ЛР2: Фильтр гауссова размытия

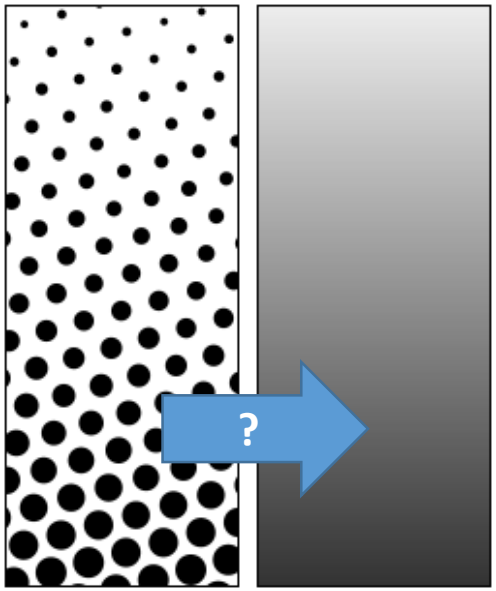
Алгоритмы цифровой обработки изображений

Фильтр

- Устройство или алгоритм обработки сигналов, выделяющий желательные и подавляющий нежелательные сигналы
- ФВЧ, ФНЧ, …

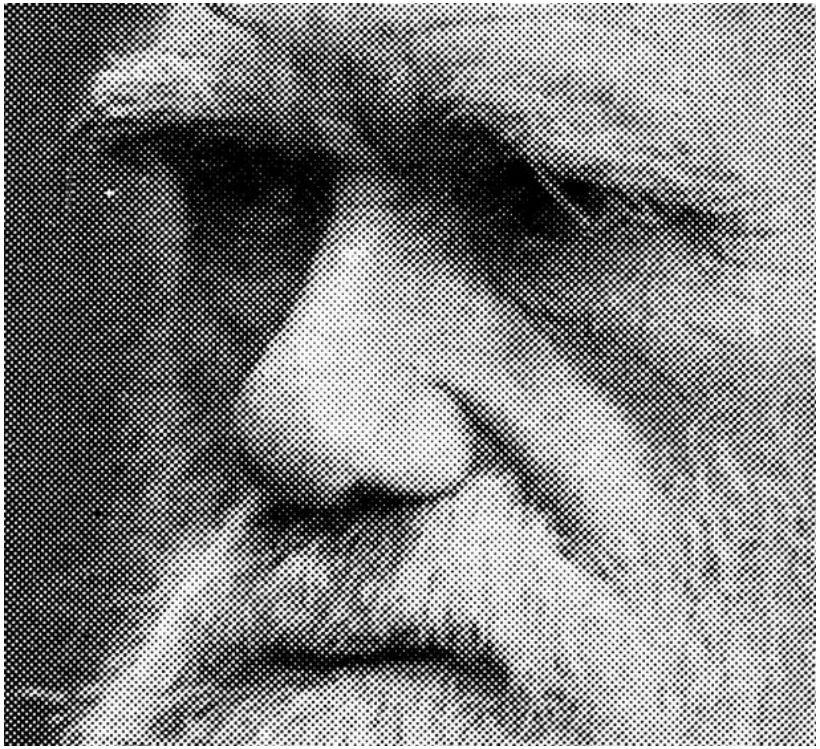
Пример: обращение полутонового преобразования

- В монохромной печати нельзя плавно варьировать краску
- Серый не получается разбавлением черного
- Поэтому растеризуется точками одного цвета, но разного размера
- Обратное преобразование?
- Размытие



Два файла-образца

- "Полутоновые" изображения darwin, ambcorps
- Источник: Wikipedia, Wellcome Trust



https://commons.wikimedia.org/wiki/File:Portrait_of_Charles_Darwin._Wellcome_M0010103.jpg
https://commons.wikimedia.org/wiki/File:An_ambulance_corps_at_work_in_the_field._Halftone._Wellcome_V0015759.jpg

Размытие

- Усреднение?
- Размер в пикселях r
- Заменяем каждый пиксель на среднее по квадрату $\pm r$
- Позаботиться о краях

pillow

- пакет обработки изображений, форк PIL
- позволяет
  - читать различные форматы
  - записывать PNG
  - получать пиксели в виде числового массива
  - строить изображение из числового массива

Установка pillow

```
pip install pillow
```

зависит от olefile

если под Windows есть ошибки, проще скачать и установить предкомпилированные колеса (пакеты wheels) с https://www.lfd.uci.edu/~gohlke/pythonlibs/

1. колесо для olefile: https://download.lfd.uci.edu/pythonlibs/zhckc95n/olefile-0.44-py2.py3-none-any.whl

pillow

2. колесо для pillow:
   надо выбрать свою версию и разрядность Питона:
   - Pillow-3.4.2-cp36-cp36m-win32.whl
   - Pillow-3.4.2-cp36-cp36m-win_amd64.whl
   - Pillow-4.3.0-cp27-cp27m-win32.whl
   - Pillow-4.3.0-cp27-cp27m-win_amd64.whl
   - Pillow-4.3.0-cp34-cp34m-win32.whl
   - Pillow-4.3.0-cp34-cp34m-win_amd64.whl
   - Pillow-4.3.0-cp35-cp35m-win32.whl
   - Pillow-4.3.0-cp35-cp35m-win_amd64.whl
   - Pillow-4.3.0-cp36-cp36m-win32.whl
   - Pillow-4.3.0-cp36-cp36m-win_amd64.whl
   - Pillow-4.3.0-pp258-pypy_41-win32.whl

numpy+MKL

3. колесо для numpy (>100MB):
   надо выбрать свою версию и разрядность Питона:
   - numpy-1.13.3+mkl-cp27-cp27m-win32.whl
   - numpy-1.13.3+mkl-cp27-cp27m-win_amd64.whl
   - numpy-1.13.3+mkl-cp34-cp34m-win32.whl
   - numpy-1.13.3+mkl-cp34-cp34m-win_amd64.whl
   - numpy-1.13.3+mkl-cp35-cp35m-win32.whl
   - numpy-1.13.3+mkl-cp35-cp35m-win_amd64.whl
   - numpy-1.13.3+mkl-cp36-cp36m-win32.whl
   - numpy-1.13.3+mkl-cp36-cp36m-win_amd64.whl

Установка колес Windows

- Поместите .whl файлы в папку Scripts вашей инсталляции Питона, где лежит pip.exe
- Откройте командную строку, перейдите в папку Scripts, установите колеса при помощи pip install

```
C:\imgproc\lab2\Scripts>
C:\imgproc\lab2\Scripts>pip install numpy-1.13.3+mkl-cp35-cp35m-win32.whl
Processing c:\imgproc\lab2\scripts\numpy-1.13.3+mkl-cp35-cp35m-win32.whl
Installing collected packages: numpy
Successfully installed numpy-1.13.3+mkl

C:\imgproc\lab2\Scripts>pip install olefile-0.44-py2.py3-none-any.whl
Processing c:\imgproc\lab2\scripts\olefile-0.44-py2.py3-none-any.whl
Installing collected packages: olefile
Successfully installed olefile-0.44

C:\imgproc\lab2\Scripts>pip install Pillow-4.3.0-cp35-cp35m-win32.whl
Processing c:\imgproc\lab2\scripts\pillow-4.3.0-cp35-cp35m-win32.whl
Requirement already satisfied: olefile in c:\imgproc\lab2\lib\site-packages (from Pillow==4.3.0)
Installing collected packages: Pillow
Successfully installed Pillow-4.3.0
```

Как читать файлы изображений

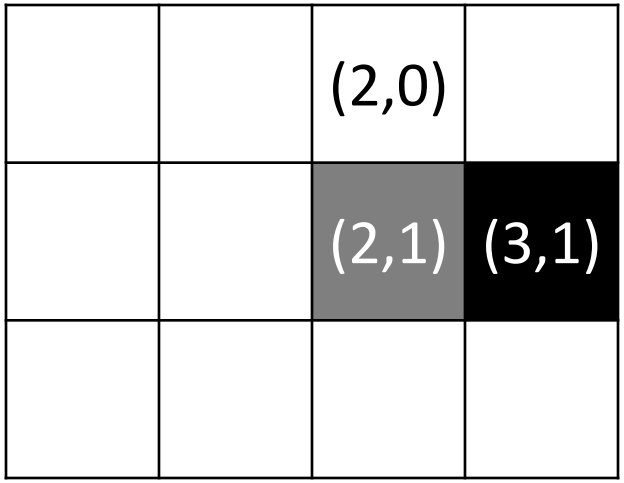
```
from PIL import Image
img = Image.open('darwin.png')
img.load()
```

Тогда `img.size` дает пару **(*W* ширина, *H* высота)**.
Режим изображения `img.mode`. `'L'` означает градации серого
0..255 (0 – черный, 255 – белый)

`img.getpixel((x,y))` дает значение пикселя с координатами x, y.
передавать именно как пару, поэтому в скобках,
x слева направо, y сверху вниз, считая с нуля

Пиксельные данные

```
>>> img.getpixel((2,1))
128
>>> img.getpixel((3,1))
0
>>> img.getpixel((0,2))
255
```

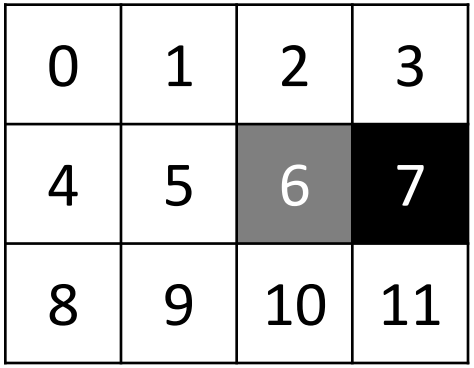


(псевдо)Массив getdata

- `img.getdata()`
- линейный массив от 0 до $W \times H - 1$

```
>>> img.getdata()[2]
255
>>> img.getdata()[6]
128
>>> img.getdata()[7]
0
```

```
<PIL.Image.Image
image mode=L size=4x3 at 0x47E8C50>
```

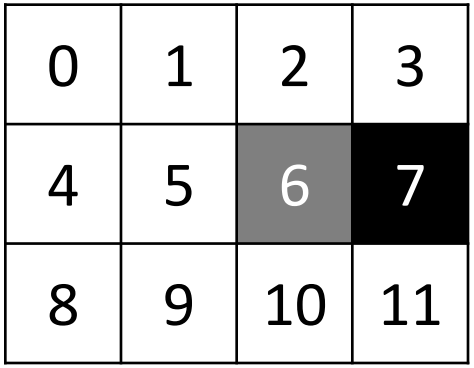


Преобразовать в numpy-массив

```
import numpy as np
a = np.array(img.getdata())
```

numpy-массивы `ndarray` – высокопроизводительные компактные массивы, поддерживающие удобную индексацию, диапазоны, множество операций

```
>>> a
array([255,255,255,255, 255,255,128,  0,
255,255,255,255])

>>> a.reshape(3, 4)
array([[255,255,255,255,
        255,255,128,  0,
        255,255,255,255])
```



Преобразовать в numpy-массив

```
>>> import numpy as np
>>> a = np.array(img.getdata())
>>> a
array([255,255,255,255,
       255,255,128,  0,
       255,255,255,255])
>>> a.reshape(3, 4)
array([[255,255,255,255],
       [255,255,128,  0],
       [255,255,255,255]])
>>> b = a.reshape(3, 4)
>>> b[1,2]
128
>>> b[1][2]
128
```

0	1	2	3
4	5	6	7
8	9	10	11

… и обратно

- имея двумерный numpy-массив, можно построить по нему объект Image, который уже можно сохранять, и т.д.
- ```
  b = np.array([[1,2,10],[20,40,255]],
                dtype=np.uint8)
  ```
- ```
  img = Image.fromarray(b)
  ```

Просматривать и сохранять

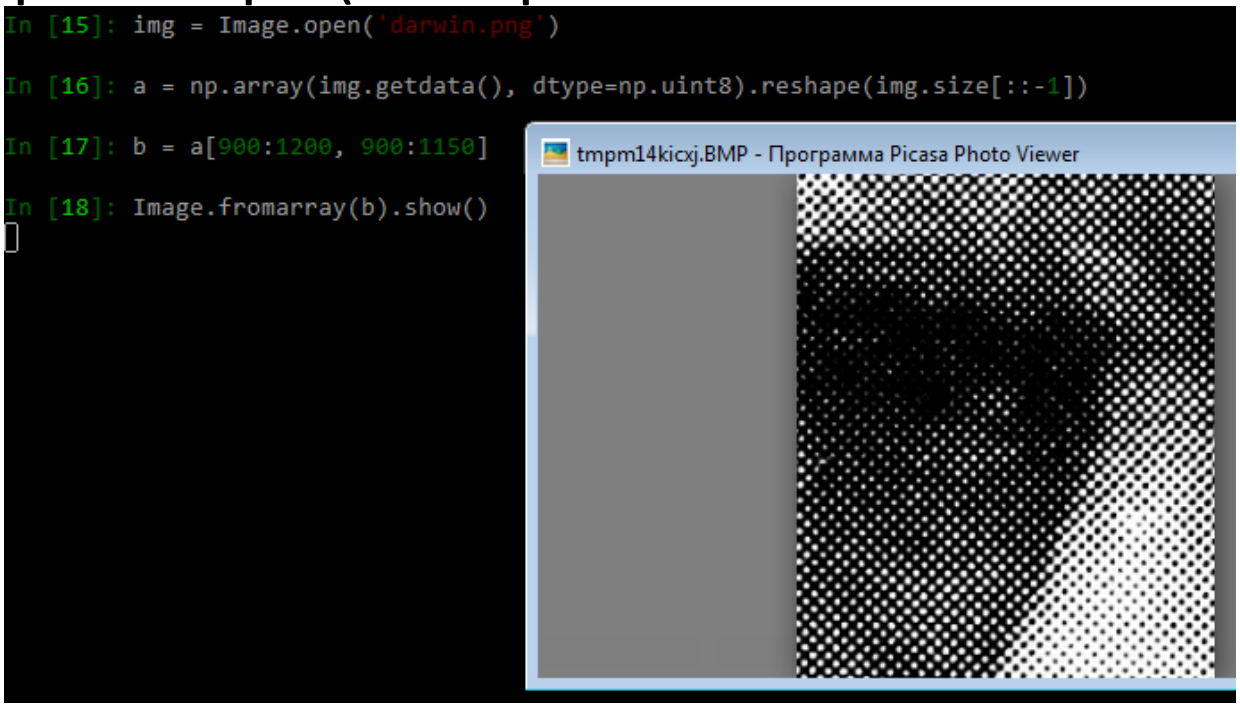
- В Jupyter Notebook, если изображение типа Image является последней строкой ячейки, оно отрисовывается автоматически прямо на странице

```
In [7]: b = a.reshape(img.size[::-1])[900:1200, 900:1150]
        Image.fromarray(b)
```

Out[7]:

Просматривать и сохранять

- В консоли можно вызывать метод .show() объекта Image для показа в стандартном средстве просмотра (на картинке Picasa Photo Viewer):



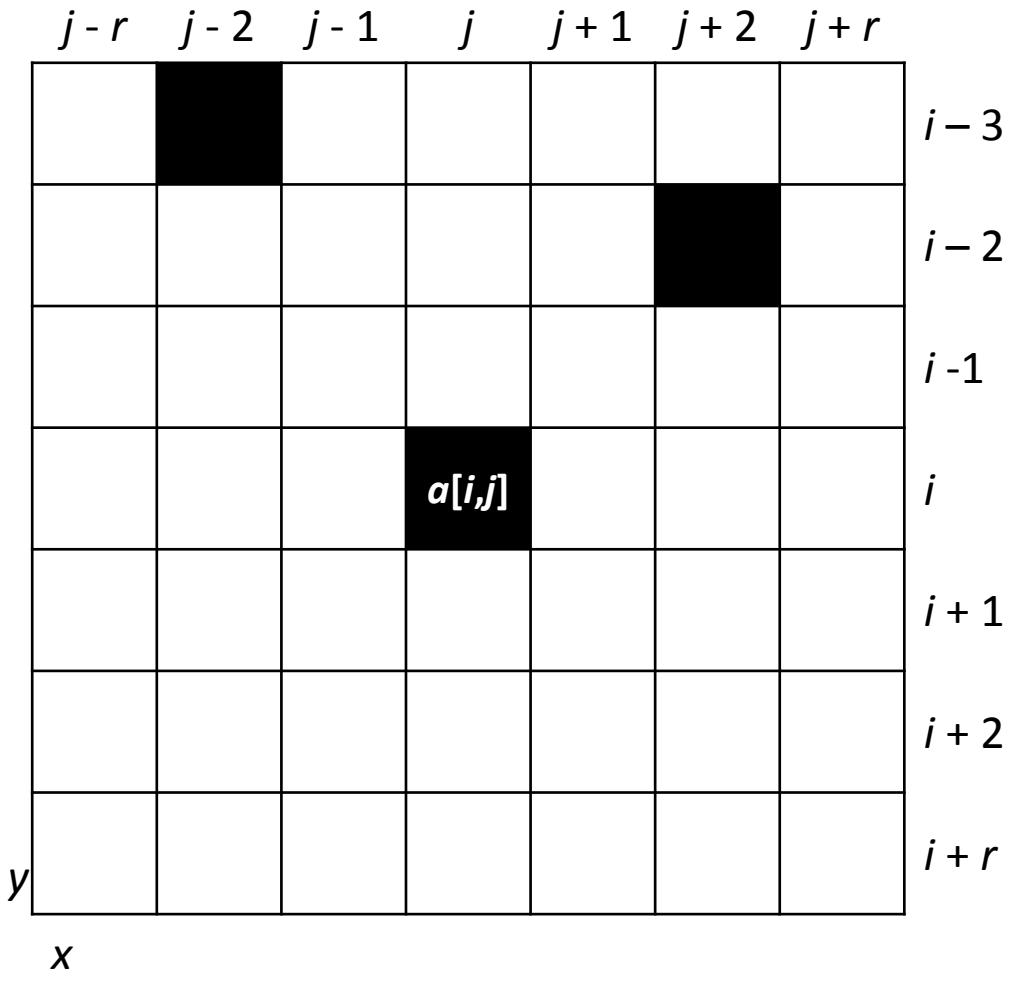
Сохранять

```
In [19]: newpic = Image.fromarray(b)
In [20]: newpic.mode = 'L'
In [21]: newpic.save('newpic.png')
```

Ломти(slice)

```
In [1]: import numpy as np

In [2]: x = np.array([[3,5,7],[4,6,8]])

In [3]: x[0]
Out[3]: array([3, 5, 7])

In [4]: x[1]
Out[4]: array([4, 6, 8])

In [5]: x[0,1]
Out[5]: 5

In [6]: x[1,0]
Out[6]: 4

In [7]: x[-1,-2]
Out[7]: 6

In [8]: x[0:1]
Out[8]: array([[3, 5, 7]])

In [9]: x[0:2]
Out[9]:
array([[3, 5, 7],
       [4, 6, 8]])
```

Простое равномерное усреднение



$$a_{ij} \mapsto \frac{\sum_{p=i-r}^{i+r} \sum_{q=j-r}^{j+r} a_{pq}}{(2r+1)^2}$$

На картинке $r = 3$, сумма включает три нуля и остальные 46 значений равны 255

сред = (0+46*255)/49 ≈ 239

Что происходит на краях, например i=1? Так как строки -2 не существует, среднее содержит меньше слагаемых

Равномерно усредняющее размытие, Питон

```
from PIL import Image
import numpy as np
def blur_py(img, r):
    w, h = img.size
    a = np.array(img.getdata(), dtype=np.uint8)\
                .reshape(h, w)
    b = np.zeros((h,w), dtype=np.uint8)
    for i in range(h):
        for j in range(w):
            s = 0.
            up, bt = max(i-r,0), min(i+r+1,h)
            lf, rt = max(j-r,0), min(j+r+1,w)
            n = (bt-up)*(rt-lf)
            for y in range(up,bt):
                for x in range(lf,rt):
                    s += a[y,x]
            b[i,j] = s / n
    return Image.fromarray(b)
```

Питоновский **for** = **МЕДЛЕННО**,
циклы надо прятать в numpy-вызовы

Равномерно усредняющее размытие, среднее в numpy

```
from PIL import Image
import numpy as np

def blur_np(img, r):
    w, h = img.size
    a = np.array(img.getdata(), dtype=np.uint8)\
                .reshape(h, w)
    b = np.zeros(a.shape, dtype=np.uint8)
    for i in range(h):
        for j in range(w):
            up, bt = max(i-r,0), min(i+r+1,h)
            lf, rt = max(j-r,0), min(j+r+1,w)
            b[i,j] = np.average(a[up:bt,lf:rt])
    return Image.fromarray(b)
```

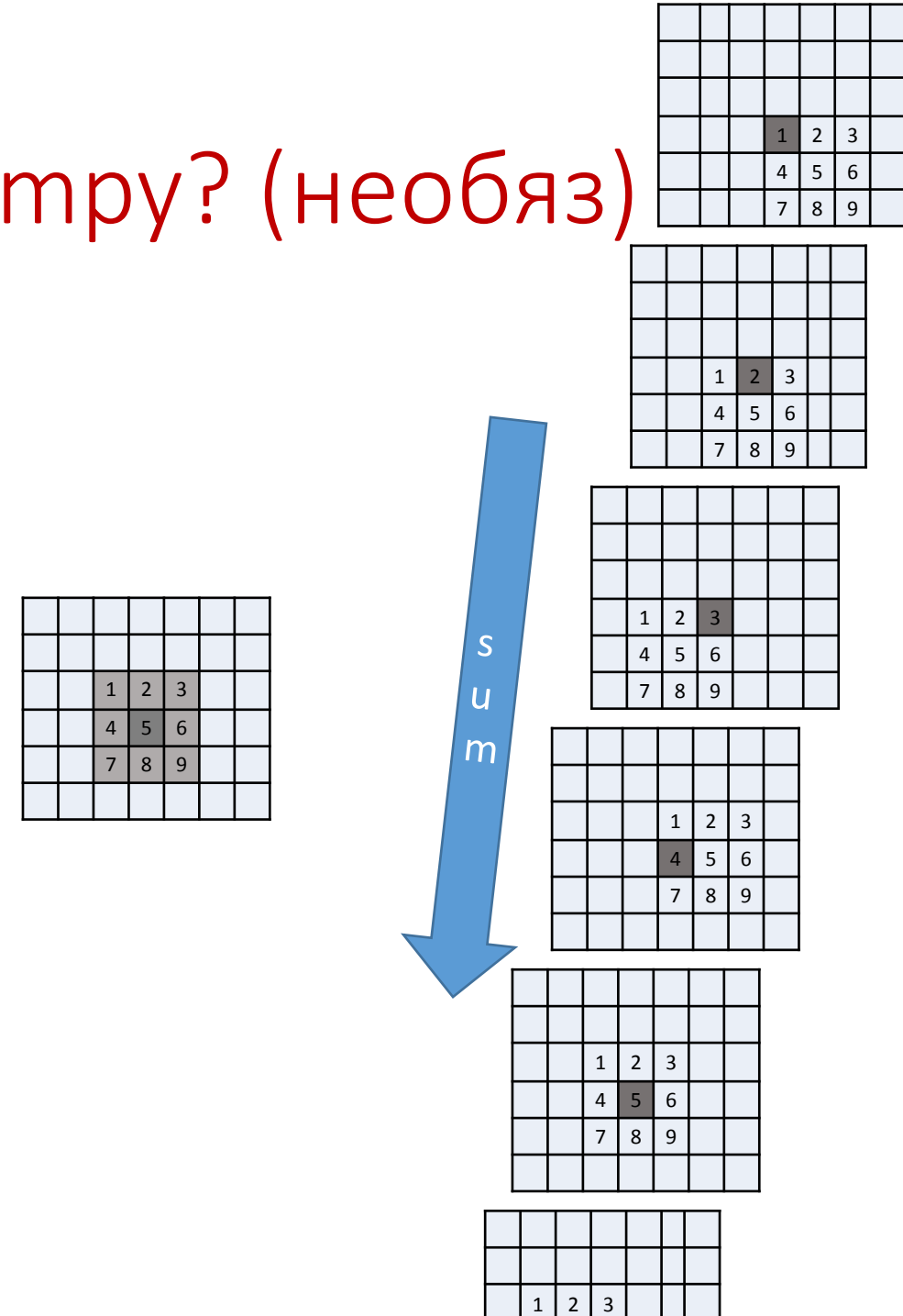
Скорость

```
In [90]: pic.size
Out[90]: (400, 400)

In [91]: %timeit newpic = blur_py(pic, 6)
1 loop, best of 3: 12.7 s per loop

In [92]: %timeit newpic = blur_np(pic, 6)
1 loop, best of 3: 2.92 s per loop
```

- 4x ускорение
- Но 3 с на 400x400 – все равно медленно
- Внешние *for*'ы остались в Питоне

Полностью в numpy? (необяз)

- Наплевав на корректность на краях, можно представлять суммы по квадратику как суммирование по вертикали стопки, в которой каждый элемент получен сдвигом в пределах квадратика {–*r* ,+r}



Циклический сдвиг (roll)

```
In [10]: x[-1]
Out[10]: array([4, 6, 8])

In [11]: x[0:2, 1:3]
Out[11]:
array([[5, 7],
       [6, 8]])

In [12]: x
Out[12]:
array([[3, 5, 7],
       [4, 6, 8]])

In [13]: np.roll(x,1,axis=1)
Out[13]:
array([[7, 3, 5],
       [8, 4, 6]])

In [14]: np.roll(x,1,axis=0)
Out[14]:
array([[4, 6, 8],
       [3, 5, 7]])

In [15]: np.roll(x,2,axis=1)
Out[15]:
array([[5, 7, 3],
       [6, 8, 4]])
```

Окаймление (padding)

```
In [21]: x = np.array([[1,2,3],
[4,5,6],[7,8,9]])

In [22]: np.pad(x,1,'constant',
constant_values=0)
array([[0, 0, 0, 0, 0],
       [0, 1, 2, 3, 0],
       [0, 4, 5, 6, 0],
       [0, 7, 8, 9, 0],
       [0, 0, 0, 0, 0]])

In [23]: np.pad(x,(1,0),'constant',
constant_values=0)
array([[0, 0, 0, 0],
       [0, 1, 2, 3],
       [0, 4, 5, 6],
       [0, 7, 8, 9]])

In [24]: np.pad(x,((1,0),(0,0)),
'constant',constant_values=0)
array([[0, 0, 0],
       [1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
In [25]:
np.pad(x,((1,0),(0,1)),'constant',
constant_values=0)
array([[0, 0, 0, 0],
       [1, 2, 3, 0],
       [4, 5, 6, 0],
       [7, 8, 9, 0]])

In [26]:
np.pad(x,((1,0),(0,0)),'edge')
array([[1, 2, 3],
       [1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

In [27]:
np.pad(x,((0,0),(0,1)),'edge')
array([[1, 2, 3, 3],
       [4, 5, 6, 6],
       [7, 8, 9, 9]])
```

Равномерно усредняющее размытие, полностью numpy

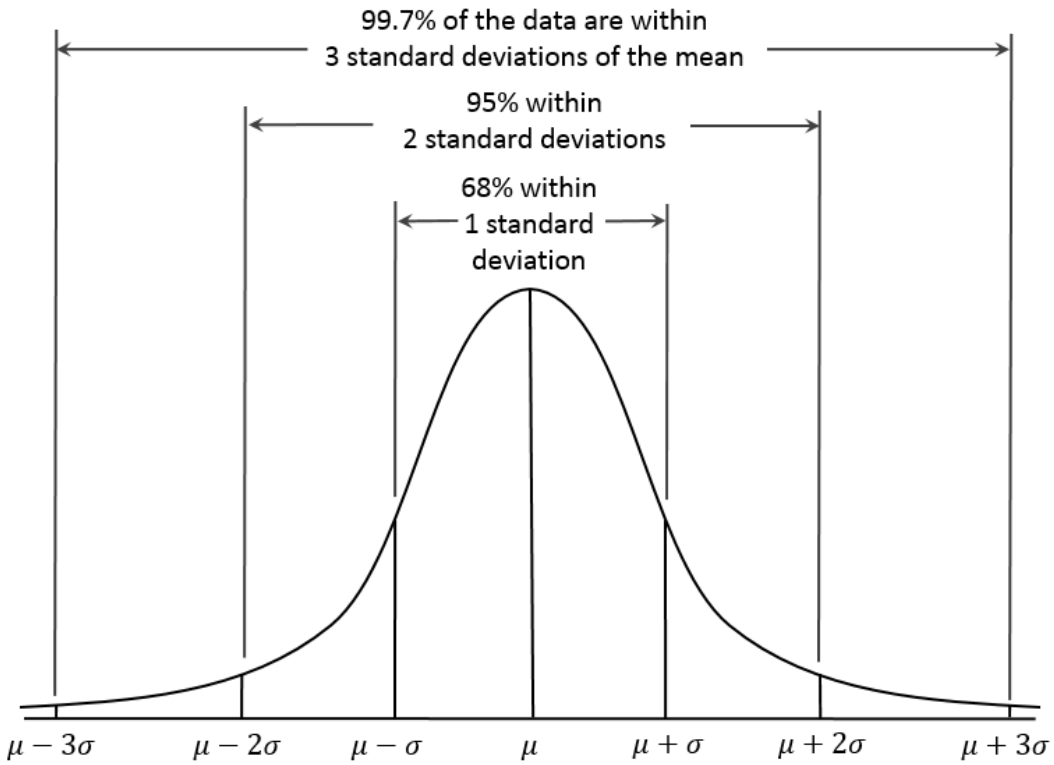
```
In [195]: def blur_npnp(img,r):
     ...:     w, h = img.size
     ...:     a = np.array(img.getdata(), dtype=np.uint8)\
     ...:                 .reshape(h, w)
     ...:     n = 2*r+1
     ...:     shifted = [np.roll(np.roll(a,i//n,axis=0),i%n,axis=1) for i in range(n**2)]
     ...:     b = (np.sum(np.stack(shifted),axis=0)/n**2).astype(np.uint8)
     ...:     return Image.fromarray(b)
     ...:

In [196]: pic.size
Out[196]: (400, 400)

In [197]: %timeit newpic = blur_np(pic, 6)
1 loop, best of 3: 2.97 s per loop

In [198]: %timeit newpic = blur_npnp(pic, 6)
1 loop, best of 3: 173 ms per loop
```

- В 17 раз быстрее, но на краях неправильно

py+np vs np+np

Гауссово размытие

- Усредняет пиксели изображения при помощи нормального распределения

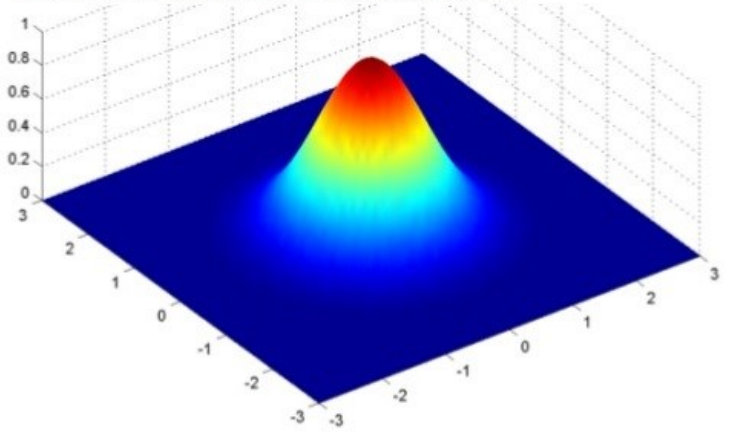


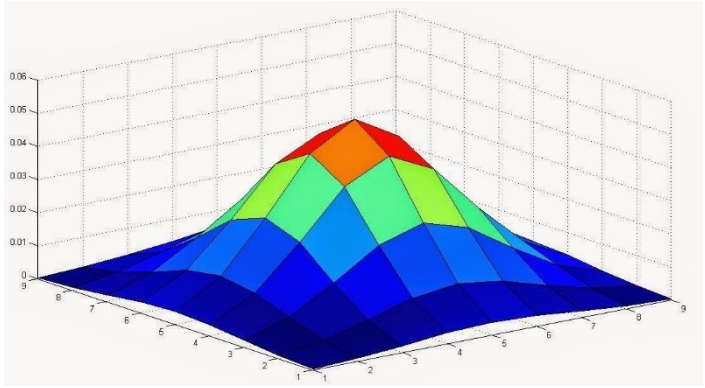
Дискретное гауссово усреднение?

- Значения берутся только до ±*r*, где сигма берется как 0.38*r*, дальше считаются нулями

$$f(x,y) = A\exp\left(-\left(\frac{(x-x_o)^2}{2\sigma_x^2} + \frac{(y-y_o)^2}{2\sigma_y^2}\right)\right).$$

Here the coefficient A is the amplitude, x_0,y_0 is the center and σ_x, σ_y are the x and y spreads of the blob.
The figure on the right was created using $A = 1$, $x_0 = 0$, $y_0 = 0$, $\sigma_x = \sigma_y = 1$.





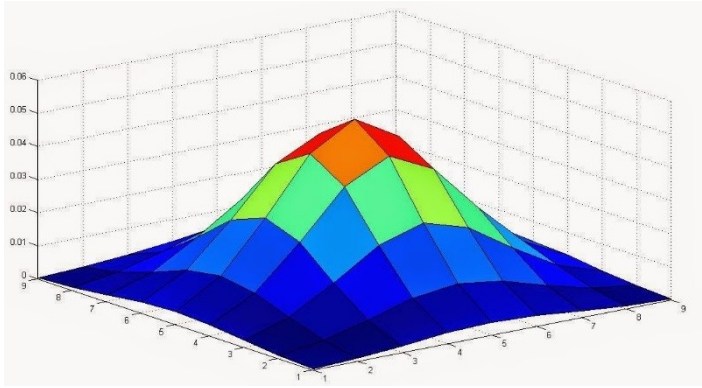
- Нормируются так, что сумма в квадрате ±*r* равна 1

Дискретное гауссово усреднение

Пример для r=3,
$\sigma = 0.38*3 = 1.14$

0,00012	0,00083	0,00262	0,00385	0,00262	0,00083	0,00012
0,00083	0,00566	0,01794	0,02636	0,01794	0,00566	0,00083
0,00262	0,01794	0,05691	0,08361	0,05691	0,01794	0,00262
0,00385	0,02636	0,08361	0,12284	0,08361	0,02636	0,00385
0,00262	0,01794	0,05691	0,08361	0,05691	0,01794	0,00262
0,00083	0,00566	0,01794	0,02636	0,01794	0,00566	0,00083
0,00012	0,00083	0,00262	0,00385	0,00262	0,00083	0,00012

- Сумма = 1
- Симметрично
- Максимальное (в центре) больше минимального (по углам) в ≈1000 раз
- Эти числа – множители, весá, на которые умножаем пиксели в квадрате и складываем = каждый пиксель заменяем на взвеш. среднее квадратика с центром в нем



Цель работы

- Создать программу, которая зачитывает изображение и сохраняет результат обработки гауссовым фильтром в квадрате от –*r* до +*r*, где параметр σ равен 0,38*r*:

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

- ЗАПРЕЩАЕТСЯ использовать готовые функции фильтрации
(например, в PIL есть ImageFilter.BLUR)
- РАЗРЕШАЕТСЯ использовать numpy-функции и пренебрегать эффектами на краях изображения