

OpenCV

Преобразования



Операции над изображением

- Аффинные и неаффинные преобразования
- Перенос
- Поворот
- Масштабирование, интерполяция
- Пирамида изображений
- Обрезка
- Арифметические операции
- Побитовые операции, маски
- Свёртка и размытие
- Резкость
- Пороговое преобразование, бинаризация
- Морфологические операции
- Перспективные преобразования

Виды преобразований на плоскости

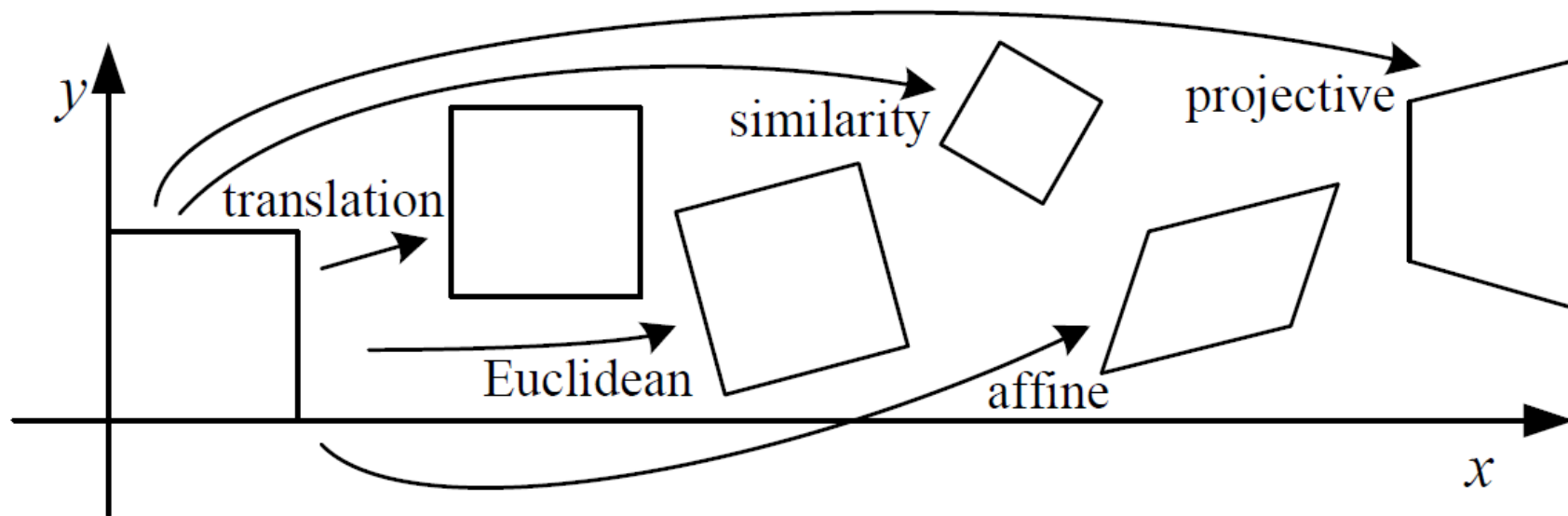

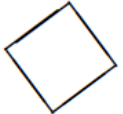
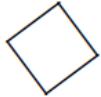

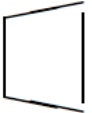


Figure 2: *Basic set of 2D planar transformations*

Виды преобразований на плоскости

Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\left[\begin{array}{c c} \mathbf{I} & \mathbf{t} \end{array} \right]_{2 \times 3}$	2	orientation + ...	
rigid (Euclidean)	$\left[\begin{array}{c c} \mathbf{R} & \mathbf{t} \end{array} \right]_{2 \times 3}$	3	lengths + ...	
similarity	$\left[\begin{array}{c c} s\mathbf{R} & \mathbf{t} \end{array} \right]_{2 \times 3}$	4	angles + ...	
affine	$\left[\begin{array}{c} \mathbf{A} \end{array} \right]_{2 \times 3}$	6	parallelism + ...	
projective	$\left[\begin{array}{c} \tilde{\mathbf{H}} \end{array} \right]_{3 \times 3}$	8	straight lines	

Однородные координаты

Для решения задач преобразования 3D-пространства и 2D-плоскости в единообразном (матричном) виде вводится формализм однородных координат.

Однородными координатами служат тройки чисел (одновременно не равные нулю), такие что:

$$\begin{pmatrix} \bar{x} \\ \bar{y} \\ w \end{pmatrix} = w \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

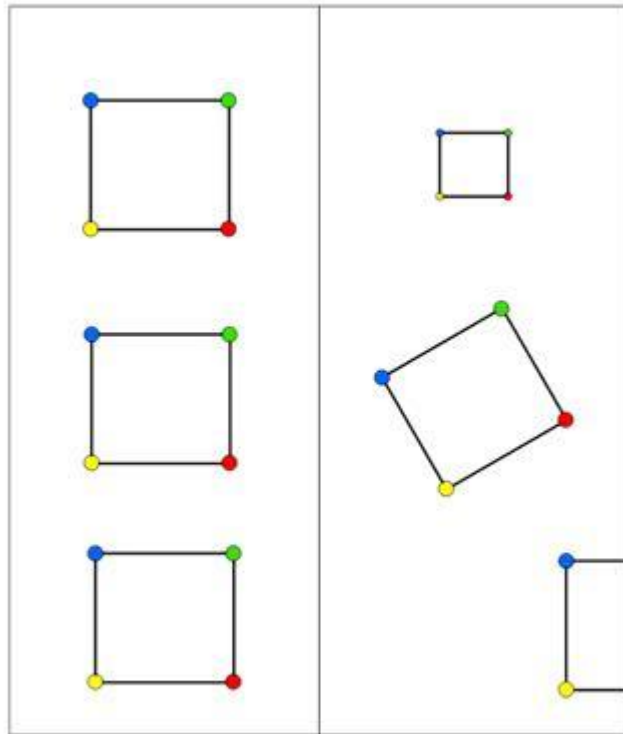
(x, y — координаты плоскости)

Зачем?

- В декартовых координатах невозможно описать бесконечно удаленную точку. А многие математические и геометрические концепции значительно упрощаются, если в них используется понятие бесконечности. Например, у бесконечно удаленной точки будет проекция на экран. ($w = 0$)
- Можно использовать унифицированный механизм работы с матрицами для выражения преобразований точек. С помощью матриц 3×3 можно описать вращение и масштабирование, однако сдвиг описать нельзя
- Можно использовать матричную запись для перспективного преобразования.

Простейшие аффинные преобразования

Аффинное преобразование (от *лат. affinis* — соприкасающийся, близкий, смежный) — отображение плоскости или пространства в себя, при котором параллельные прямые переходят в параллельные прямые, пересекающиеся — в пересекающиеся, скрещивающиеся — в скрещивающиеся.



- масштабирование

- поворот

- перемещение

Произведение матриц

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix}$$

$$(1, 2, 3) \cdot (8, 10, 12) = 1 \times 8 + 2 \times 10 + 3 \times 12 = 64$$

We can do the same thing for the **2nd row** and **1st column**:

$$(4, 5, 6) \cdot (7, 9, 11) = 4 \times 7 + 5 \times 9 + 6 \times 11 = 139$$

And for the **2nd row** and **2nd column**:

$$(4, 5, 6) \cdot (8, 10, 12) = 4 \times 8 + 5 \times 10 + 6 \times 12 = 154$$

And we get:

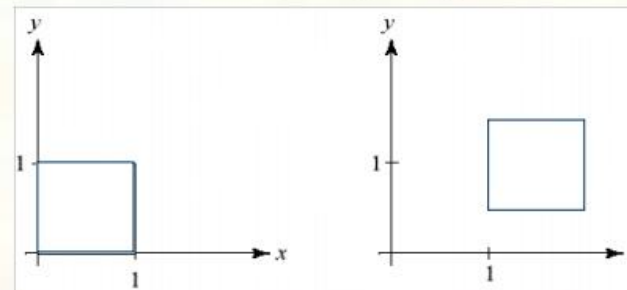
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix} \quad \checkmark$$

$$\mathbf{p}' = \mathbf{M}_T \mathbf{p}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$x' = x + t_x$$

$$y' = y + t_y$$



$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1/2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{dst}(x, y) = \text{src}(M_{11}x + M_{12}y + M_{13}, M_{21}x + M_{22}y + M_{23})$$

Матрица преобразования

В **OpenCV** аффинные преобразования осуществляются функцией **cvWarpAffine()**:

```
cv2.warpAffine(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]])
```

- **src** – input image.
- **dst** – output image that has the size **dsize** and the same type as **src**.
- **M** – 2x3 transformation matrix.
- **dsize** – size of the output image.
- **flags** – combination of interpolation methods (see [resize\(\)](#)) and the optional flag **WARP_INVERSE_MAP** that means that **M** is the inverse transformation ().
- **borderMode** – pixel extrapolation method (see [borderInterpolate\(\)](#)); when **borderMode=BORDER_TRANSPARENT**, it means that the pixels in the destination image corresponding to the “outliers” in the source image are not modified by the function.
- **borderValue** – value used in case of a constant border; by default, it is 0.

$$\text{dst}(x, y) = \text{src}(M_{11}x + M_{12}y + M_{13}, M_{21}x + M_{22}y + M_{23})$$

Перемещение

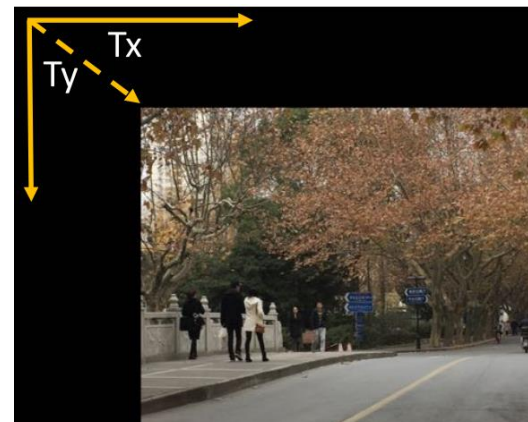
Трансляционная матрица:

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

t_x – величина смещения по оси x (горизонтальное)

t_y – величина смещения по оси y (вертикальное)

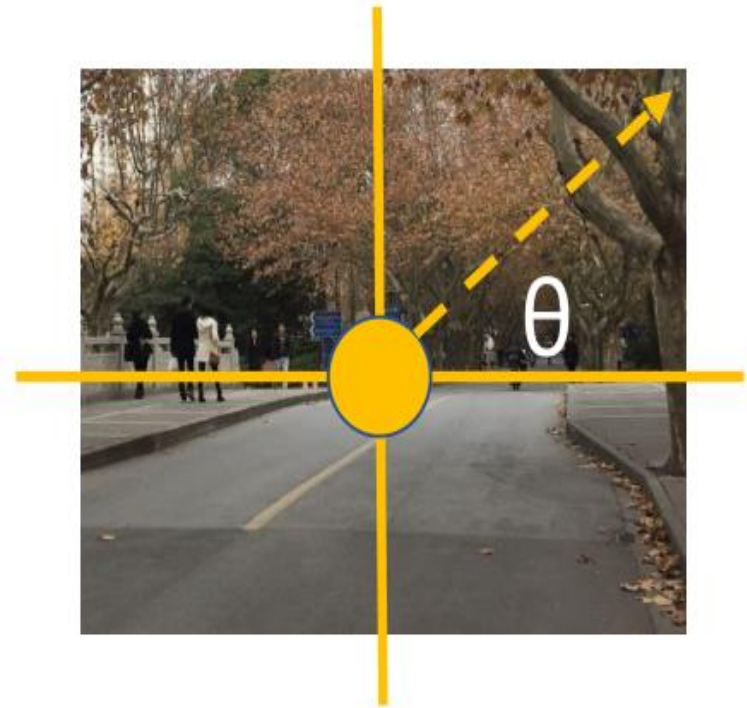
```
1 T
array([[ 1. ,  0. , 311.25],
       [ 0. ,  1. , 207.5 ]], dtype=float32)
```



Поворот

$$M = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

θ – угол поворота



`cv.GetRotationMatrix2D(center, angle, scale, mapMatrix)`

$$\begin{bmatrix} \alpha & \beta & (1 - \alpha) \cdot \text{center.x} - \beta \cdot \text{center.y} \\ -\beta & \alpha & \beta \cdot \text{center.x} + (1 - \alpha) \cdot \text{center.y} \end{bmatrix}$$

где $\alpha = \text{scale} \cdot \cos \text{angle}$,
 $\beta = \text{scale} \cdot \sin \text{angle}$

Поворот (транспонирование)

```
img = cv2.imread('input.jpg')
```

```
rotated_image = cv2.transpose(img)
```

```
cv2.imshow('Rotated Image - Method 2', rotated_image)
```

```
cv2.waitKey()
```

```
cv2.destroyAllWindows()
```



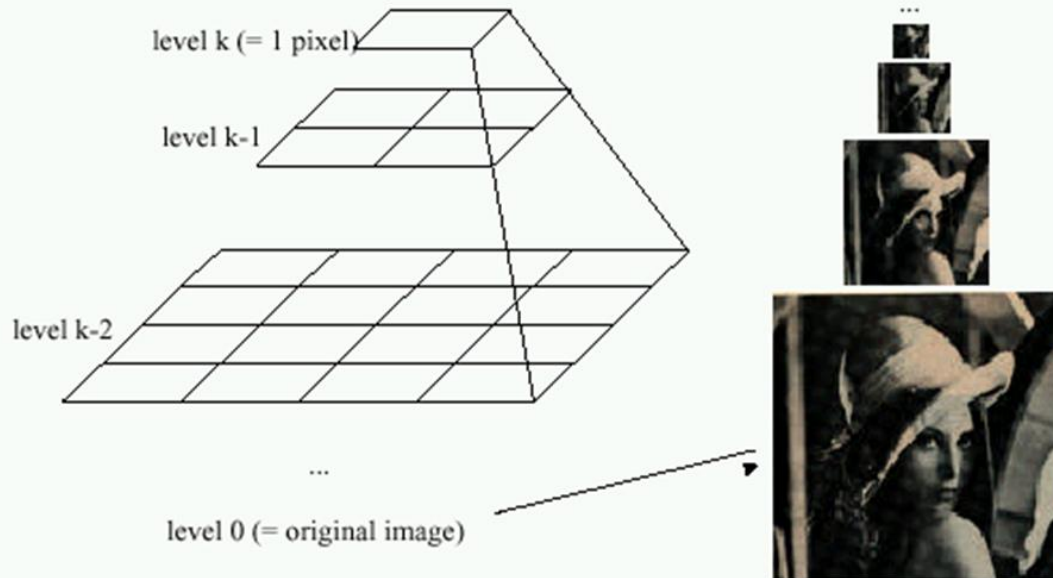
Горизонтальное отображение

```
flipped = cv2.flip(image, 1)  
cv2.imshow('Horizontal Flip', flipped)  
cv2.waitKey()  
cv2.destroyAllWindows()
```



Пирамиды изображений

Idea: Represent $N \times N$ image as a “pyramid” of $1 \times 1, 2 \times 2, 4 \times 4, \dots, 2^k \times 2^k$ images (assuming $N = 2^k$)



- Известна как Пирамида Гауссиан
- В компьютерной графике – “mip map” [Williams, 1983]

Пирамиды изображений



512

256

128

64

32

16

8



На высшем уровне
полоса в пиксель –волос,
на среднем – полоска,
на нижнем - нос

Figure from David Forsyth

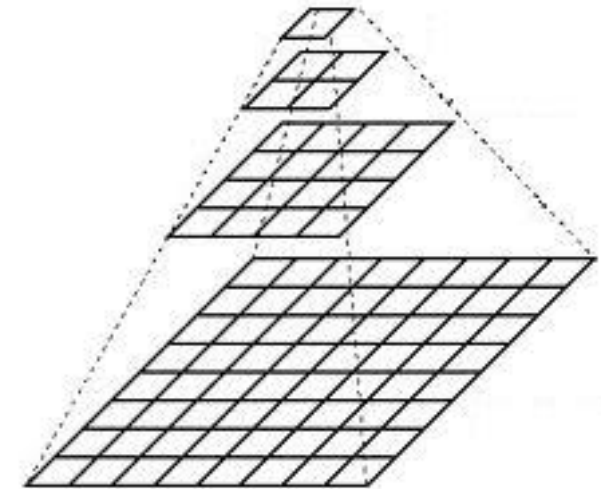
Применение пирамид изображений

- Улучшение сопоставления шаблонов
- Поиск сдвига
 - Классическая стратегия последовательного уточнения («coarse-to-fine strategy»)
- Поиск по масштабу (размеру)
 - Сравнение шаблонов
 - Поиск объектов с разным масштабом
 - Поиск лиц
 - Ключевых точек
- Основа «кратномасштабного анализа» изображений, вейвлет-анализа
- Multiresolution methods

Применение пирамид изображений

1. Свёртка изображения при помощи ядра-Гауссиана:

$$\frac{1}{16} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$



2. Удаление четных строк и столбцов

Пирамиды изображений

```
import cv2
```

```
image = cv2.imread('images/input.jpg')
```

```
smaller = cv2.pyrDown(image)
```

```
larger = cv2.pyrUp(smaller)
```

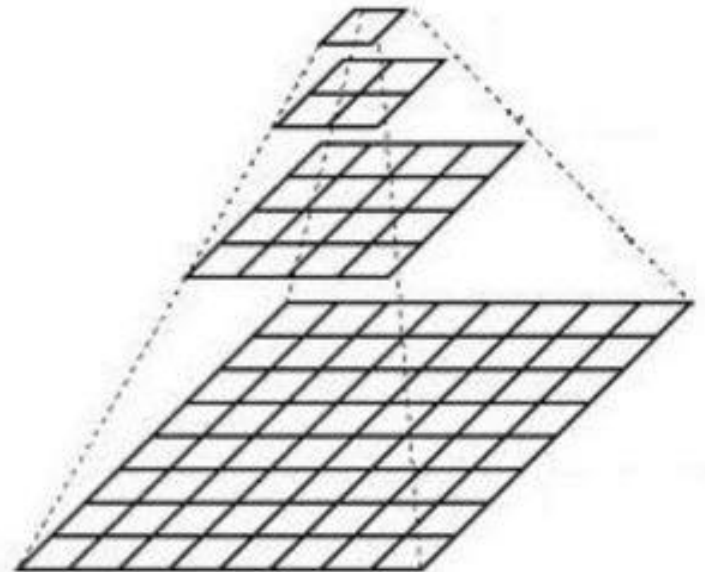
```
cv2.imshow('Original', image )
```

```
cv2.imshow('Smaller ', smaller )
```

```
cv2.imshow('Larger ', larger )
```

```
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```



Взвешенное сложение

Python: `cv.AddWeighted(src1, alpha, src2, beta, gamma, dst)` → None

Parameters:

`src1` – first input array.

`alpha` – weight of the first array elements.

`src2` – second input array of the same size and channel number as `src1`.

`beta` – weight of the second array elements.

`dst` – output array that has the same size and number of channels as the input arrays.

`gamma` – scalar added to each sum.

`dtype` – optional depth of the output array; when both input arrays have the same depth, `dtype` can be set to -1, which will be equivalent to `src1.depth()`.

The function `addWeighted` calculates the weighted sum of two arrays as follows:

$$\text{dst}(I) = \text{saturate}(\text{src1}(I) * \alpha + \text{src2}(I) * \beta + \gamma)$$

Взвешенное сложение

$$dst = \alpha \cdot img1 + \beta \cdot img2 + \gamma$$

```
import cv2
img1 = cv2.imread("road_add.jpg")
img2 = cv2.imread("car_add.jpg")
dst = cv2.addWeighted(img1,0.5,img2,0.7,0)
cv2.imshow('dst',dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Маскирование и побитовые операции

```
import cv2
import numpy as np
```

```
# If you're wondering why only two dimensions, well this is a grayscale image,
# if we doing a colored image, we'd use
# rectangle = np.zeros((300, 300, 3), np.uint8)
```

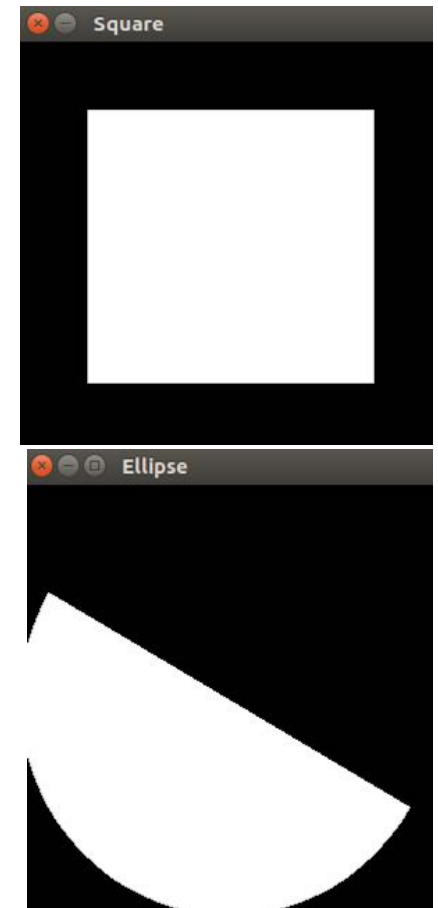
```
# Making a square
```

```
square = np.zeros((300, 300), np.uint8)
cv2.rectangle(square, (50, 50), (250, 250), 255, -2)
cv2.imshow("Square", square)
cv2.waitKey(0)
```

```
# Making an ellipse
```

```
ellipse = np.zeros((300, 300), np.uint8)
cv2.ellipse(ellipse, (150, 150), (150, 150), 30, 0, 180, 255, -1)
cv2.imshow("Ellipse", ellipse)
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```



Маскирование и побитовые операции

Shows only where they intersect

And = cv2.bitwise_and(square, ellipse)

cv2.imshow("AND", And)

cv2.waitKey(0)

Shows where either square or ellipse is

bitwiseOr = cv2.bitwise_or(square, ellipse)

cv2.imshow("OR", bitwiseOr)

cv2.waitKey(0)

Shows where either exist by itself

bitwiseXor = cv2.bitwise_xor(square, ellipse)

cv2.imshow("XOR", bitwiseXor)

cv2.waitKey(0)

Shows everything that isn't part of the square

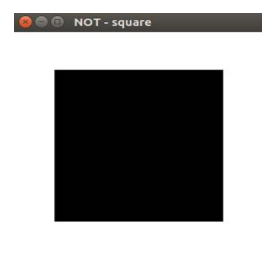
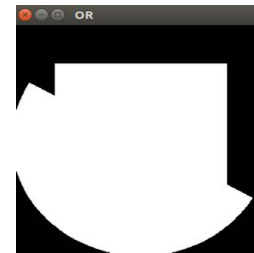
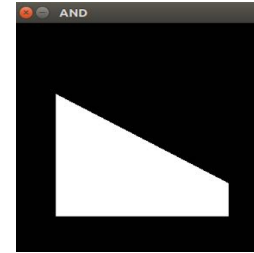
bitwiseNot_sq = cv2.bitwise_not(square)

cv2.imshow("NOT - square", bitwiseNot_sq)

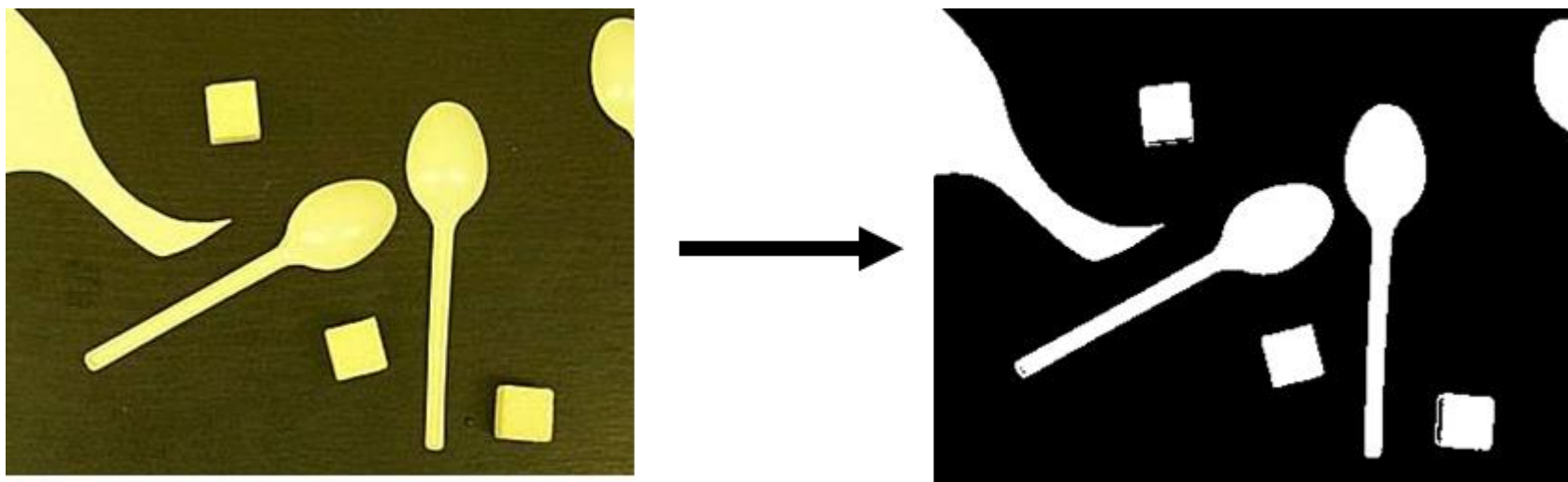
cv2.waitKey(0)

Notice the last operation inverts the image totally

cv2.destroyAllWindows()

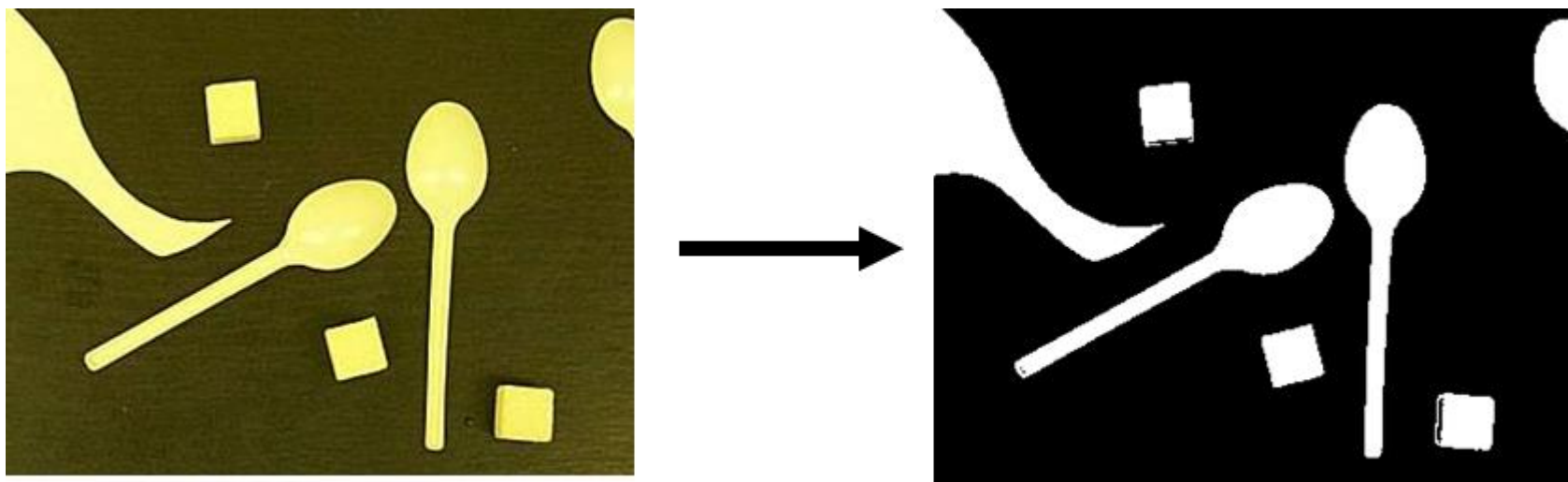


Бинаризация изображений



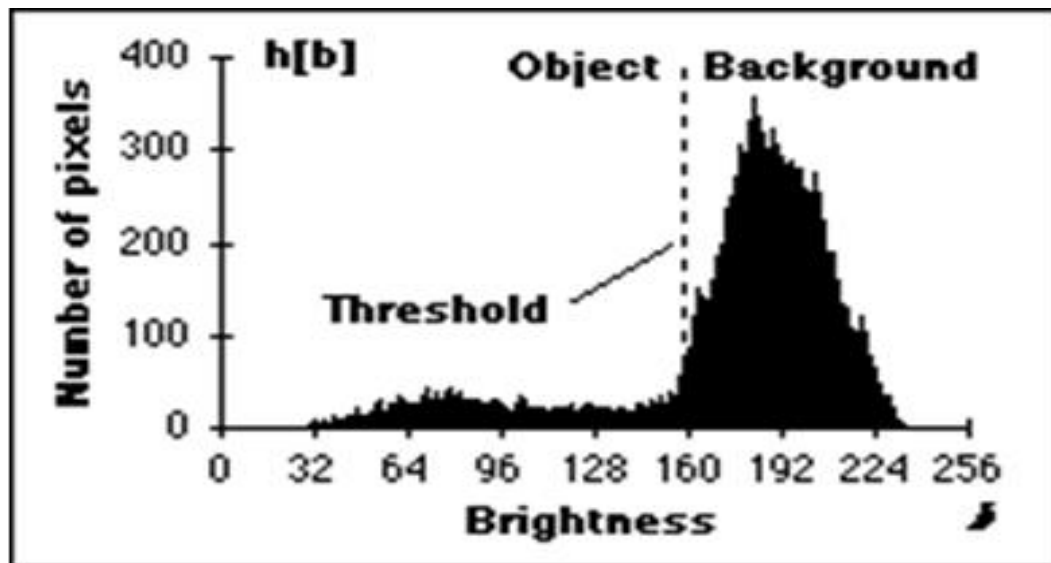
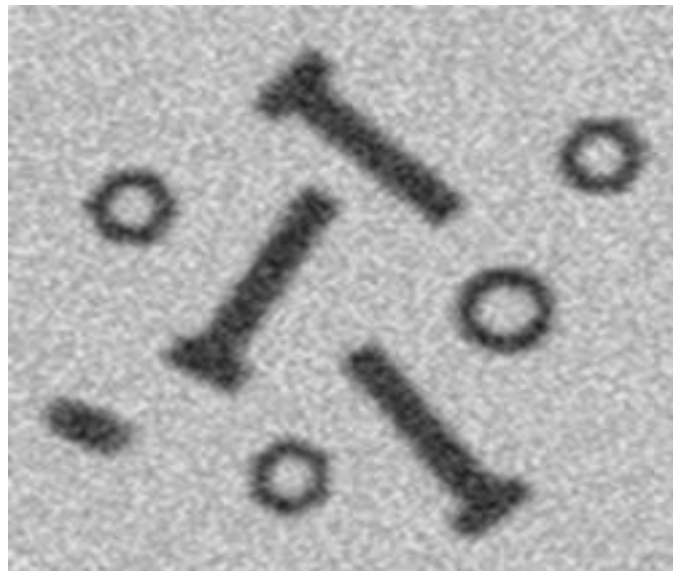
- Пиксель бинарного изображения может принимать только значения 0 и 1
- Бинаризация – построение бинарного изображения по полутоновому / цветному
- Разделение изображения на фон (0) и контрастные объекты (1)

Пороговая фильтрация



- Простейший вариант - пороговая фильтрация (thresholding)
- Выделение областей, яркость которых выше/ниже некоторого порога, заданного «извне»

Пороговая фильтрация

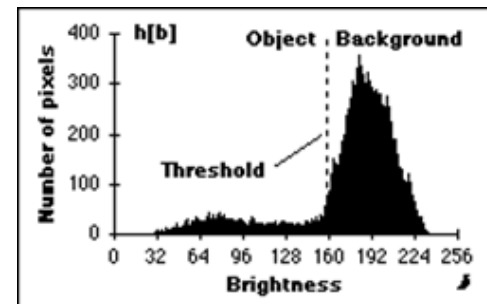


Определение порога автоматически, по характеристикам изображения

Анализ гистограммы

Пороговая фильтрация

- Анализ симметричного пика гистограммы
- Применяется когда фон изображения дает отчетливый и доминирующий пик гистограммы, симметричный относительно своего центра.



Алгоритм:

1. Сгладить гистограмму;
2. Найти ячейку гистограммы h_{\max} с максимальным значением;
3. На стороне гистограммы не относящейся к объекту (на примере – справа от пика фона) найти яркость h_p , количество пикселей с яркостью $\geq h_p$ равняется $p\%$ (например 5%) от пикселей яркости которых $\geq h_{\max}$;
4. Пересчитать порог $T = h_{\max} - (h_p - h_{\max})$;

Thresholding, Binarization & Adaptive Thresholding

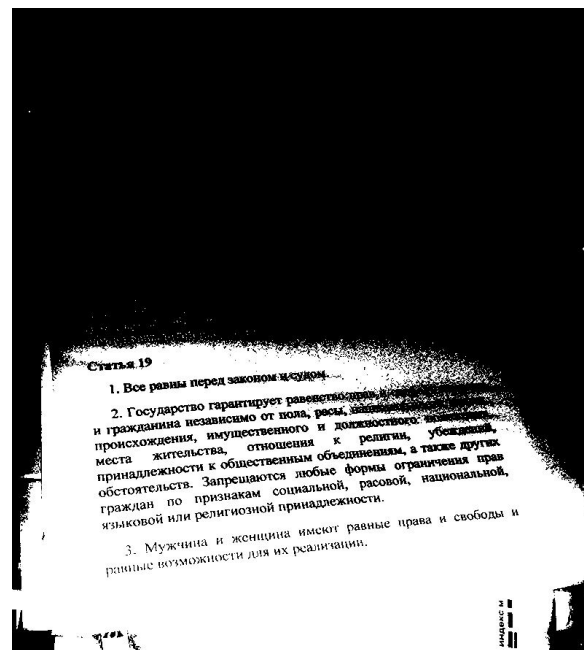
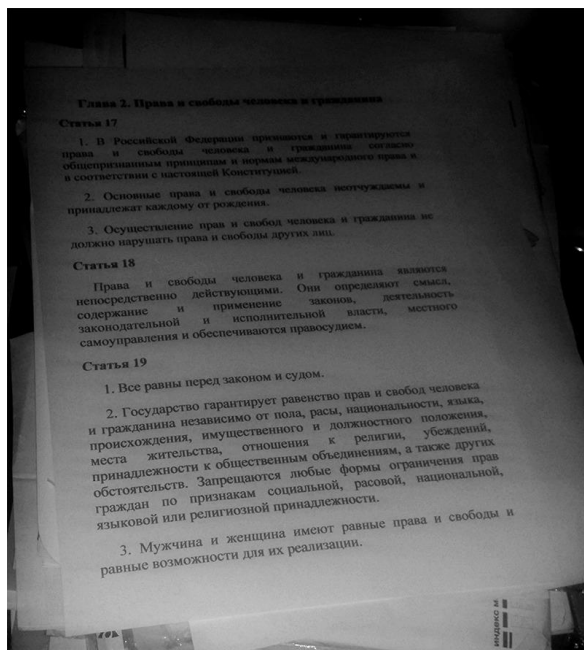
```
import cv2
import numpy as np

# Load our image as greyscale
image = cv2.imread('images/gradient.jpg',0)
cv2.imshow('Original', image)
# Values below 127 goes to 0 (black, everything above goes to 255 (white)
ret,thresh1 = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)
cv2.imshow('1 Threshold Binary', thresh1)
# Values below 127 go to 255 and values above 127 go to 0 (reverse of above)
ret,thresh2 = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY_INV)
cv2.imshow('2 Threshold Binary Inverse', thresh2)
# Values above 127 are truncated (held) at 127 (the 255 argument is unused)
ret,thresh3 = cv2.threshold(image, 127, 255, cv2.THRESH_TRUNC)
cv2.imshow('3 THRESH TRUNC', thresh3)
# Values below 127 go to 0, above 127 are unchanged
ret,thresh4 = cv2.threshold(image, 127, 255, cv2.THRESH_TOZERO)
cv2.imshow('4 THRESH TOZERO', thresh4)
# Reverse of above, below 127 is unchanged, above 127 goes to 0
ret,thresh5 = cv2.threshold(image, 127, 255, cv2.THRESH_TOZERO_INV)
cv2.imshow('5 THRESH TOZERO INV', thresh5)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

Адаптивная бинаризация

Необходима в случае неравномерной яркости фона/объекта.



Адаптивная бинаризация

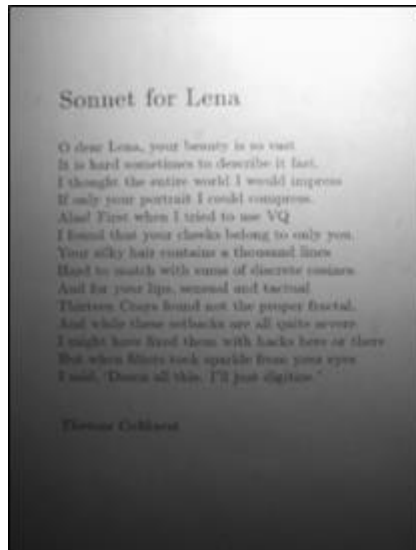
Необходима в случае неравномерной яркости фона/объекта.

1. Для каждого пикселя изображения $I(x, y)$:
 1. В окрестности пикселя радиуса r вычисляется индивидуальный для данного пикселя порог T ;
 2. Если $I(x, y) > T + C$, результат 1, иначе 0;

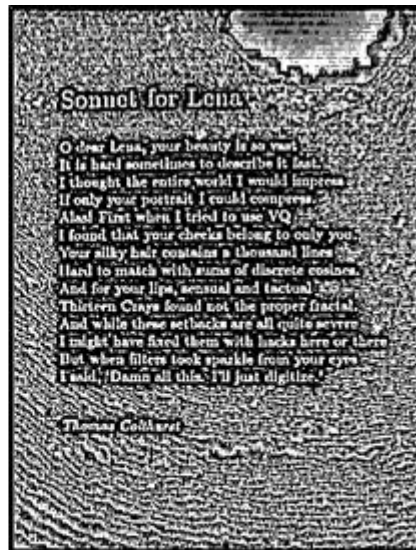
Варианты выбора T :

- $T = mean$
- $T = median$
- $T = (min + max) / 2$

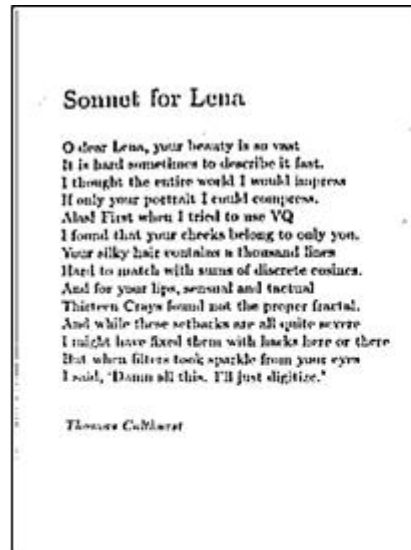
Адаптивная бинаризация



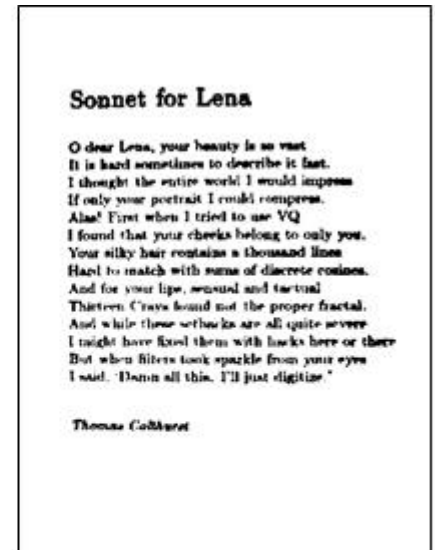
Исходное



$r=7, C=0$



$r=7, C=7$



$r=75, C=10$

Адаптивная бинаризация

Python: `cv2.adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C[, dst]) → dst`

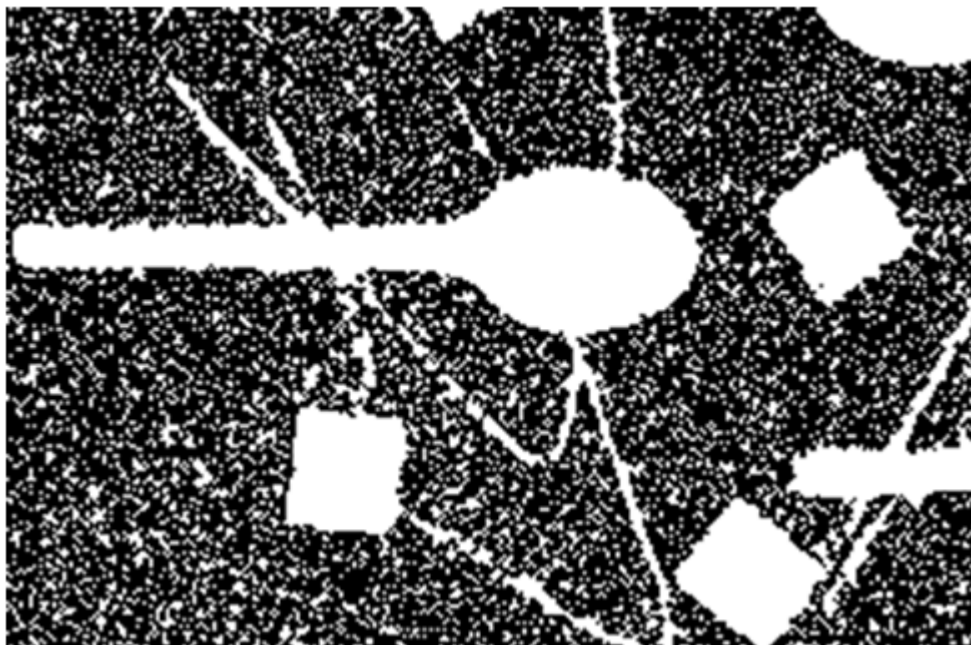
Parameters:

- **src** – Source 8-bit single-channel image.
- **maxValue** – Non-zero value assigned to the pixels for which the condition is satisfied. See the details below.
- **adaptiveMethod** – Adaptive thresholding algorithm to use, `ADAPTIVE_THRESH_MEAN_C` or `ADAPTIVE_THRESH_GAUSSIAN_C`. See the details below.
- **thresholdType** – Thresholding type that must be either `THRESH_BINARY` or `THRESH_BINARY_INV`.
- **blockSize** – Size of a pixel neighborhood that is used to calculate a threshold value for the pixel: 3, 5, 7, and so on.
- **C** – Constant subtracted from the mean or weighted mean (see the details below). Normally, it is positive but may be zero or negative as well.

Адаптивная бинаризация

```
import cv2
import numpy as np
image = cv2.imread('text.tiff', 0)
cv2.imshow('Original', image), cv2.waitKey(0)
# Values below 127 goes to 0 (black, everything above goes to 255
(white)
ret, thresh1 = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)
cv2.imshow('Threshold Binary', thresh1)
cv2.waitKey(0)
# It's good practice to blur images as it removes noise
image = cv2.GaussianBlur(image, (3, 3), 0)
# Using adaptiveThreshold
thresh = cv2.adaptiveThreshold(image, 255,
cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 81, 5)
cv2.imshow("Adaptive Mean Thresholding", thresh), cv2.waitKey(0)
ret, th2 = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
cv2.imshow("Otsu's Thresholding", th2), cv2.waitKey(0)
# Otsu's thresholding after Gaussian filtering blur =
cv2.GaussianBlur(image, (5,5), 0)
ret, th3 = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
cv2.imshow("Guassian Otsu's Thresholding", th3), cv2.waitKey(0)
```

Шум в бинарных изображениях

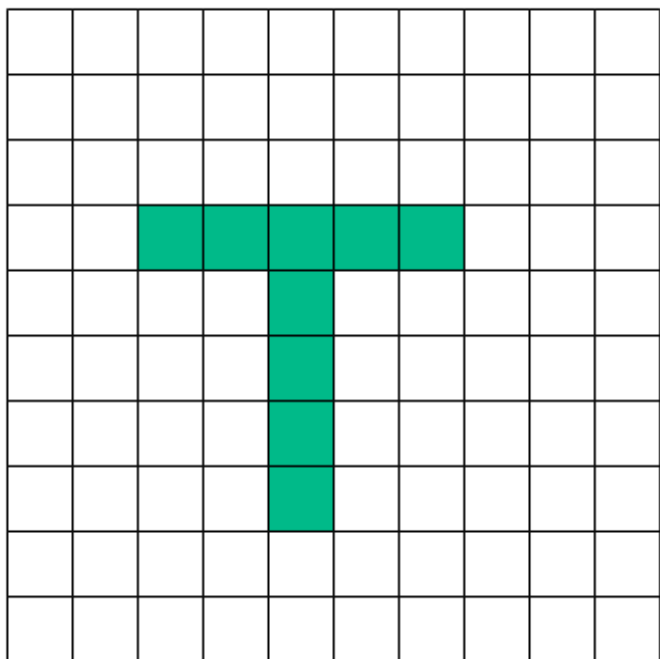


Часто возникает из-за невозможности полностью подавить шум в изображениях, недостаточной контрастности объектов и т.д.

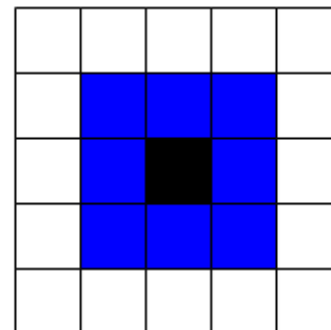
- По одному пикселу невозможно определить – шум или объект. Нужно рассматривать окрестность пиксела!

Математическая морфология

А

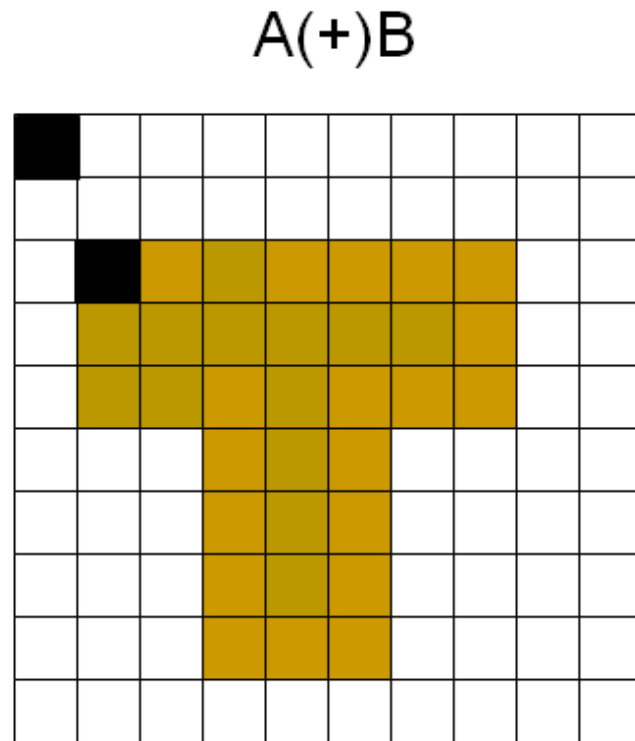
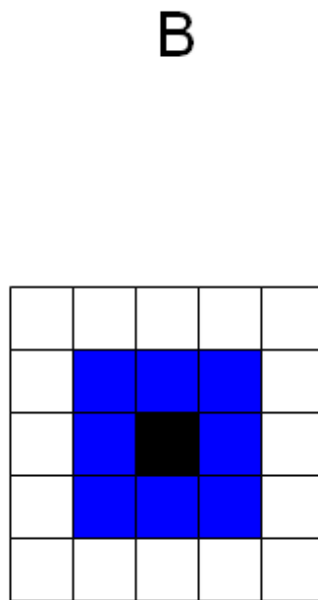
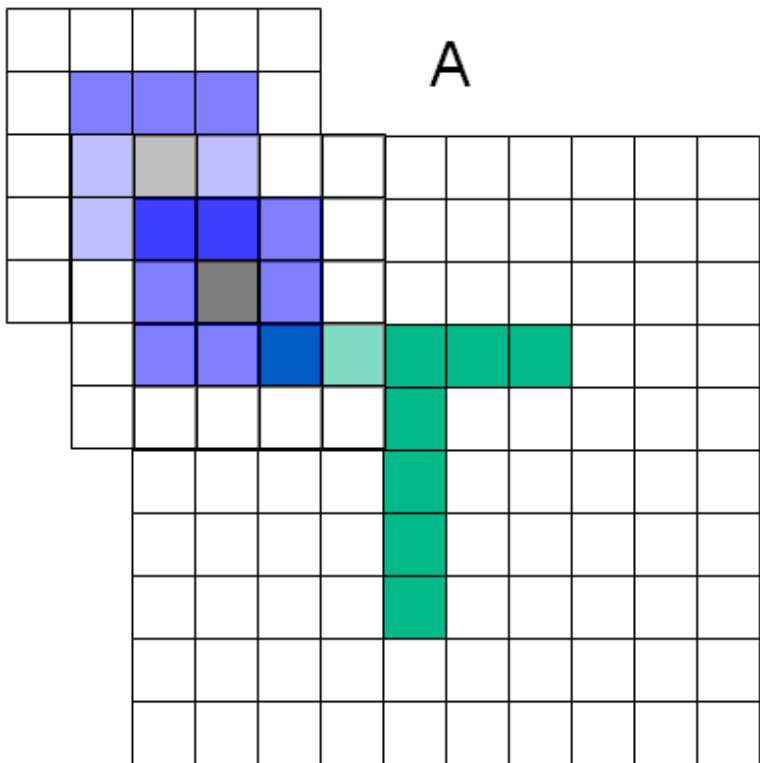


В



Множество А обычно является объектом обработки, а множество В (называемое структурным элементом) – инструментом.

Дилатация

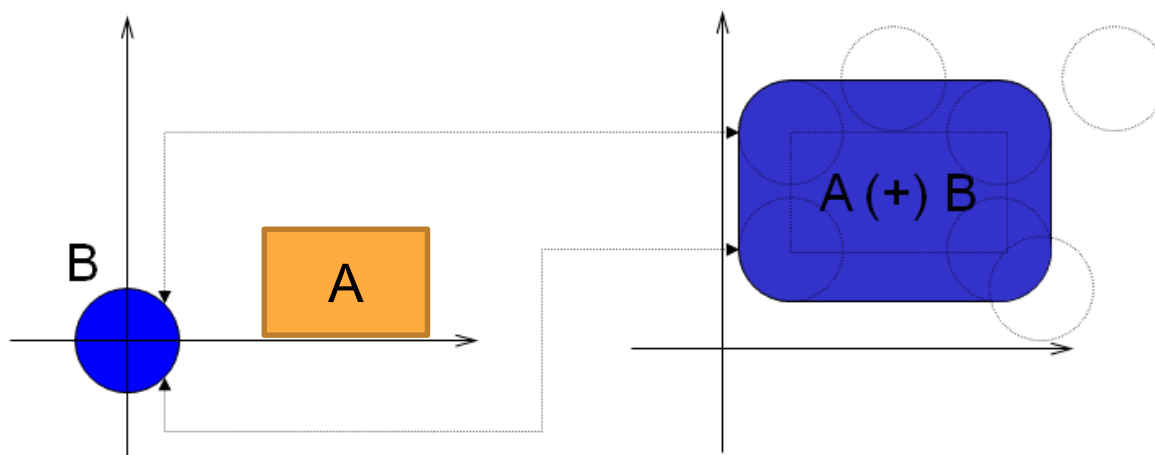


Операция «расширение» - аналог логического «или»

Дилатация

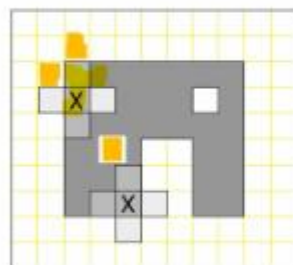
Расширение (dilation)

$$A (+) B = \{t \in \mathbb{R}^2: t = a + b, a \in A, b \in B\}$$



$$A \oplus B = B \oplus A$$

$$A \oplus (B \oplus C) = (A \oplus B) \oplus C$$

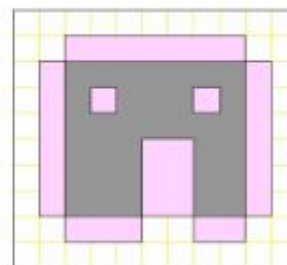


Original Image



Structuring
Element

X : origin



After Dilation

Дилатация

`cv2.dilation(src, kernel, dst, anchor, iterations, borderType, borderValue)`

Parameters:

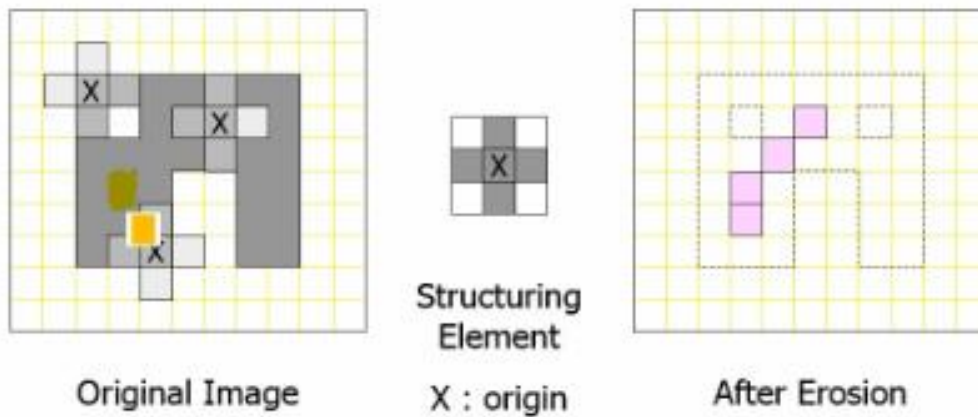
`src` – the depth should be one of `CV_8U`, `CV_16U`, `CV_16S`, `CV_32F` or `CV_64F`.

`kernel` – structuring element. `cv2.getStructuringElement()` .

`anchor` – structuring element. default `(-1,-1)`.

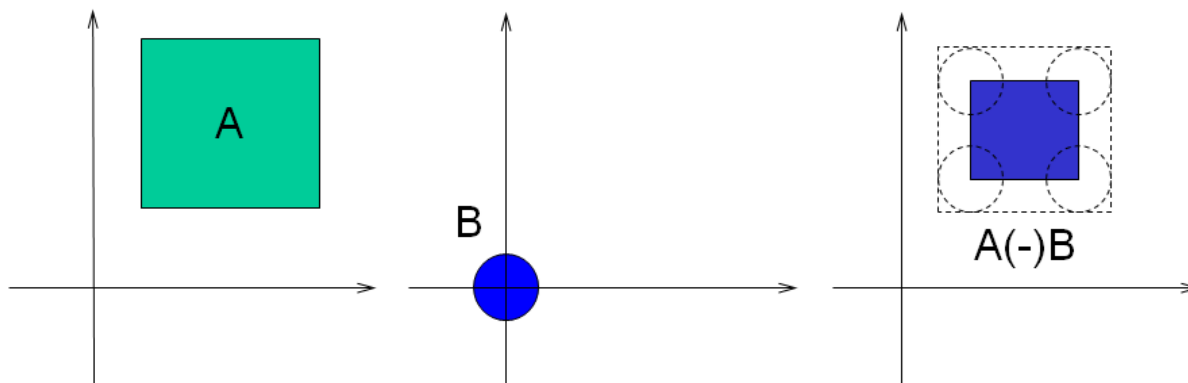
`iterations` – dilation

Сужение



Сужение (erosion)

$A (-) B = (A^C (+) B)^C$, где A^C – дополнение A



Сужение

`cv2.erode(src, kernel, dst, anchor, iterations, borderType, borderValue)`

Parameters:

`src` – the depth should be one of `CV_8U`, `CV_16U`, `CV_16S`, `CV_32F` or `CV_64F`.

`kernel` – structuring element. `cv2.getStructuringElement()`.

`anchor` – structuring element. default `(-1,-1)`.

`iterations` – erosion

Открытие и закрытие



Morphological opening operation



Morphological closing operation

Открытие и закрытие

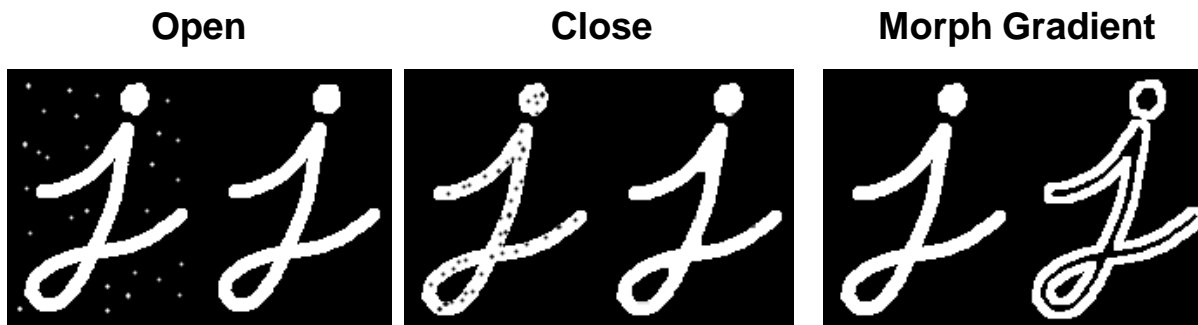
Где применить закрытие и открытие?



Открытие и закрытие

`cv2.morphologyEx(src, op, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]) → dst`
Parameters:

- **src** – Source image. The number of channels can be arbitrary. The depth should be one of `CV_8U`, `CV_16U`, `CV_16S`, `CV_32F` or `CV_64F`.
- **op** – Type of a morphological operation that can be one of the following:
 - **MORPH_OPEN** - an opening operation
 - **MORPH_CLOSE** - a closing operation
 - **MORPH_GRADIENT** - a morphological gradient. Dilation and erosion
 - **MORPH_TOPHAT** - “top hat”. Opening. It is the difference between input image and Opening of the image.
 - **MORPH_BLACKHAT** - “black hat”. Closing. It is the difference between the closing of the input image and input image.
- **kernel** – structuring element. `cv2.getStructuringElement()`.
- **anchor** – structuring element origin. default (-1,-1).
- **iterations** – erosion and dilation
- **borderType** – Pixel extrapolation method. See `borderInterpolate` for details.
- **borderValue** – Border value in case of a constant border. The default value has a special meaning.



Открытие и закрытие

`cv2.getStructuringElement(shape, ksize[, anchor])` →
retval

Parameters:

- **shape** –
 - Element
 - **MORPH_RECT** :
[[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1]]
 - **MORPH_ELLIPSE** :
[[0, 0, 1, 0, 0],
[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1],
[0, 0, 1, 0, 0]]
 - **MORPH_CROSS** :
[[0, 0, 1, 0, 0],
[0, 0, 1, 0, 0],
[1, 1, 1, 1, 1],
[0, 0, 1, 0, 0],
[0, 0, 1, 0, 0]]
- **ksize** – structuring element

`kernel =`

`cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))`

kernel =

cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))

kernel =

cv2.getStructuringElement(cv2.MORPH_CROSS, (5, 5))