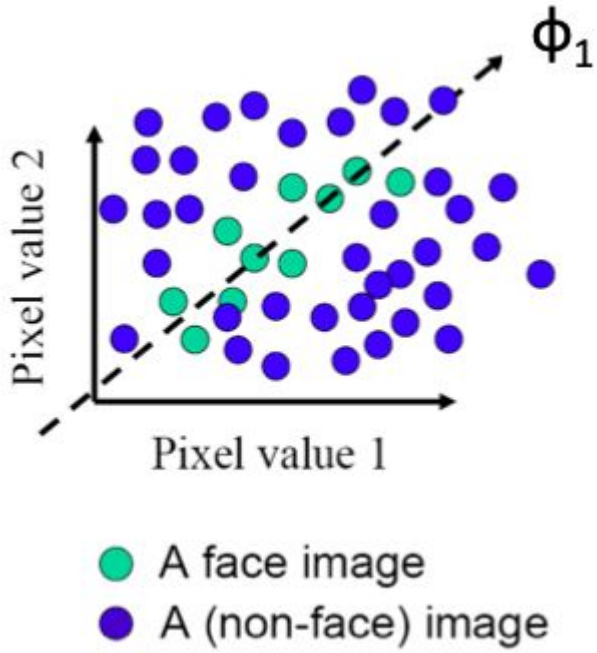
- Изображение - это точка в пространстве высокой размерности
  - Черно белое изображение размера NxM - это точка в пространстве R^NM
  - Например, 100x100 изображение = 10000 размерность



Пространство лиц (!)

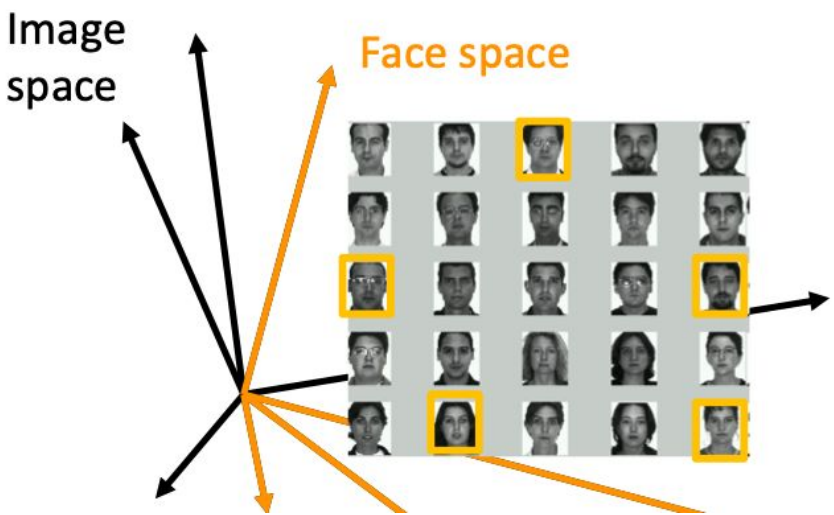
- Изображения 100x100 содержат не только изображения, но и много всего другого

Пространство лиц (!)



- Изображение - это точка в пространстве высокой размерности
  - Черно белое изображение размера NxM - это точка в пространстве R^NM
  - Например, 100x100 изображение = 10000 размерность
- Однако, лишь относительно небольшая часть точек в этом пространстве действительно соответствует лицам
- Мы хотим найти подпространство лиц!

Пространство лиц (!)



- Найти подпространство размерности n, такое что проекция точек датасета, на это подпространство, будет иметь наибольшую дисперсию

Пространство лиц (!)

- Проецируем на подпространство меньшей размерности, которое содержит ключевые характеристики
- PCA
- Сравнивать два изображения с лицам, проецируя их на пространство меньшей размерности и измеряя метрику расстояния между ними

Eigenfaces

Идея:

- предполагаем, что большинство изображений с лицами лежат в пространстве более низкой размерности определяемом k (k<<d) направлениями максимальной дисперсии
- используем PCA, чтобы определить собственные вектора (“eigenfaces”) которое задают это подпространство
- Представляем все изображения с лицами из датасета как линейную комбинацию eigenfaces

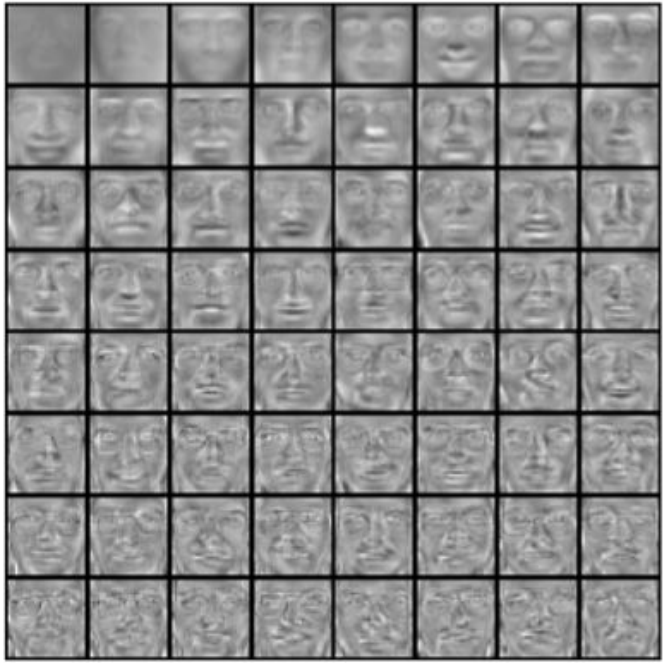
Eigenfaces

Training images: $\mathbf{x}_1,\ldots,\mathbf{x}_N$

Eigenfaces

Top eigenvectors: $\phi_1,\ldots,\phi_k$

Mean: μ



Eigenface algorithm (training)

1. Преобразовать изображения (матрицы) в точки (очень длинные векторы)

2. Подсчитать “среднее” лицо

$$\mu = \frac{1}{N}\sum x_i$$

3. Центрировать данные

$$X_c = \begin{bmatrix} | & & | \\ x_1 & \dots & x_n \\ | & & | \end{bmatrix} - \begin{bmatrix} | & & | \\ \mu & \dots & \mu \\ | & & | \end{bmatrix}$$

$$= X - \mu 1^T = X - \frac{1}{n} X 1 1^T = X\left(I - \frac{1}{n} 1 1^T\right)$$

Eigenface algorithm

4. Вычислить матрицу ковариации

$$\Sigma = \frac{1}{n}\begin{bmatrix} | & & | \\ x_1^c & \dots & x_n^c \\ | & & | \end{bmatrix}\begin{bmatrix} - & x_1^c & - \\ & \vdots & \\ - & x_n^c & - \end{bmatrix} = \frac{1}{n} X_c X_c^T$$

5. Найти собственные векторы ковариационной матрицы (SVD)

6. Найти проекции каждого изображения из обучающей выборки на эти собственные вектора

$$x_i \rightarrow \left(x_i^c \cdot \phi_1, x_i^c \cdot \phi_2, \dots, x_i^c \cdot \phi_K\right) \equiv \left(a_1, a_2, \dots, a_K\right)$$

7. Восстановить изначальное изображение

$$x_i \approx \mu + a_1\phi_1 + a_2\phi_2 + \dots + a_K\phi_K$$

Eigenface algorithm

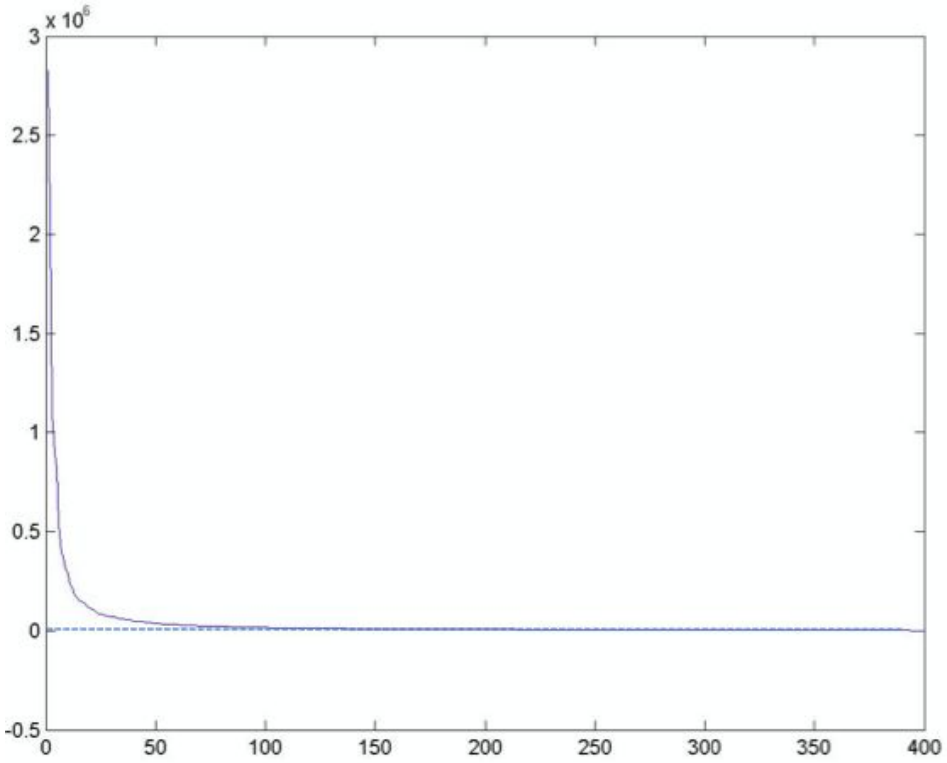
6. Найти проекции каждого изображения из обучающей выборки на эти собственные вектора

$$x_i \rightarrow \left(x_i^c \cdot \phi_1, x_i^c \cdot \phi_2, \ldots, x_i^c \cdot \phi_K\right) \equiv \left(a_1, a_2, \ldots, a_K\right)$$

7. Восстановить изначальное изображение

$$x_i \approx \mu + a_1\phi_1 + a_2\phi_2 + \ldots + a_K\phi_K$$

Eigenface (variance)



Eigenfaces

Выбирая K наилучших eigenfaces -> уменьшаем размерность

Маленькое количество eigenfaces приводит к слишком большой потери информации, а следовательно мы хуже сможем различать лица

Eigenface algorithm (testing)

1. Берем тестовое изображение с лицом t
2. Проецируем его в найденное подпространство и вычисляем коэффициенты проекции

$$t \rightarrow \left((t-\mu)\cdot\phi_1, (t-\mu)\cdot\phi_2, \ldots, (t-\mu)\cdot\phi_K\right) \equiv \left(w_1, w_2, \ldots, w_K\right)$$

3. Сравниваем проекцию w со всеми N проекциями тренировочных изображений
   a. Метрика сравнения: Euclidean
   b. Классификация: k-NN

Eigenfaces

- Алгоритм очень требователен к данным
  - Все лица центрированы
  - Одинакового размера
  - Ограниченный угол поворота головы
    - Альтернатива: получить PCA для каждого из возможных углов
    - Использовать тот, на котором достигается наименьшая ошибка
- Методы сжатия - полностью unsupervised

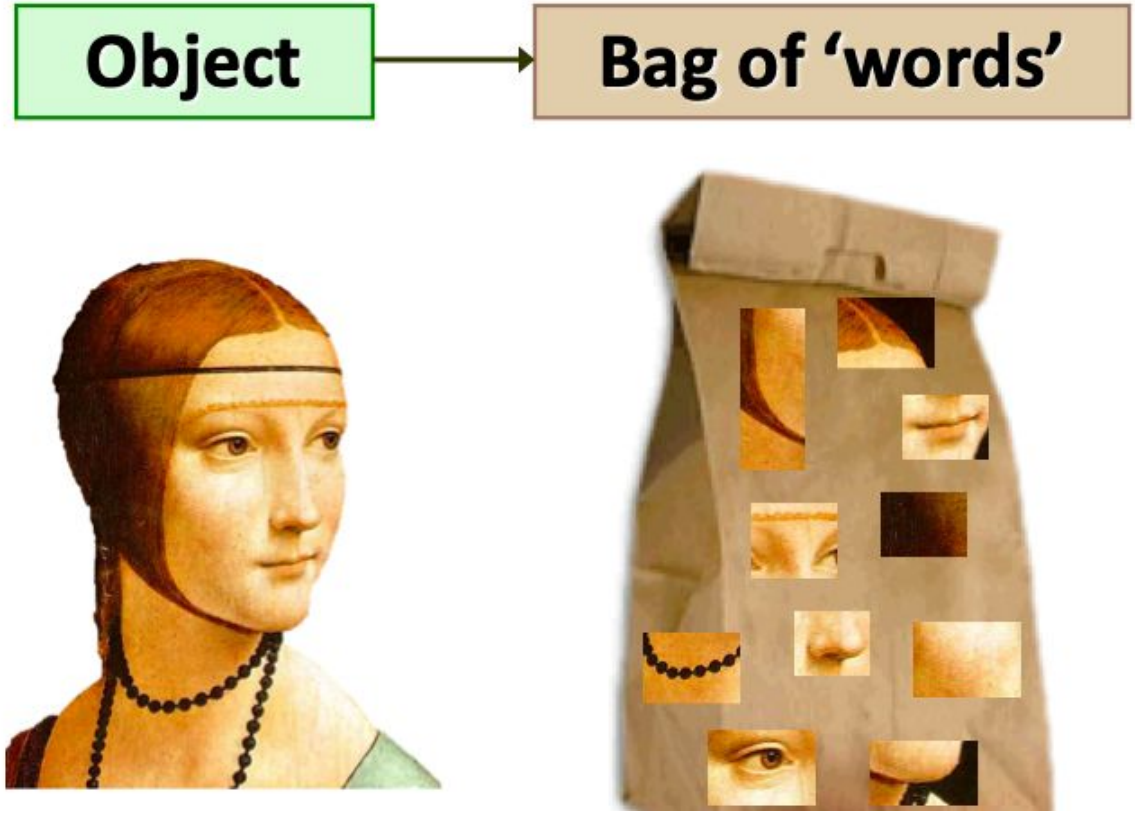
- (иногда это хорошо!)
- Не учитывает, что лица - это 3D объекты
- Нет никакого механизма сохранения распределения классов

Eigenfaces

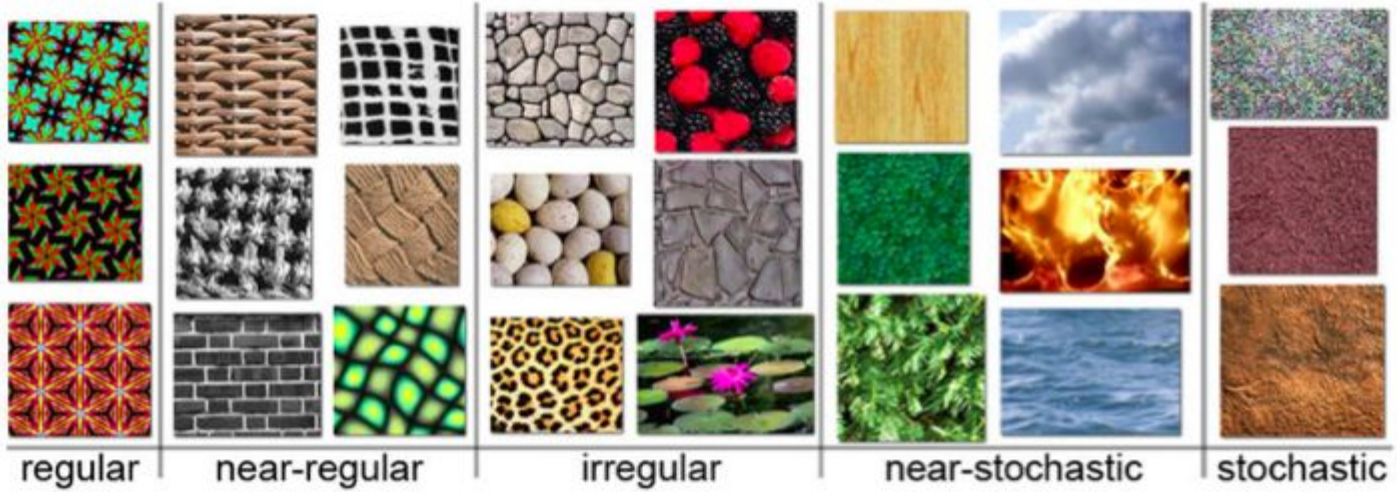
- Pros:
  - Не итеративный
  - Достигает глобального оптимума
- Cons:
  - Плохо интерпретируется
  - Не гибкий
- Ограничения
  - PCA - отлично подходит для восстановления из пространства низкой размерности (сжатие, разжатие), но плохо подходит для векторного описания

Bag of visual words

BoW

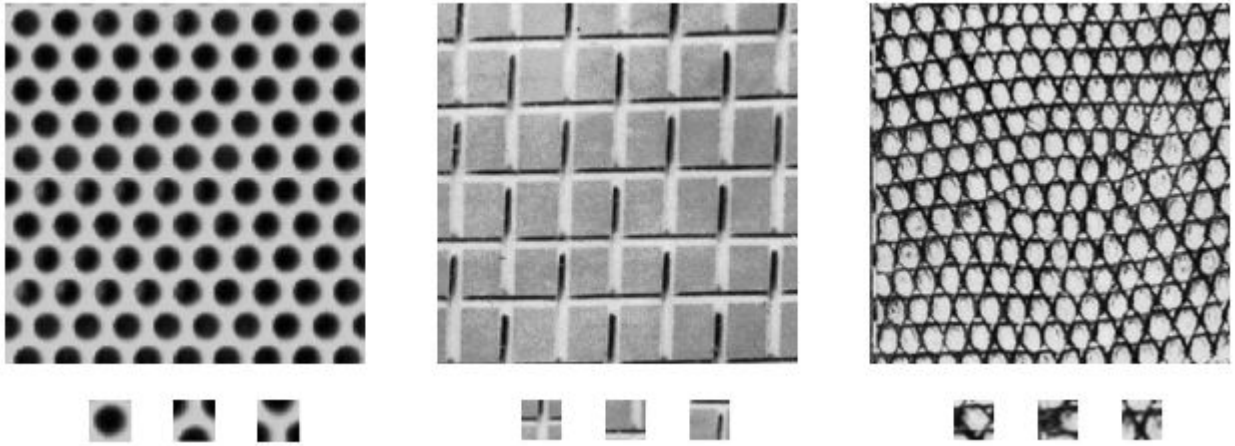


Текстуры

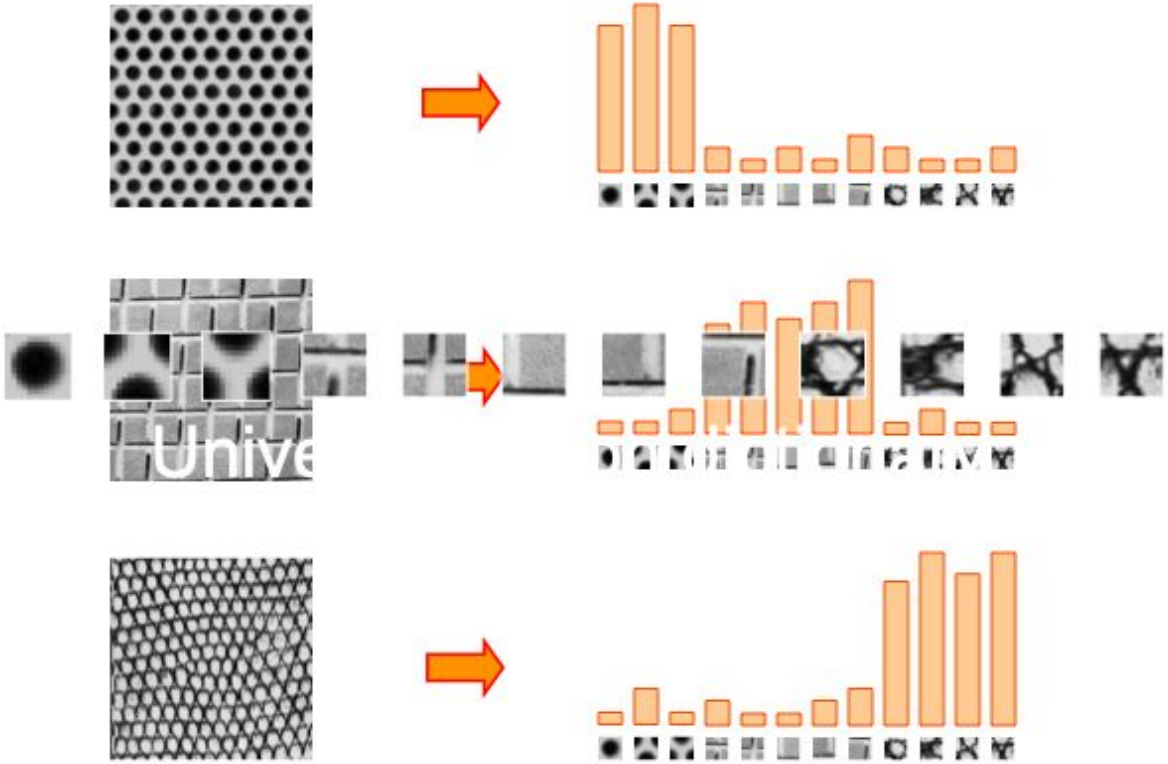

regular
near-regular
irregular
near-stochastic
stochastic

Текстуры

Текстуры характеризуются повторяющимися элементами (**текстонами)** и их частотой



Текстуры



BoW models

Представление текстового документа, не учитывающее порядок слов, а только частоту слов в документе

BoW models

Представление текстового документа, не учитывающее порядок слов, а только частоту слов в документе

BoW models

Представление текстового документа, не учитывающее порядок слов, а только частоту слов в документе

BoW models

Представление текстового документа, не учитывающее порядок слов, а только частоту слов в документе

Bag of features

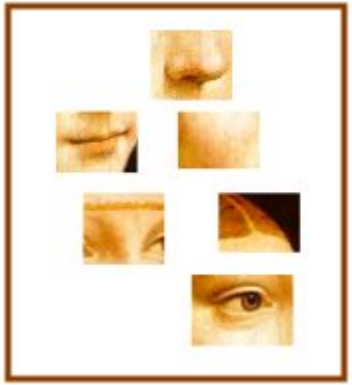
Работает довольно хорошо для задач классификации, определения, что вообще изображено на картинке

Bag of features

- Берем много изображений, извлекаем из них кучу признаков (например, SIFTом), и строим “визуальный словарь” - список общих фич
- С новым изображением - извлекаем фичи, строим гистограмму частот - для каждой фичи, находим наиближайшее слово в словаре

Bag of features

1. Извлекаем фичи

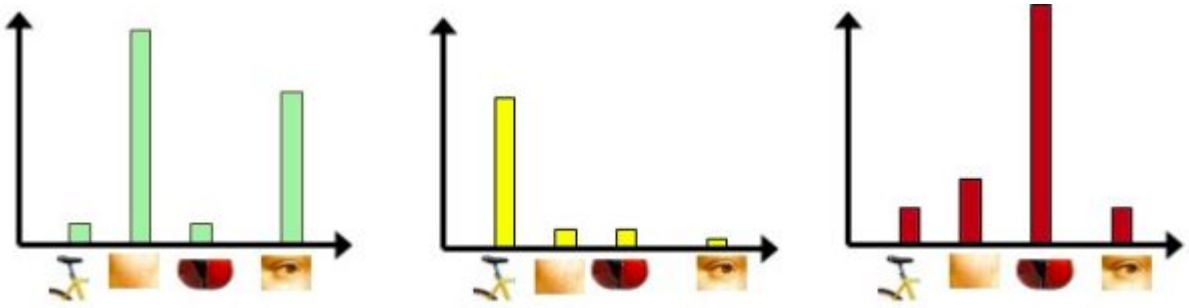


Bag of features

1. Извлекаем фичи
2. Строим “визуальный словарь”

Bag of features

1. Извлекаем фичи
2. Строим “визуальный словарь”
3. Для каждой фичи находим ближайшее “слово” из словаря
4. Строим гистограмму частот “слов” для словаря



Bag of features

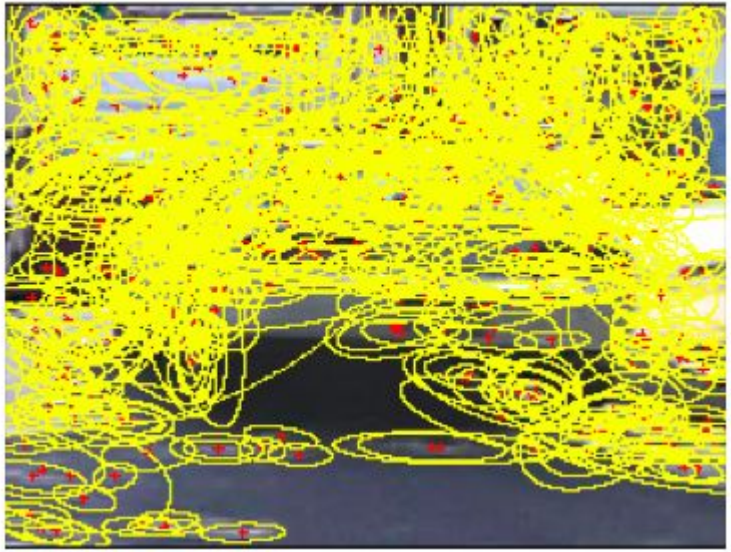
1. Извлекаем фичи

- Делаем сетку на изображении
  - Vogel & Schiele, 2003
  - Fei-Fei & Perona, 2005

Bag of features

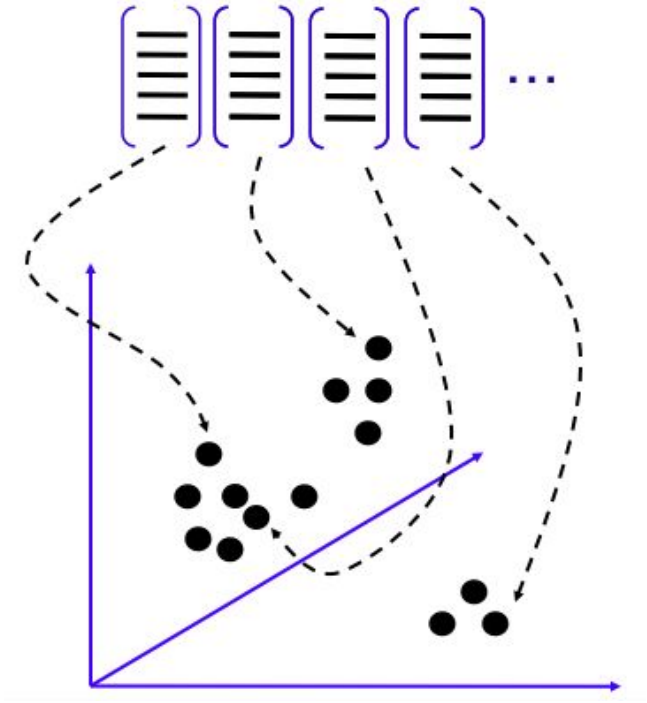
1. Извлекаем фичи

- Делаем сетку на изображении
  - Vogel & Schiele, 2003
  - Fei-Fei & Perona, 2005
- Детекторы ключевых точек
  - Csurka et al. 2004
  - Fei-Fei & Perona, 2005
  - Sivic et al. 2005



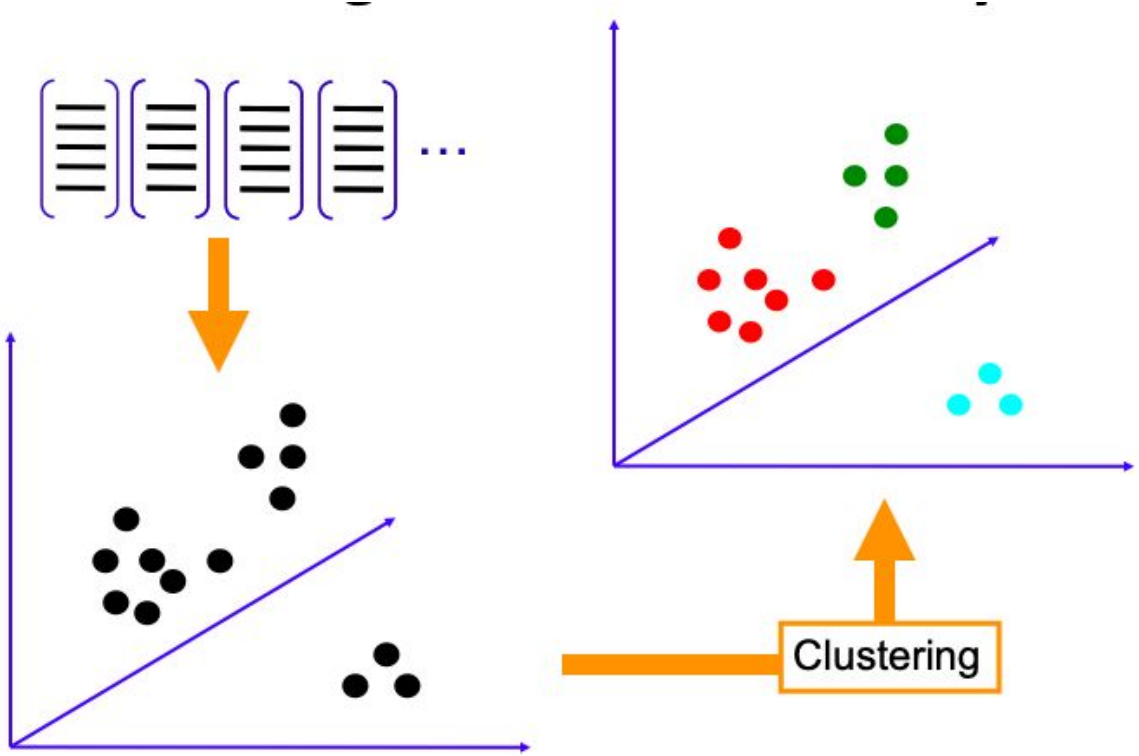
Bag of features

2. Строим визуальный словарь



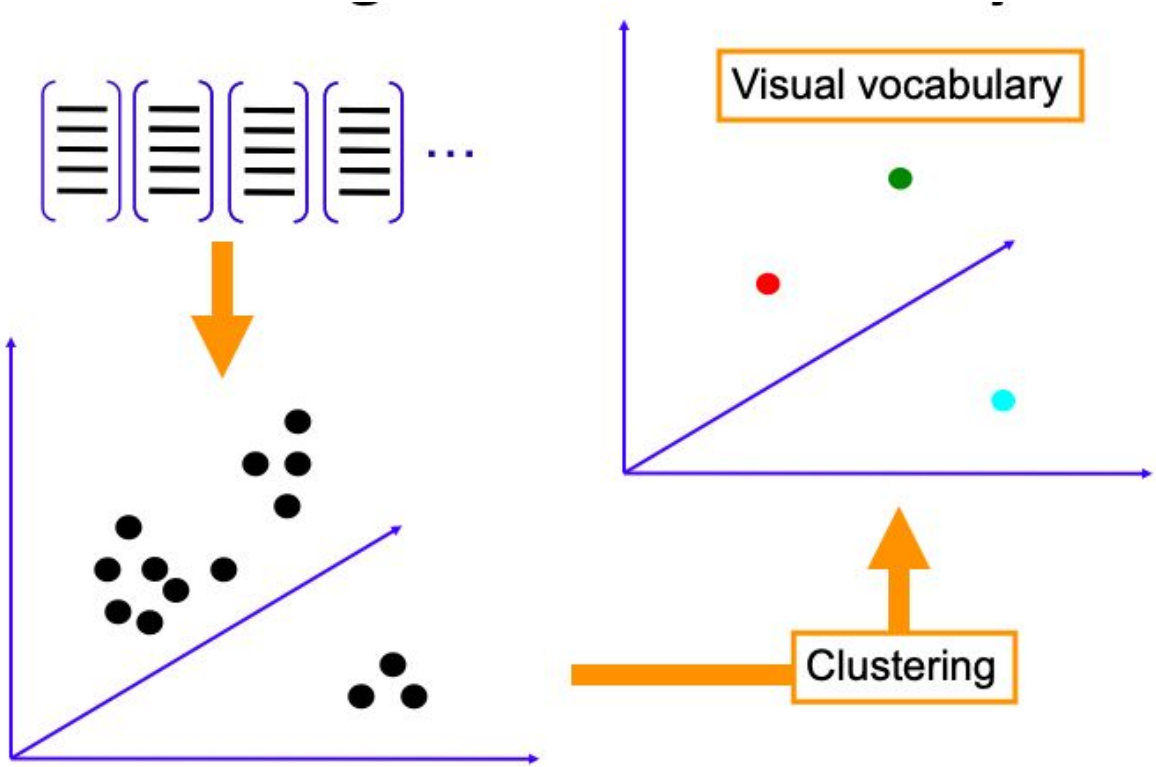
Bag of features

2. Строим визуальный словарь



Bag of features

2. Строим визуальный словарь



K-means

- Хотим минимизировать сумму квадратов расстояний между точками x_i и ближайшими к ним центров кластеров m_k

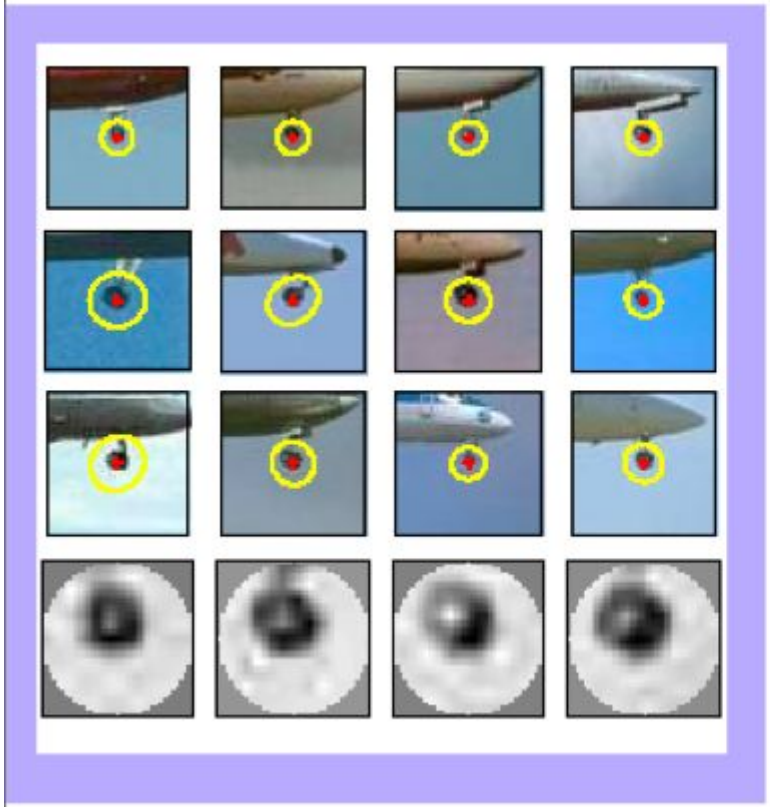
$$D(X,M) = \sum_{\text{cluster } k} \sum_{\substack{\text{point } i \text{ in} \\ \text{cluster } k}} (x_i - m_k)^2$$

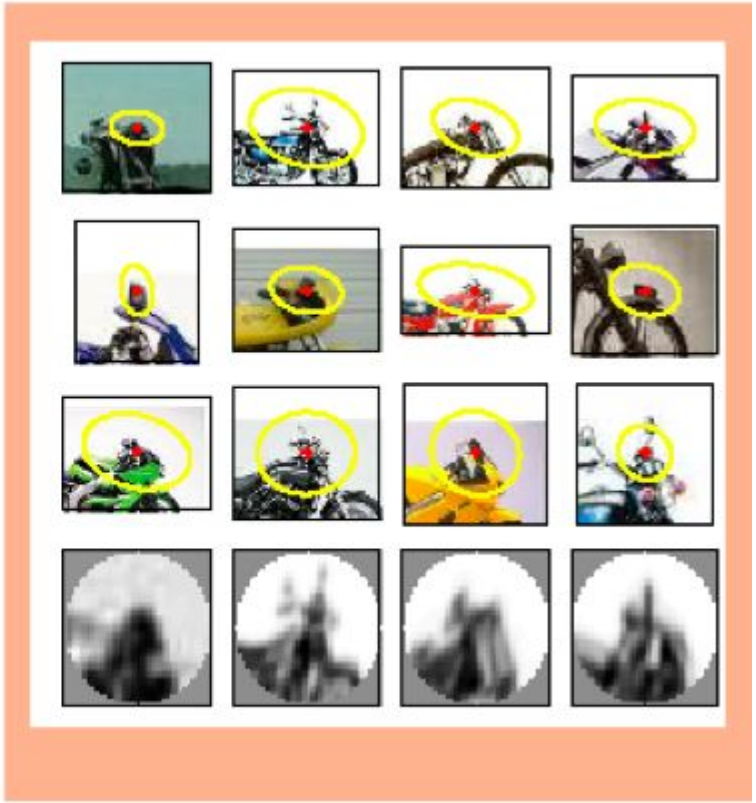
- Алгоритм:
- Рандомно инициализируем K центров кластеров
- Итерироваться пока не сойдется
  - Назначаем каждую точку к ближайшему центру
  - Пересчитываем каждый центр кластера, как среднее его точек

K-means

- Кластеризация - хороший метод для построения визуального словаря:
  - Unsupervised
  - Каждый центр кластера, полученный через k-means, становится “визуальным словом” в словаре
  - Словарь строится на обучающей выборке
  - Если обучающая выборка, довольно репрезентативна, то словарь становится универсальным
- Словарь используется для описания каждой фичи
  - Берем feature vector, смотрим на ближайшее к нему слово из словаря, теперь feature vector - просто индекс этого слова

Пример визуальных слов





BoW: проблемы

- Как выбрать размер словаря?
  - Слишком маленький: недостаточно репрезентативен
  - Слишком большой: переобучение
- Вычислительная сложность
  - Трудно найти “ближайшего” соседа, если соседей 100_000

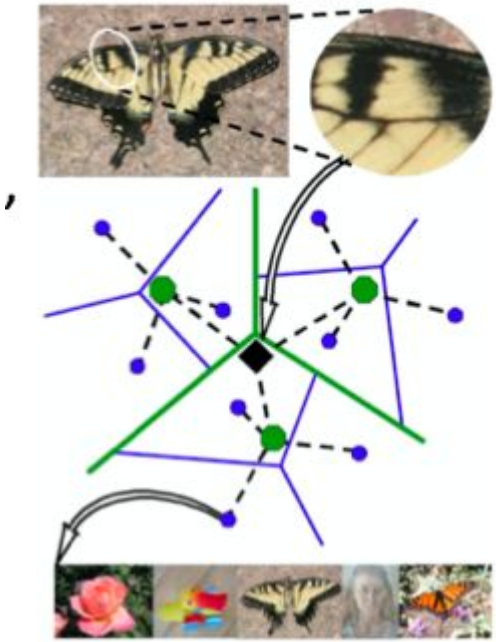


Image representation

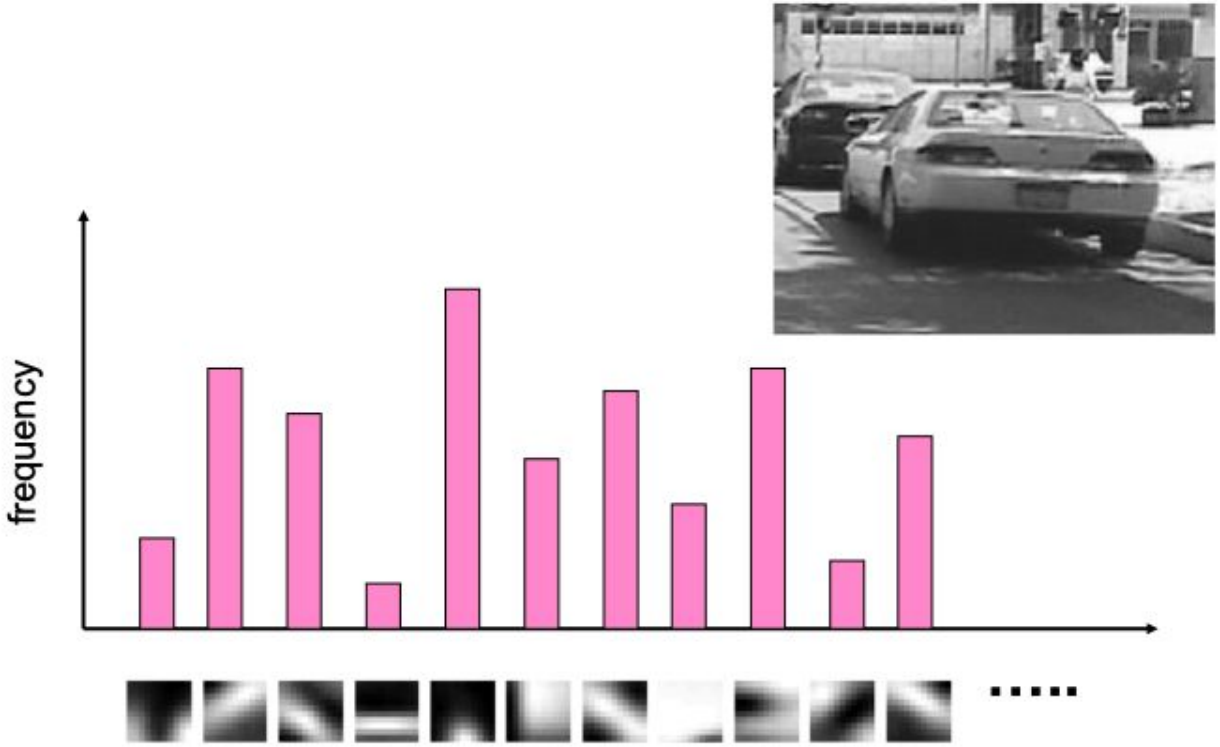
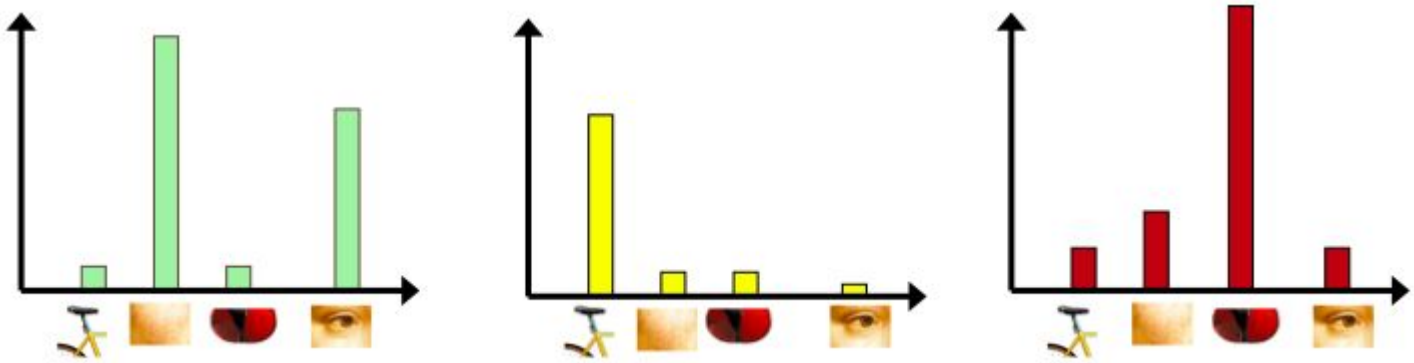


Image representation

- Теперь у нас есть векторное описание каждого изображения (частота слов из словаря). Что с ними делать дальше?

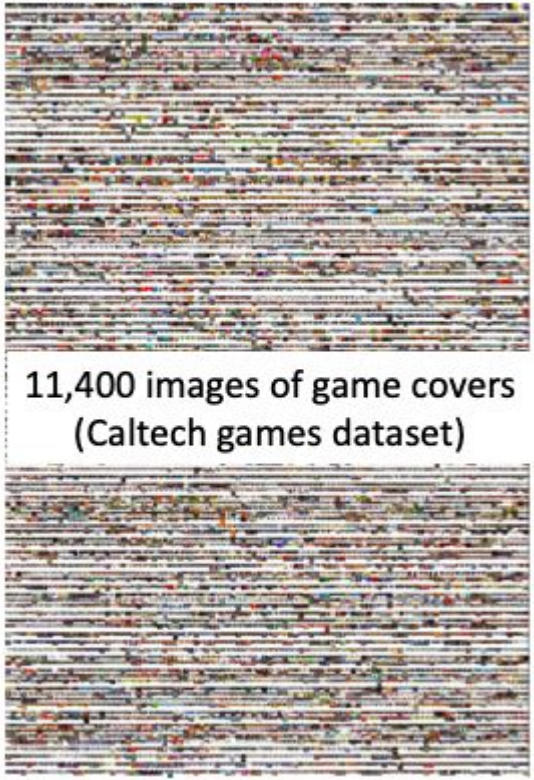


BoVW репрезентация

- Использовать как векторное описание для стандартных классификаторов (например, k-NN, SVM)

- Кластеризовать BoVW векторы
  - Находим визуальные темы в изображении!

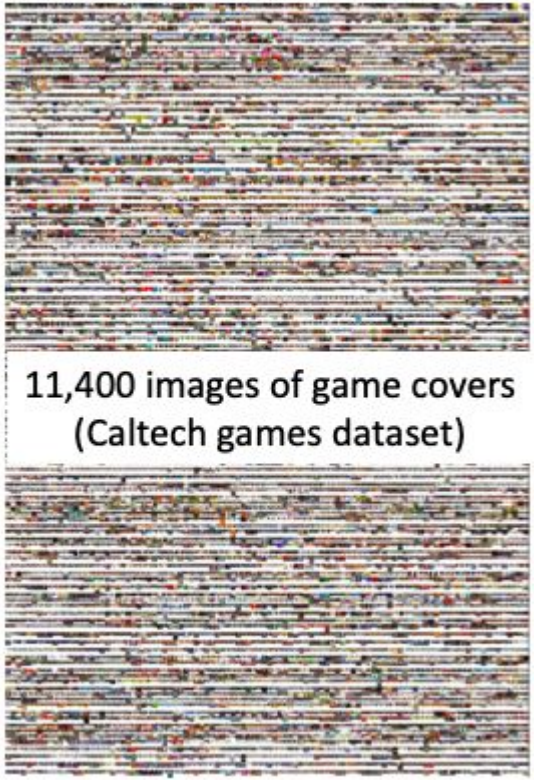
Large-scale image search



BoW модели полезны в задаче поиска изображений.

Как найти это изображение в датасете?

Large-scale image search



Строим базу данных изображений

- Извлекаем фичи из изображений
- Строим словарь с помощью k-means (обычно, k порядка 100_000)
- Считаем *вес* для каждого слова

Weighting words

- Как и в случае с текстом, некоторые визуальные слова уникальнее, чем другие

и, а, когда **vs.** *корова, хоккей, банк*

- Чем больше количество документов, в которых встречается слово, тем оно менее репрезентативно
  - Например, если слово встречается во всех документах, то оно бесполезно

TF-IDF

- Вместо того, чтобы строить гистограмму частот слов, мы будем учитывать какое у слова inverse document frequency (IDF)

- IDF слова j = $\log \frac{\text{number of documents}}{\text{number of documents in which } j \text{ appears}}$

TF-IDF

- Значения столбца j в изображении i:

term frequency of *j* in *I* **x** *inverse document frequency* of *j*

- j - столбец/визуальное слово/центр кластера
- i - изображение/документ

Inverted file

- В каждом изображении около 1000 признаков
- В словаре около 100_000 визуальных слов
  - Гистограммы получаются разряженными (sparse), почти везде нули

- Inverted file:
  - отображение из слов в документы

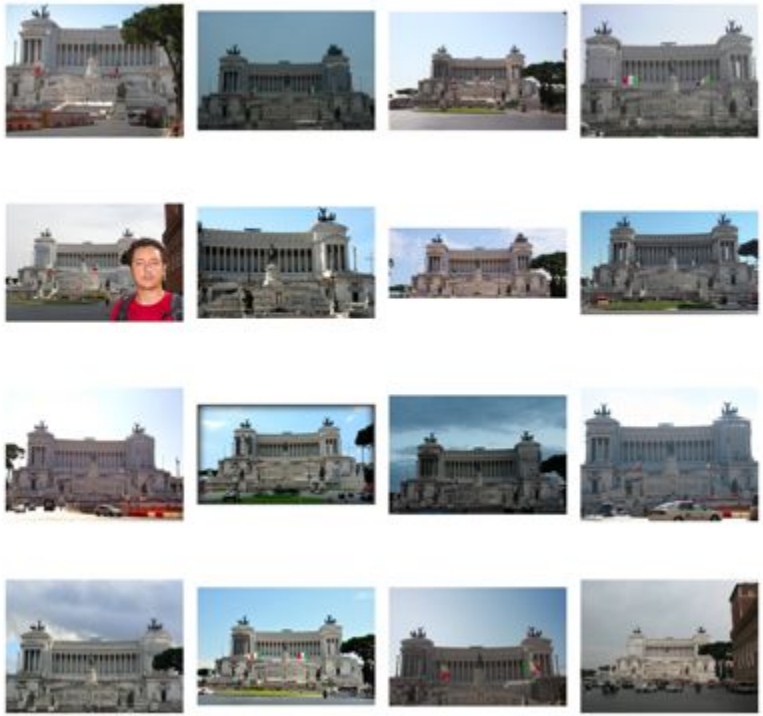
```
"a":      {2}
"banana": {2}
"is":     {0, 1, 2}
"it":     {0, 1, 2}
"what":   {0, 1}
```

Inverted file

- Приходит новое изображение
  - Сравниваем его только с теми изображениями из бд, у которых хоть как-нибудь пересекаются гистограммы

```
"a":      {2}
"banana": {2}
"is":     {0, 1, 2}
"it":     {0, 1, 2}
"what":   {0, 1}
```

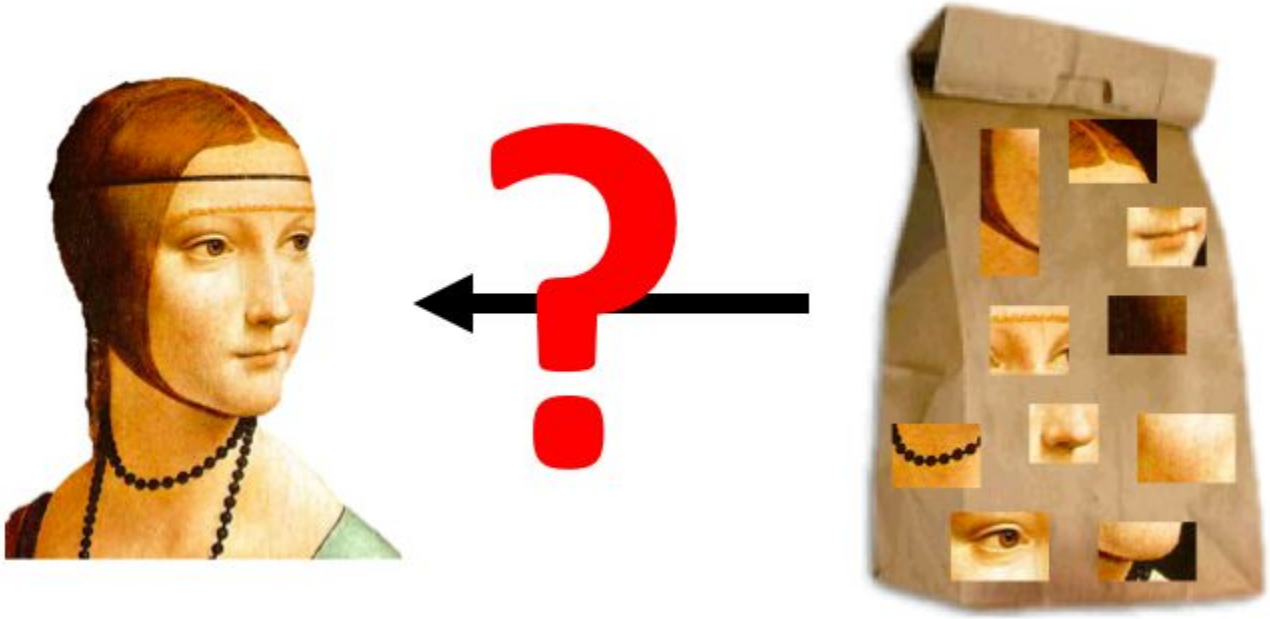
Пример, BoW search



Пример, BoW search

Spatial pyramid matching

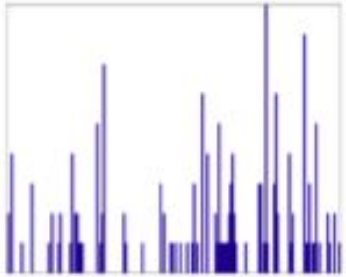
Пространственная информация



Pyramide matching

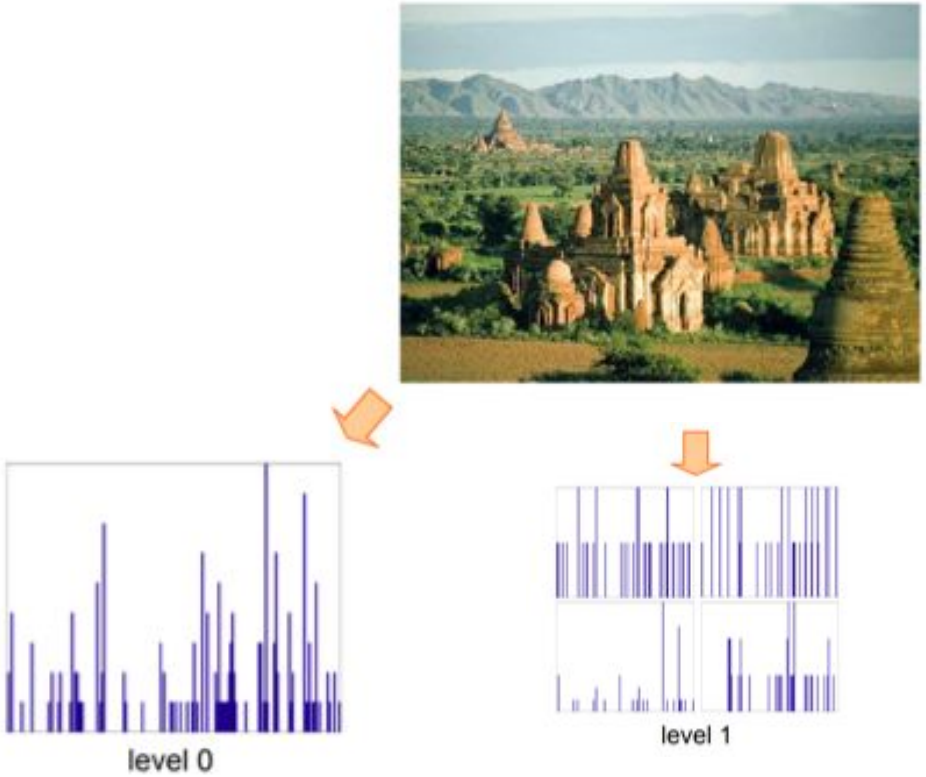
- Очень полезное представление изображений
- Пирамиды строятся на уменьшенных копиях изображения
- Каждый уровень пирамиды - 1/4 размера предыдущего уровня
- Самый низкий уровень - самое высокое разрешение
- Самый высокий уровень - самое низкое разрешение

Pyramide matching + BoW



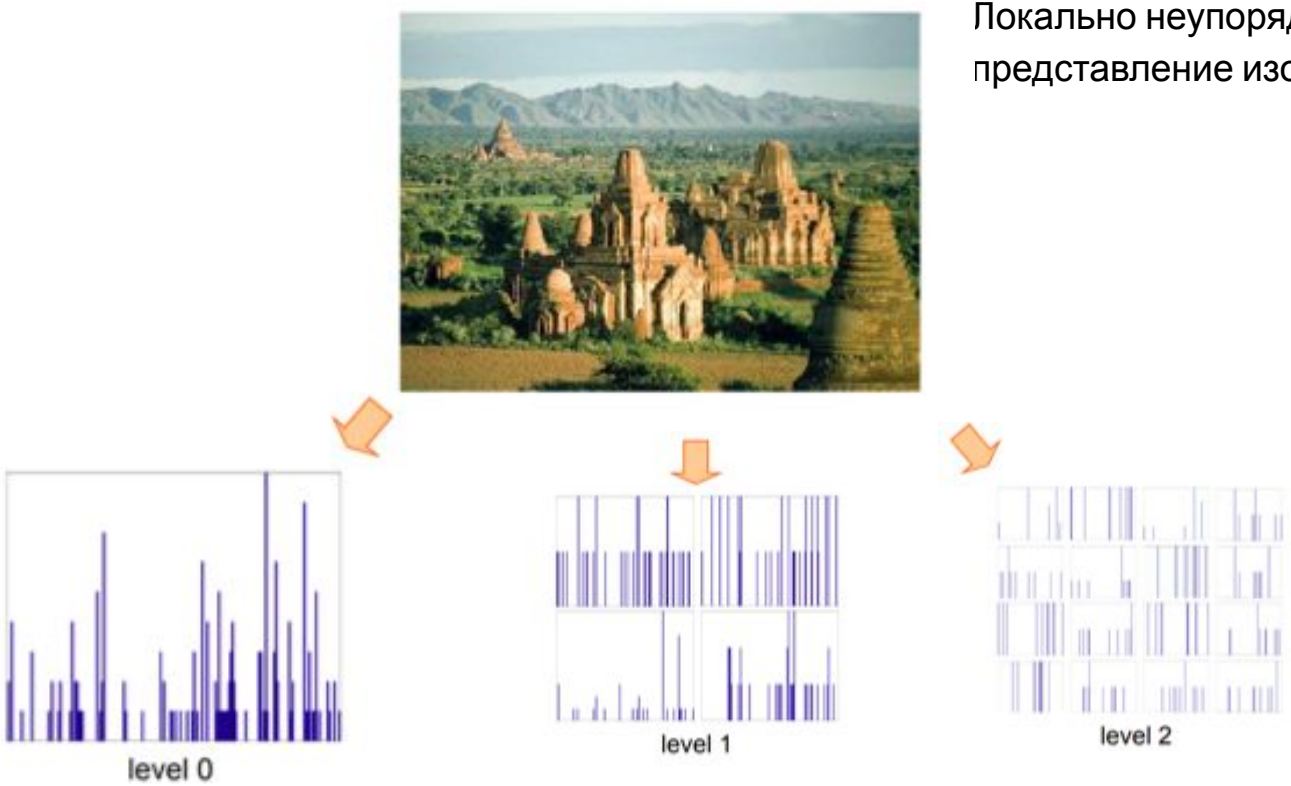
Покально неупорядоченное представление изображения

Pyramide matching + BoW



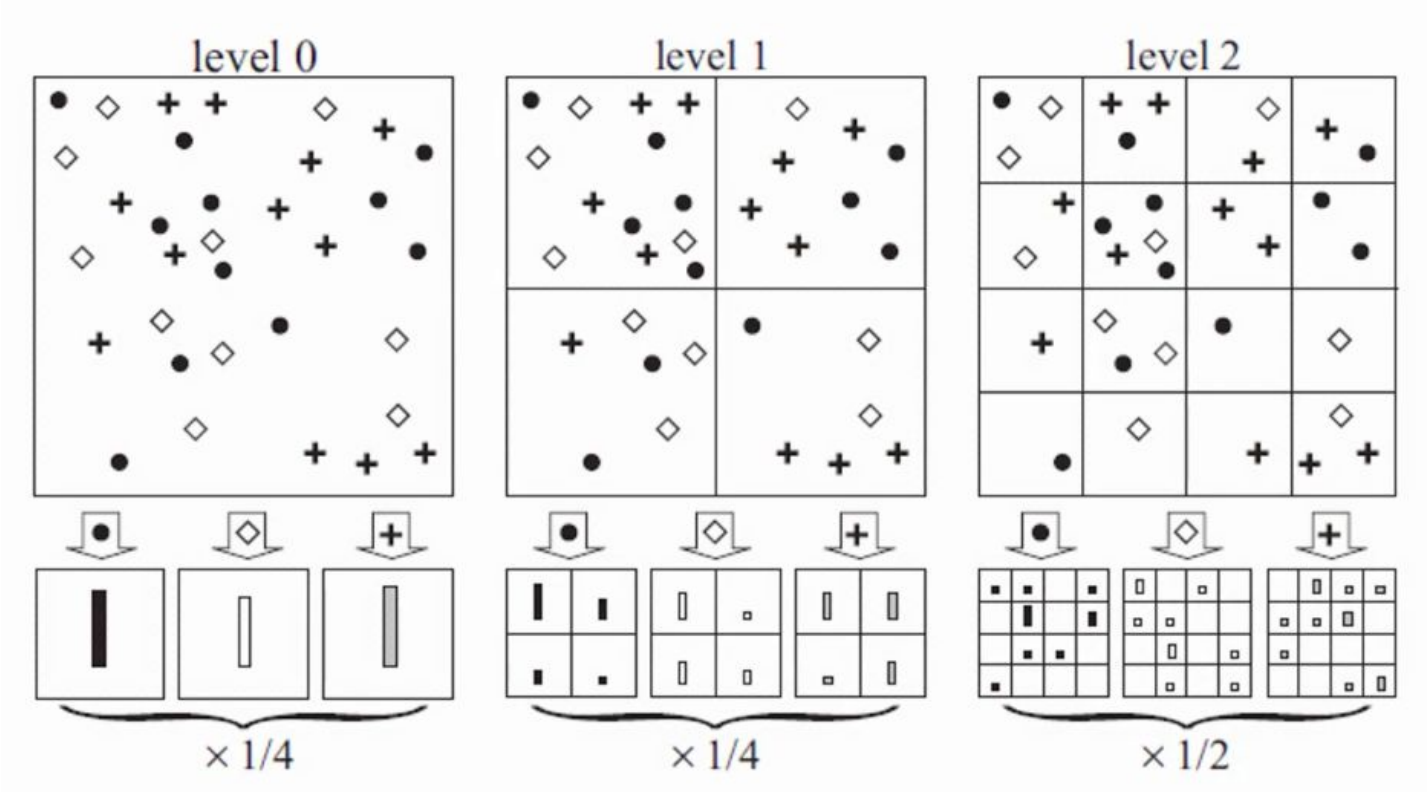
Покально неупорядоченное представление изображения

Pyramide matching + BoW



Покально неупорядоченное представление изображения

Pyramide matching + BoW



It’s coding time!