«Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Отчёт по лабораторной работе

«Лабораторная работа о информатике №7»

Выполнил: студент 1 курса Липинский Илья (Группа: M3109).

Приняла: Ханжина Наталья Евгеньевна

Санкт-Петербург 2017

Линейные фильтры

Исходник.

Оттенки серого.

Сепия.

Сепия в красном.

Негатив.

Шумы.

Ярче.

Темнее.

Только ч/б

Мой фильтр.

Матричные фильтры

Исходник.

Размытие.

Повышенная резкость.

Мой матричный фильтр.

```python
import sys, random, copy
from PIL import Image, ImageDraw #Подключим необходимые библиотеки.

IMAGE_PATH = "" #Путь до картинки, с которой будем работать, пустой если находится в той же директоррии, что и программа.
IMAGE_NAME = "Pic" #Имя картинки.
IMAGE_TYPE = "jpg" #Тип картинки, крайне рекомендуется jpg.

original_image = Image.open(IMAGE_PATH + IMAGE_NAME + "." + IMAGE_TYPE) #Открываем изображение.
#draw = ImageDraw.Draw(image) #Создаем инструмент для рисования.
image = Image.open("Pic.jpg")
draw = ImageDraw.Draw(image)
#ЛИНЕЙНЫЕ ФИЛЬТРЫ

#Константное преобразование
def constant(pixel):
    r, g, b = pixel
    return r, g, b
#Выделение красной компоненты.
def Only_red(pixel):
    r, g, b = pixel
    return r, 0, 0
#Выделение зеленой компоненты.
def Only_green(pixel):
    r, g, b = pixel
    return 0, g, 0
#Выделение синей компоненты:
def Only_blue(pixel):
    r, g, b = pixel
    return 0, 0, b
#Оттенки серого:
def Grey(pixel):
    r, g, b = pixel
    return r, r, r
#Сепия
def Sepia(pixel):
    K = 30 #Коэффициент
    r, g, b = pixel
    S = (r + g + b) // 3 #Находим среднее значение
    r = S + K * 2 #Первое значение пиксела ( R )
    g = S + K     #Второе значение пиксела ( G )
    b = S         #Третье значение пиксела ( B )
    if (r > 255):
        r = 255
    if (g > 255):
        g = 255
    if (b > 255):
        b = 255
    return r, g, b
#Сепия с красным оттенком
def Sepia_red(pixel):
    K = 30 #Коэффициент
    r, g, b = pixel
    S = (r + g + b) // 3 #Находим среднее значение
    r = S + K * 2
    g = S + K
    b = S
    if (r > 255):
        r = 255
    if (g > 255):
        g = 255
    if (b > 255):
        b = 255
```

```
    return r + 150, g, b
#Негатив
def Negative(pixel):
    r, g, b = pixel
    return 255-r, 255-g, 255-b
#Шумы
def Noises(pixel):
    r, g, b = pixel
    rand = random.randint(-70, 70) #Диапазон
    r+=rand
    g+=rand
    b+=rand
    if (r < 0):
        r = 0
    if (g < 0):
        g = 0
    if (b < 0):
        b = 0
    if (r > 255):
        r = 255
    if (g > 255):
        g = 255
    if (b > 255):
        b = 255
    return  r, g, b
#Повышенная яркость
def Bright(pixel):
    r, g, b = pixel
    r+=150
    g+=150
    b+=150
    if (r < 0):
        r = 0
    if (g < 0):
        g = 0
    if (b < 0):
        b = 0
    if (r > 255):
        r = 255
    if (g > 255):
        g = 255
    if (b > 255):
        b = 255
    return r, g, b
#Пониженная яркость
def Dark(pixel):
    r, g, b = pixel
    r-=150
    g-=150
    b-=150
    if (r < 0):
        r = 0
    if (g < 0):
        g = 0
    if (b < 0):
        b = 0
    if (r > 255):
        r = 255
    if (g > 255):
        g = 255
```

```
    if (b > 255):
        b = 255
    return r, g, b
#Черный или белый
def BlackOrWhite(pixel):
    r, g, b = pixel
    S=r+g+b
    if (S > (((255 + 100) // 2) * 3)):
        r, g, b = 255, 255, 255
    else:
        r, g, b = 0, 0, 0
    return r, g, b
#Свой фильтр
def MyFilter(pixel):
    r, g, b = pixel
    return r + 20, g + 10, b + 100


#Выделение красной компоненты.
#Список преобразований
linear_transformations = [constant, Only_red, Only_green, Only_blue,
                          Grey,Sepia, Sepia_blue, Negative, Noises,
                          Bright, Dark,BlackOrWhite, MyFilter]

#Последовательное применение всех преобразований с сохранением результата.
for transformation in linear_transformations:
    image = original_image.copy() #Создаем новое изображение, чтобы не испортить оригинальное.
    width = image.size[0] #Определяем ширину.
    height = image.size[1] #Определяем высоту.
    pixels = image.load() #Выгружаем значения пикселей.

    #Перебираем каждый пиксель
    for i in range(width):
        for j in range(height):
            pixels[i, j] = transformation(pixels[i, j]) #Применяем текущее преобразование.

    image.save(IMAGE_PATH + IMAGE_NAME + "_" + transformation.__name__ + "." + IMAGE_TYPE); #Сохранение картинки.



#МАТРИЧНЫЕ ФИЛЬТРЫ

#Константная матрица 3x3.
def const():
    return [ [0, 0, 0], [0, 1, 0], [0, 0, 0] ]

#Фильтр увеличения резкости
def Sharpness():
    return [ [-1, -1, -1], [-1, 9, -1], [-1, -1, -1] ]

#Фильтр размытия (по Гауссу)
def Gauss_blur():
    return [ [0.000789, 0.006581, 0.013347, 0.006581, 0.000789],
             [0.006581, 0.054901, 0.111345, 0.054901, 0.006581],
             [0.013347, 0.111345, 0.225821, 0.111345, 0.013347],
             [0.006581, 0.054901, 0.111345, 0.054901, 0.006581],
             [0.000789, 0.006581, 0.013347, 0.006581, 0.000789] ]

#Мой матричный фильтр
def MyMatrixFilter():
```

```python
def MyMatrixFilter():
    return [ [1,0 ,0 ,0 ,1], [0 ,0 ,0 ,0 ,0],[0 ,0 ,0 ,0 ,0],[0 ,0 ,0 ,0 ,0],[1,0 ,0 ,0 ,1] ]

matrix_filters = [const, Sharpness, Gauss_blur, MyMatrixFilter]


def matrix_transformation(old_pixels, width, height, x, y, get_matrix):
    matrix = get_matrix()
    n = len(matrix)  # Узнаем размерность матрицы.
    new_color = [0, 0, 0]
    matrix_sum = 0  # Посчитаем сумму в матрице преобразования для того, чтобы потом поделить на это значение.
    # Таким образом интенсивность изображения не измениться.
    # Перебираем соседей
    for i in range(n):
        for j in range(n):
            new_x = x - n // 2 + i  # Вычисляем координату соседа, с учетом того, что "мы" в центре матрицы.
            new_y = y - n // 2 + j

            # Проверяем соседа на существование.
            if 0 <= new_x < width and 0 <= new_y < height:
                matrix_sum += matrix[i][j]
                # Перебираем цветовую компоненту.
                for c in range(3):
                    new_color[c] += old_pixels[new_x, new_y][c] * matrix[i][
                        j]  # Добавляем цвет соседа умноженный на коэффициент из матрицы.

    for c in range(3):
        if matrix_sum != 0:
            new_color[c] /= matrix_sum  # Нормируем цвет.
        else:
            new_color[c] = 0
    return int(new_color[0]), int(new_color[1]), int(new_color[2])


for matrix in matrix_filters:
    image = original_image.copy()  # Создаем новое изображение, чтобы не испортить оригинальное.
    width, height = image.size  # Определяем ширину и высоту
    pixels = image.load()  # Выгружаем значения пикселей.
    old_pixels = original_image.load()  # Выгружаем значения пикселей оригинального изображения.

    # Перебираем каждый пиксель
    for i in range(width):
        for j in range(height):
            pixels[i, j] = matrix_transformation(old_pixels, width, height, i, j,
                                                 matrix)  # Применяем текущее преобразование.

    image.save(IMAGE_PATH + IMAGE_NAME + "_matrix_" + matrix.__name__ + "." + IMAGE_TYPE)  # Сохранение картинки.
```