

МГУ им. М.Ю.Ломоносова

Кафедра вычислительной механики

Отчёт по работе с BMP-изображениями в Python-3

Студент:	Скворцов Андрей Сергеевич
Преподаватель:	Почеревин Роман Владимирович
Группа:	223

Москва
2024

Отчёт по работе с BMP-изображениями в Python-3

Задача. Реализовать алгоритмы фрактального сжатия и восстановления изображения.

Решение. Используя библиотеку PIL, можно решить эту задачу эффективнее, чем, например, imageio:

```
from PIL import Image, ImageDraw
```

Пусть изображения заданы внутри программы. Выгрузим, посчитаем длину и ширину:

```
imin = Image.open("input.bmp")
```

```
x1, y1 = imin.size
```

```
px1 = imin.load()
```

Далее создадим выходное изображение и возможность изменять в нем пиксели, а также введем необходимые переменные:

```
imout = Image.new("RGB (x1, y1), (0, 0, 0)) draw = ImageDraw.Draw(imout)
```

$$size_square = 15 fractal = 0 \quad (1)$$

Теперь - логика программы. Найдем среднюю яркость квадратного блока со стороной $size_square$ и запишем ее в переменную $factor$. Затем найдем среднюю яркость 3 соседних блоков того же размера и сохраним их значения в список $factora$.

```
for i in range(0, x1, size_square) :      for j in range(0, y1, size_square) :
    if i + 2 * size_square - 1 < x1 and j + 2 * size_square - 1 < y1 :
        for k in range(size_square) :      for t in range(size_square) :
            r = px1[i+k, j+t][0]          g =
px1[i + k, j + t][1]                    b = px1[i + k, j + t][2]
            fractal += (r + g + b) // 3
        fractala = [0] * 3
        for k in range(size_square) :      for t in range(size_square) :
            r = px1[i+size_square+k, j+t][0]          g =
px1[i + size_square + k, j + t][1]        b = px1[i +
size_square + k, j + t][2]
            fractala[0] += (r + g + b) // 3
        for k in range(size_square) :      for t in range(size_square) :
            r = px1[i+k, j+size_square+t][0]          g =
px1[i+k, j + size_square + t][1]        b = px1[i+k, j +
size_square + t][2]
            fractala[1] += (r + g + b) // 3
        for k in range(size_square) :      for t in range(size_square) :
            r = px1[i + size_square + k, j + size_square +
g = px1[i + size_square + k, j + size_square +
b = px1[i + size_square + k, j + size_square +
t][0]
t][1]
t][2]
            fractala[2] += (r + g + b) // 3
```

Теперь посмотрим какой из этих 3 блок больше похож на первый сравнивая их по средней яркости и заменяя все 4 блока на самый похожий:

```

fractala[0] = abs(fractal - fractala[0]) fractala[1]
= abs(fractal - fractala[1]) fractala[2] = abs(fractal - fractala[2])
mindifference = min(fractala)
index = fractala.index(mindifference)
for k in range(2 * sizesquare) : for t in range(2 *
sizesquare) :
    if index == 0 : draw
    elif index == 1 :
        draw.point((i+k, j+t), (px1[i+k
        draw.point((i+k, j+t), (px1[i+sizesquare+
k

```

Теперь нарисуем оставшиеся блоки также как в исходном изображении:

```

else: for k in range(x1 - i): for
t in range(y1 - j): draw.point((i + k, j + t), (px1[i
+ k, j + t][0], px1[i + k, j + t][1], px1[i + k, j + t][2]))

```

Сохраним изображение и очистим память от элемента *draw*:

```

imout.save("out.bmp "BMP")
del draw

```