МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Санкт – Петербургский национальный исследовательский университет информационных технологий, механики и оптики»
Факультет инфокоммуникационных технологий
Кафедра интеллектуальных технологий в гуманитарной сфере

Лабораторная работа № 1

По мультимедийным технологиям.

Выполнил студент группы К3143:
Садовщиков Игорь Иванович

Преподаватель
Хлопотов Максим Валерьевич

САНКТ – ПЕТЕРБУРГ
2019

Лабараторная работа №1

Задание 1.1

In [2]:

```
from skimage import *
from skimage.io import *
from collections import Counter
import matplotlib.pyplot as plt
import skimage.measure
import numpy as np
import math
```

In [8]:

```
def rgb2yuv(filename: str) -> np.ndarray:
    image = io.imread(filename)

    r = image[:,:,0]
    g = image[:,:,1]
    b = image[:,:,2]

    y = 0.299*r+0.587*g+0.114*b
    u = -0.1687*r-0.3313*g+0.5*b+128
    v = 0.5*r-0.4187*g-0.0813*b+128

    yuv = np.array(np.dstack((y, u, v)))
    return yuv
```

In [9]:

```
def yuv2rgb(image: np.ndarray) -> np.ndarray:
    y = image[:,:,0]
    u = image[:,:,1]
    v = image[:,:,2]

    r = np.clip(y+1.402*(v-128), 0, 255)
    g = np.clip(y-0.34414*(u-128)-0.71414*(v-128), 0, 255)
    b = np.clip(y+ 1.772*(u-128), 0, 255)

    rgb = np.ubyte(np.dstack((r,g,b)))
    return rgb
```

Функция brightness() отвечает за скалярное равномерное квантование

In [86]:

```
def brightness(image, factor=16):
    read = np.array(skimage.img_as_ubyte(image))
    divided = np.round(read/factor)
    multiply = np.clip(divided * factor, 0, 255)
    image = multiply.astype('uint8')
    return image
```

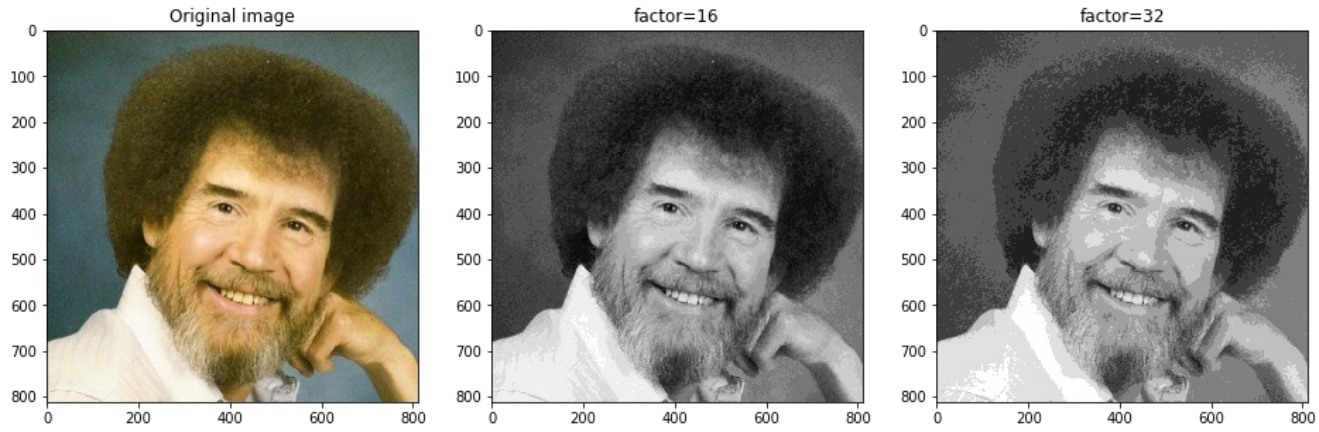
Пример работы функции

In [81]:

```
fig, axes = plt.subplots(ncols = 3)
ax = axes.ravel()
fig.set_size_inches(16,16)

img = io.imread('temp.jpeg', as_grey=True)
```

```
ax[0].imshow(io.imread('temp.jpeg'));
ax[1].imshow(brightness(img, 16), cmap = 'gray');
ax[2].imshow(brightness(img, 32), cmap = 'gray');
ax[0].set_title('Original image')
ax[1].set_title('factor=16')
ax[2].set_title('factor=32')
plt.show()
```



Вывод: чем больше шаг квантования, тем больше будет заметна разница между изображениями. При шаге 256 останутся только контуры изображения, что удобно использовать, например, в нейросетях.

Задание 1.2

In [69]:

```
def rgb2yuv(image: np.ndarray) -> np.ndarray:

    r = image[:,:,0]
    g = image[:,:,1]
    b = image[:,:,2]

    y = 0.299*r+0.587*g+0.114*b
    u = -0.1687*r-0.3313*g+0.5*b+128
    v = 0.5*r-0.4187*g-0.0813*b+128

    yuv = np.array(np.dstack((y, u, v)))
    return yuv
```

In [23]:

```
def yuv2rgb(image: np.ndarray) -> np.ndarray:
    y = image[:,:,0]
    u = image[:,:,1]
    v = image[:,:,2]

    r = np.clip(y+1.402*(v-128), 0, 255)
    g = np.clip(y-0.34414*(u-128)-0.71414*(v-128), 0, 255)
    b = np.clip(y+ 1.772*(u-128), 0, 255)

    rgb = np.ubyte(np.dstack((r,g,b)))
    return rgb
```

In [25]:

```
def decimation(image: np.ndarray, factor=4):
    y = image[:,:,0]
    s = int(factor/2)
    u = skimage.measure.block_reduce(img[:, :, 1], block_size = (s,s), func=np.mean)
    v = skimage.measure.block_reduce(img[:, :, 2], block_size = (s,s), func=np.mean)
    return y, u, v
```

In [ ]:

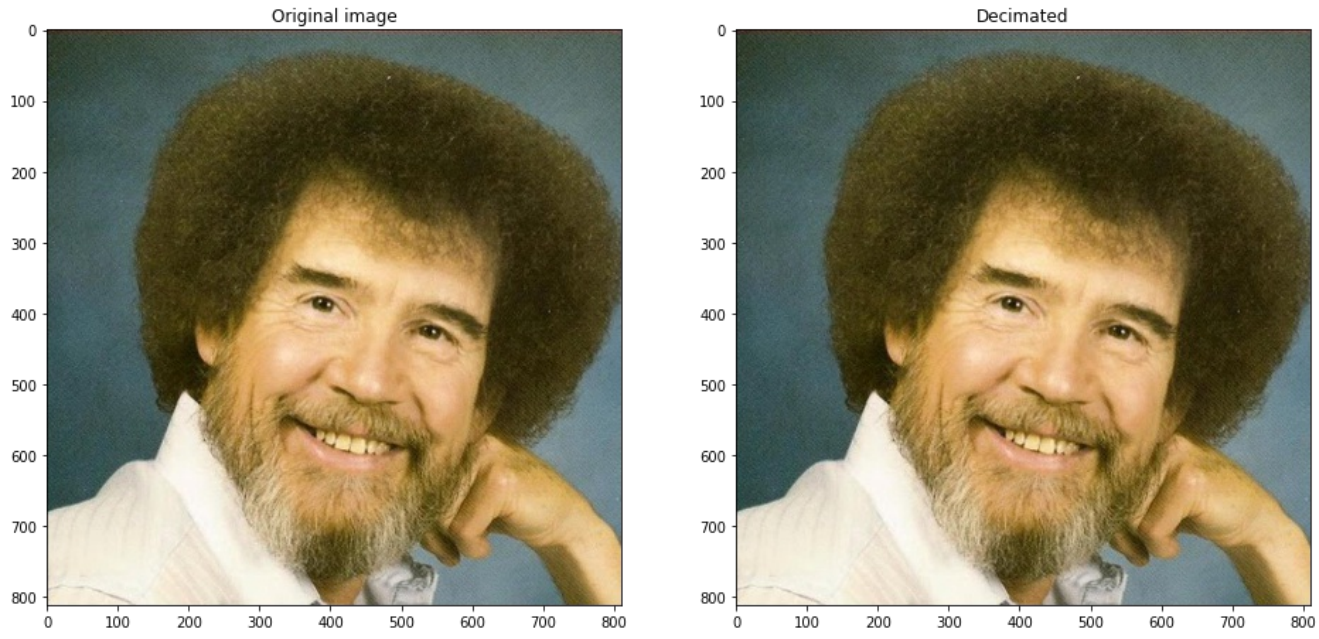
```
def undecimation(y, u, v, factor=4) -> np.ndarray:
    w = y.shape[0]
    h = y.shape[1]
    u0 = np.zeros((w,h))
    v0 = np.zeros((w, h))
    q = 0
    for i in range(0, w, factor):
        w = 0
        for j in range(0, h, factor):
            u0[i:(i + factor), j:(j + factor)] = u[q, w]
            v0e[i:(i + factor), j:(j + factor)] = v[q, w]
            w += 1
        q += 1
    yuv = np.array(np.dstack((y, u, v)))
    return yuv
```

In [74]:

```
fig, axes = plt.subplots(ncols = 2)
ax = axes.ravel()
fig.set_size_inches(16,16)

img = io.imread('temp.jpeg')
dec = decimation(rgb2yuv(img))
dec2 = yuv2rgb(undecimation(dec))

ax[0].imshow(img);
ax[1].imshow(dec2);
ax[0].set_title('Original image')
ax[1].set_title('Decimated')
plt.show()
```



Вывод: Сжатие визуально незаметно, однако размер изображения уменьшается

Задание 1.3

In [61]:

```
def entropy(image: np.ndarray) -> np.ndarray:

    values = Counter()
    for i in range(len(image)):
        for j in range(len(image[i])):
```

```
            values[image[i][j]] += 1

    H = 0
    wh = len(image)*len(image[0])
    for elem in values:
        p = values[elem]/wh
        H -= p * math.log(p, 2)
    return H
```

In [62]:

```
def MSE(image1: np.ndarray, image2: np.ndarray) -> float:
    wh = len(image1)*len(image1[0])
    diff = [(P - Q) ** 2 for P, Q in zip (image1, image2)]
    summ = np.sum(diff)
    MSE = summ/wh
    return MSE
```

Энтропия и среднеквадратичная ошибка для задания 1.1

In [90]:

```
img = skimage.img_as_ubyte(io.imread('temp.jpeg', as_grey=True))
img32 = brightness(img, 32)

print('Энтропия исходного изображения: ', img)
print('Энтропия изображения после квантования (шаг 32): ', entropy(img32))
print('Среднеквадратичная ошибка двух изображений: ', MSE(img, img32))
```

```
Энтропия исходного изображения:  7.644767772975403
Энтропия изображения после квантования (шаг 32):  2.7823782653618534
Среднеквадратичная ошибка двух изображений:  80.21028439012835
```

Вывод: энтропии двух изображений отличаются, потому что изменилась частота появления определенных пикселей.

Энтропия и среднеквадратичная ошибка для задания 1.2

In [67]:

```
def entropy3(image: np.ndarray) -> np.ndarray:
    for n in range(3):
        values = Counter()
        for i in range(len(image[:,:,0])):
            for j in range(len(image[:,:,0][0])):
                values[image[:,:,n][i][j]] += 1

        H = 0
        wh = len(image[:,:, 0])*len(image[:,:, 0][0])
        for elem in values:
            p = values[elem]/wh
            H -= p * math.log(p, 2)
    return H
```

In [68]:

```
def MSE3(image1: np.ndarray, image2: np.ndarray) -> float:
    wh = len(image1[:,:, 0])*len(image1[:,:, 0][0])
    MSE = 0
    for i in range(3):
        diff = [(P - Q) ** 2 for P, Q in zip (image1[:, :, i], image2[:,:, i])]
        summ = np.sum(diff)
        MSE += summ/wh
    return MSE/3
```

In [76]:

```
img = io.imread('temp.jpeg')
dec = decimation(rgb2yuv(img))
dec2 = yuv2rgb(undecimation(dec))

print('Энтропия изображения до децимации: ', entropy3(img))
print('Энтропия изображения после децимации: ', entropy3(dec2))
print('Среднеквадратичная ошибка двух изображений: ', MSE3(img,dec2))
```

```
Энтропия изображения до децимации:  7.560810742721225
Энтропия изображения после децимации:  7.562794281948213
Среднеквадратичная ошибка двух изображений:  1.836482176384114
```

Вывод: энтропии незначительно отличаются для двух изображений, а среднеквадратичная ошибка минимальна, следовательно изображения сжались практически без потерь в качестве