МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

ЛАБОРАТОРНАЯ РАБОТА

Алгоритм фрактального сжатия

студента 4 курса 431 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Серебрякова Алексея Владимировича

Научный руководитель

доцент, к. п. н. _____ А. С. Гераськин

подпись, дата

Саратов 2022

Основы фрактального сжатия изображений

Алгоритмы*

Из песочницы

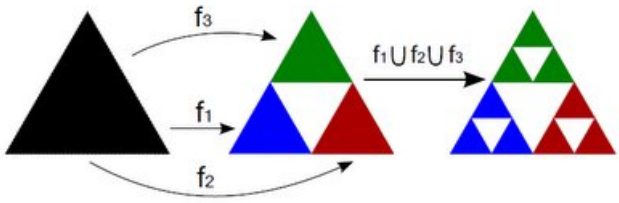
Фракталы — удивительные математические объекты, подкупающие своей простотой и богатыми возможностями по построению объектов сложной природы при помощи всего лишь нескольких коэффициентов и простой итеративной схемы.
Именно эти возможности и позволяют использовать их для сжатия изображений, особенно для фотографий природы и прочих сложных самоподобных изображений.
В этой статье я постараюсь коротко дать ответ на простой вопрос: «Как же это делается?».

Для начала все-таки придется немного углубиться в математику и ввести несколько определений. Нам пригодятся следующие:

- Метрика — функция, заданная на пространстве, возвращающая расстояние между двумя точками этого пространства. Например, привычная нам Эвклидова метрика. Если на пространстве задана метрика, оно называется метрическим. Метрика должна удовлетворять определенным условиям, но не будем углубляться.
- Сжимающее отображение (преобразование) — функция на метрическом пространстве, равномерно уменьшающая расстояние между двумя точками пространства. Например, y=0.5x.

Сжимающие отображения обладают важным свойством. Если взять любую точку и начать итеративно применять к ней одно и то же сжимающее отображение: f(f(f...f(x))), то результатом будет всегда одна и та же точка. Чем больше раз применим, тем точнее найдем эту точку. Называется она *неподвижной точкой* и для каждого сжимающего отображения она существует, причем только одна.

Несколько аффинных сжимающих отображений образуют систему итерированных функций (СИФ). По сути, СИФ — это множительная машина. Она принимает исходное изображение, искажает его, перемещает, и так несколько раз.
Например, вот так при помощи СИФ из трех функций строится треугольник Серпинского:



$$f_1\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$f_2\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1/2 \\ 0 \end{bmatrix}$$
$$f_3\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1/4 \\ \sqrt{3}/4 \end{bmatrix}$$

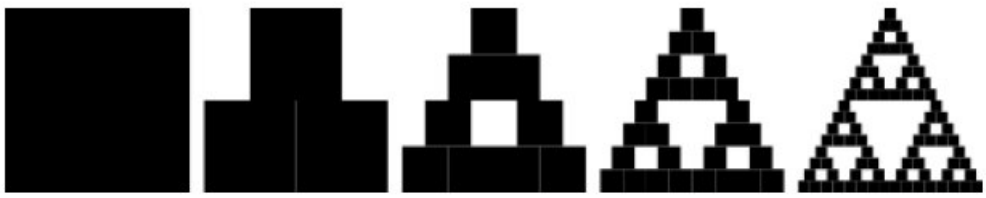
Исходный треугольник три раза множится, уменьшается и переносится. И так далее. Если так продолжать до бесконечности, получим известный фрактал площадью 0 и размерностью 1,585.

Если функции, входящие в СИФ,— сжимающие, то сама СИФ тоже имеет неподвижную точку. Вот только эта «точка» будет уже не привычной нам точкой в N-мерном пространстве, а множеством таких точек, то есть изображением. Оно называется *аттрактором* СИФ. Для СИФ, приведенной выше, аттрактором будет треугольник Серпинского.

Тут мы переходим на следующий уровень — в пространство изображений. В этом пространстве:

- Точка пространства — это изображение.
- Расстояние между точками показывает, насколько похожи изображения между собой, насколько «близки» (естественно, если его задать соответствующим образом).
- Сжимающее отображение делает два любых изображения более похожими (в смысле заданной метрики).

Имея СИФ, найти аттрактор просто. Во всяком случае, имея под рукой компьютер. Можно взять абсолютно любое начальное изображение и начать применять к нему СИФ. Пример с тем же треугольником, но уже построенным из квадрата:



Получается, что для построения довольно сложной фигуры нам потребовалось 18 коэффициентов. Выигрыш, как говорится, налицо.
Вот если бы мы умели решать обратную задачу — имея аттрактор, строить СИФ… Тогда достаточно взять аттрактор-изображение, похожее на фотографию вашей кошки и можно довольно выгодно его закодировать.

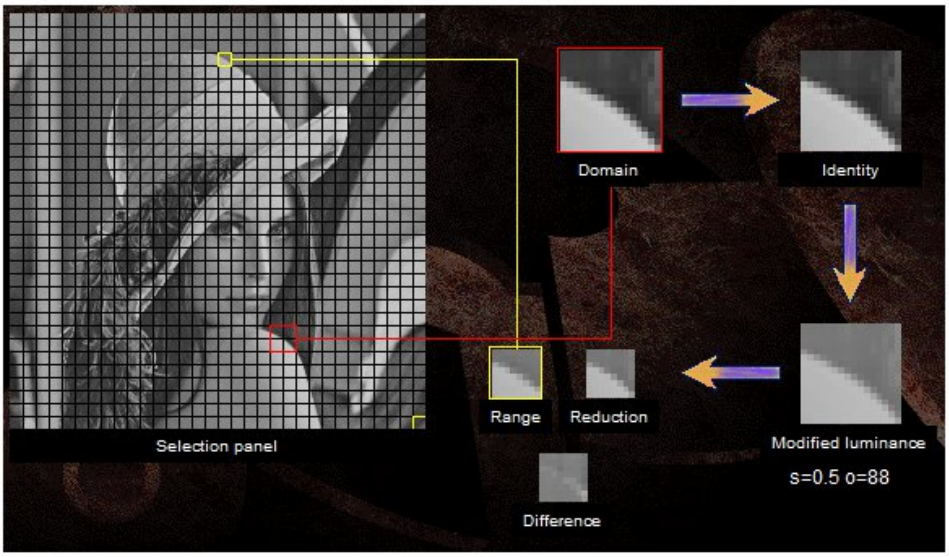
Вот здесь, собственно, начинаются проблемы. Произвольные изображения, в отличие от фракталов, не самоподобны, так что так просто эта задача не решается. Как это сделать придумал в 1992 году Арнольд Жакин, в то время — аспирант Майкла Барнсли, который считается отцом фрактального сжатия.

Самоподобие нам нужно обязательно — иначе ограниченные в своих возможностях аффинные преобразования не смогут правдоподобно приблизить изображения. А если его нет между частью и целым, то можно поискать между частью и частью. Примерно так, видимо, рассуждал и Жакин.

Упрощенная схема кодирования выглядит так:

- Изображение делится на небольшие неперекрывающиеся квадратные области, называемые ранговыми блоками. По сути, разбивается на квадраты. См. картинку ниже.
- Строится пул всех возможных перекрывающихся блоков в четыре раза больших ранговых — доменных блоков.
- Для каждого рангового блока по очереди «примеряем» доменные блоки и ищем такое преобразование, которое делает доменный блок наиболее похожим на текущий ранговый.
- Пара «преобразование-доменный блок», которая приблизилась к идеалу, ставится в соответствие ранговому блоку. В закодированное изображение сохраняются коэффициенты преобразования и координаты доменного блока. Содержимое доменного блока нам ни к чему — вы же помните, нам все равно с какой точки стартовать.

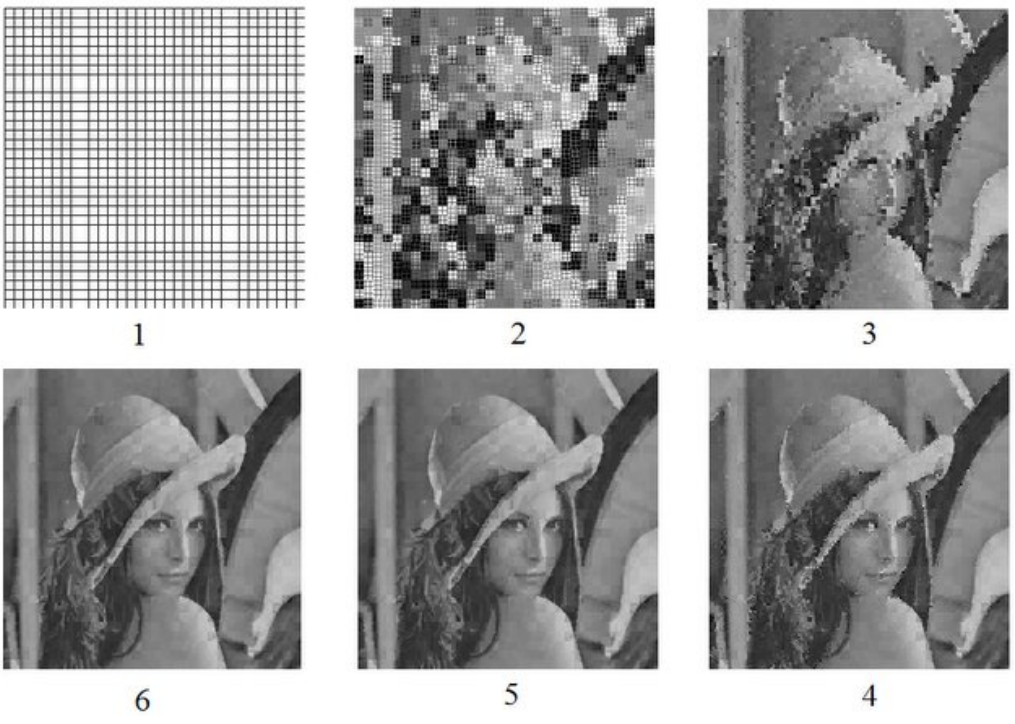
На картинке ранговый блок обозначен жёлтым, соответствующий ему доменный — красным. Также показаны этапы преобразования и результат.



Можете поиграться сами: Coder demo.

Получение оптимального преобразования — отдельная тема, однако большого труда оно не составляет. Но другой недостаток схемы виден невооруженным глазом. Можете сами подсчитать, сколько доменных блоков размером 32×32 содержит двухмегапиксельное изображение. Полный их перебор для каждого рангового блока и есть основная проблема такого вида сжатия — кодирование занимает очень много времени. Разумеется, с этим борются при помощи различных ухищрений, вроде сужения области поиска или предварительной классификации доменных блоков. С различным ущербом для качества.

Декодирование же производится просто и довольно быстро. Берем любое изображение, делим на ранговые области, последовательно заменяем их результатом применения соотв. преобразования к соотв. доменной области (что бы она ни содержала в данный момент). После нескольких итераций исходное изображение станет похоже на себя:



Код программы

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage
from scipy import optimize
import numpy as np
import math
import time
import os
np.warnings.filterwarnings('ignore')

def get_greyscale_image(img):
    return np.mean(img[:, :, :2], 2)

def extract_rgb(img):
    return img[:, :, 0], img[:, :, 1], img[:, :, 2]

def assemble_rbg(img_r, img_g, img_b):
    shape = (img_r.shape[0], img_r.shape[1], 1)
    return np.concatenate((np.reshape(img_r, shape), np.reshape(img_g, shape),
                           np.reshape(img_b, shape)), axis=2)

def reduce(img, factor):
    result = np.zeros((img.shape[0] // factor, img.shape[1] // factor))
    for i in range(result.shape[0]):
        for j in range(result.shape[1]):
            result[i, j] = np.mean(img[i * factor:(i + 1) * factor, j * factor:(j + 1) * factor])
    return result

def rotate(img, angle):
    return ndimage.rotate(img, angle, reshape=False)

def flip(img, direction):
    return img[::direction, :]

def apply_transformation(img, direction, angle, contrast=1.0, brightness=0.0):
    return contrast * rotate(flip(img, direction), angle) + brightness

def find_contrast_and_brightness1(D, S):
    contrast = 0.75
    brightness = (np.sum(D - contrast * S)) / D.size
    return contrast, brightness

def find_contrast_and_brightness2(D, S):
    A = np.concatenate((np.ones((S.size, 1)), np.reshape(S, (S.size, 1))), axis=1)
    b = np.reshape(D, (D.size,))
    x, _, _, _ = np.linalg.lstsq(A, b)
    return x[1], x[0]

def generate_all_transformed_blocks(img, source_size, destination_size, step):
    factor = source_size // destination_size
    transformed_blocks = []
    for k in range((img.shape[0] - source_size) // step + 1):
        for l in range((img.shape[1] - source_size) // step + 1):
            # Extract the source block and reduce it to the shape of a destination block
            S = reduce(img[k * step:k * step + source_size, l * step:l * step + source_size], factor)
            # Generate all possible transformed blocks
```

```
            for direction, angle in candidates:
                transformed_blocks.append((k, l, direction, angle, apply_transformation(S, direction, angle)))
    return transformed_blocks

def compress(img, source_size, destination_size, step):
    transformations = []
    transformed_blocks = generate_all_transformed_blocks(img, source_size, destination_size, step)
    i_count = img.shape[0] // destination_size
    j_count = img.shape[1] // destination_size
    for i in range(i_count):
        transformations.append([])
        for j in range(j_count):
            print("{}/{} ; {}/{}".format(i, i_count, j, j_count))
            transformations[i].append(None)
            min_d = float('inf')
            D = img[i * destination_size:(i + 1) * destination_size, j * destination_size:(j + 1) * destination_size]
            for k, l, direction, angle, S in transformed_blocks:
                contrast, brightness = find_contrast_and_brightness2(D, S)
                S = contrast * S + brightness
                d = np.sum(np.square(D - S))
                if d < min_d:
                    min_d = d
                    transformations[i][j] = (k, l, direction, angle, contrast, brightness)
    return transformations

def decompress(transformations, source_size, destination_size, step, nb_iter=8):
    factor = source_size // destination_size
    height = len(transformations) * destination_size
    width = len(transformations[0]) * destination_size
    iterations = [np.random.randint(0, 256, (height, width))]
    cur_img = np.zeros((height, width))
    for i_iter in range(nb_iter):
        print(i_iter)
        for i in range(len(transformations)):
            for j in range(len(transformations[i])):
                k, l, flip, angle, contrast, brightness = transformations[i][j]
                S = reduce(iterations[-1][k * step:k * step + source_size, l * step:l * step + source_size], factor)
                D = apply_transformation(S, flip, angle, contrast, brightness)
                cur_img[i * destination_size:(i + 1) * destination_size,
                j * destination_size:(j + 1) * destination_size] = D
        iterations.append(cur_img)
        cur_img = np.zeros((height, width))
    return iterations

def reduce_rgb(img, factor):
    img_r, img_g, img_b = extract_rgb(img)
    img_r = reduce(img_r, factor)
    img_g = reduce(img_g, factor)
    img_b = reduce(img_b, factor)
    return assemble_rbg(img_r, img_g, img_b)

def compress_rgb(img, source_size, destination_size, step):
    img_r, img_g, img_b = extract_rgb(img)
    return [compress(img_r, source_size, destination_size, step), \
            compress(img_g, source_size, destination_size, step), \
            compress(img_b, source_size, destination_size, step)]

def decompress_rgb(transformations, source_size, destination_size, step, nb_iter=8):
    img_r = decompress(transformations[0], source_size, destination_size, step, nb_iter)[-1]
    img_g = decompress(transformations[1], source_size, destination_size, step, nb_iter)[-1]
    img_b = decompress(transformations[2], source_size, destination_size, step, nb_iter)[-1]
    return assemble_rbg(img_r, img_g, img_b)
```

```
def plot_iterations(iterations, target=None):
    # Configure plot
    plt.figure()
    nb_row = math.ceil(np.sqrt(len(iterations)))
    nb_cols = nb_row
    # Plot
    for i, img in enumerate(iterations):
        plt.subplot(nb_row, nb_cols, i + 1)
        plt.imshow(img, cmap='gray', vmin=0, vmax=255, interpolation='none')
        if target is None:
            plt.title(str(i))
        else:
            # Display the RMSE
            plt.title(str(i) + ' (' + '{0:.2f}'.format(np.sqrt(np.mean(np.square(target - img)))) + ')')
        frame = plt.gca()
        frame.axes.get_xaxis().set_visible(False)
        frame.axes.get_yaxis().set_visible(False)
    plt.tight_layout()

directions = [1, -1]
angles = [0, 90, 180, 270]
candidates = [[direction, angle] for direction in directions for angle in angles]

def test_greyscale():
    print("0 - Сжатие, 1 - Расжатие")
    oper = input()

    if oper == "0":
        start_time = int(round(time.time()))
        img = mpimg.imread('studies\\tkisi\\monkey.png')
        imgg = img
        img = get_greyscale_image(img)
        img = reduce(img, 4)
        plt.figure()
        plt.imshow(img, cmap='gray', interpolation='none')
        transformations = compress(img, 8, 4, 8)
        end_time = int(round(time.time()))
        plt.savefig('studies\\tkisi\\comp_monkey.png')
        print("Скорость сжатия: " + str((end_time - start_time)) + " сек")
        size_origin = os.stat('studies\\tkisi\\monkey.png').st_size
        size_compress = os.stat('studies\\tkisi\\comp_monkey.png').st_size
        print("Коэффициент сжатия: ")
        print(round(size_compress / size_origin, 3))
        width = np.size(imgg, 1)
        height = np.size(imgg, 0)
        quality = (101 - ((width * height * 3) / size_compress))
        print("Качество сжатия: " + str(quality))

    if oper == "1":
        img = mpimg.imread('studies\\tkisi\\monkey.png')
        imgg = img
        img = get_greyscale_image(img)
        img = reduce(img, 4)
        plt.figure()
        plt.imshow(img, cmap='gray', interpolation='none')
        transformations = compress(img, 8, 4, 8)
        iterations = decompress(transformations, 8, 4, 8)
        plt.imshow(imgg)
        plt.savefig('studies\\tkisi\\decomp_monkey.png')

def test_rgb():
```

```
    print("0 - Сжатие, 1 - Расжатие")
    oper = input()

    if oper == "0":
        start_time = int(round(time.time()))
        img = mpimg.imread('studies\\tkisi\\lena.gif')
        imgg = img
        img = reduce_rgb(img, 8)
        transformations = compress_rgb(img, 8, 4, 8)
        plt.figure()
        plt.subplot(121)
        plt.imshow(np.array(img).astype(np.uint8), interpolation='none')
        end_time = int(round(time.time()))
        plt.savefig('studies\\tkisi\\comp_lena.png')
        print("Скорость сжатия: " + str((end_time - start_time)) + " сек")
        size_origin = os.stat('studies\\tkisi\\lena.gif').st_size
        size_compress = os.stat('studies\\tkisi\\comp_lena.png').st_size
        print("Коэффициент сжатия: ")
        print(round(size_compress / size_origin, 3))
        width = np.size(imgg, 1)
        height = np.size(imgg, 0)
        quality = (101 - ((width * height) * 3) / size_compress)
        print("Качество сжатия: " + str(quality))

    if oper == "1":
        img = mpimg.imread('studies\\tkisi\\lena.gif')
        imgg = img
        img = reduce_rgb(img, 8)
        transformations = compress_rgb(img, 8, 4, 8)
        retrieved_img = decompress_rgb(transformations, 8, 4, 8)
        plt.figure()
        plt.subplot(121)
        plt.imshow(np.array(img).astype(np.uint8), interpolation='none')
        plt.subplot(122)
        plt.imshow(imgg)
        plt.savefig('studies\\tkisi\\decomp_lena.png')

if __name__ == '__main__':
    test_greyscale()
    # test_rgb()
```