МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра Прикладная математика и информатика
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки / специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Сравнение методов сжатия данных»

Обучающийся М. С. Сорокин _____
(Инициалы Фамилия) (личная подпись)

Руководитель к.ф.м.н. доцент Г. А. Тырыгина
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант канд. филол. наук, М.В. Дайнеко
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Тема выпускной квалификационной работы: «Сравнение методов сжатия данных».

Выпускная квалификационная работа состоит из введения, трёх разделов, 28 рисунков, заключения, списка литературы и источников, состоящего из 20 наименований, среди которых есть и иностранные.

Объектом исследования является сжатие данных.

Предметом исследования являются методы сжатия изображений.

Целью работы является сравнение фрактального сжатия изображений со стандартным методом JPEG (Joint Photographic Experts Group).

Для выполнения поставленной цели нужно решить следующие задачи: описать методы сжатия данных, описать математические основы фрактального сжатия изображений, осуществить программную реализацию алгоритма фрактального сжатия изображений, оценить его эффективность по сравнению с устоявшимися методами сжатия.

Первый раздел посвящен теоретической информации о методах сжатия данных, где особое внимание уделено фрактальному сжатию изображений.

Второй раздел описывает математическую модель фрактального сжатия изображений.

В третьем разделе описана структура программной реализации алгоритма фрактального сжатия и проведен анализ его эффективности.

В заключении, на основании проведенных испытаний, подчеркивается, что алгоритм фрактального сжатия изображений недостаточно эффективен, что можно исправить упрощением вычислений при компрессии.

Работа представляет интерес для читателей, интересующихся нестандартными методами сжатия изображений.

Abstract

The title of a graduation work is «comparison of data compression methods».

The graduation work consists of an introduction, three parts, 28 figures, a conclusion and a list of 20 references including foreign sources.

The object of the study is data compression.

The subject of the study is image compression methods.

The purpose of the work: Comparison of fractal image compression with standard JPEG (Joint Photographic Experts Group) method.

In order to fulfil the set goal, it is necessary to solve the following tasks: describe the methods of data compression, describe the mathematical foundations of fractal image compression, carry out the software implementation of the algorithm of fractal image compression, evaluate its efficiency in comparison with the established compression methods.

The first section is devoted to theoretical information about data compression techniques, where fractal image compression is emphasized.

The second section describes the mathematical model of fractal image compression.

The third section describes the structure of the software implementation of the algorithm and analyzes its efficiency.

In conclusion, based on the performed tests, it is emphasized that the fractal image compression algorithm is not efficient enough, which can be remedied by simplifying the computations in compression.

The work is of interest to readers interested in non-standard methods of image compression.

Содержание

Введение 5
1 Обзор методов сжатия данных 7
1.1 Описание методов сжатия данных 7
1.2 Фрактальный подход 8
2 Математические основы фрактального сжатия изображений 16
2.1 Основные принципы 16
2.2 Математическая модель метода 18
2.3 Декодирование изображения 27
3 Программная реализация 29
3.1 Описание структуры программы 29
3.2 Тестирование программы 35
3.3 Анализ эффективности алгоритма 38
Заключение 42
Список используемой литературы и используемых источников 43

Введение

В современном мире объем цифровой информации стремительно увеличивается, вместе с ним увеличиваются требования к оборудованию для хранения и передачи этой информации, в частности, изображений, которые, в зависимости от своих характеристик могут занимать много дискового пространства, а передача подобных файлов может сильно затянуться. Для оптимизации хранения и уменьшения требований к объему памяти применяются методы сжатия изображений.

Среди великого множества различных алгоритмов сжатия, особое место занимает фрактальный метод. В основе метода лежит принцип поиска и воспроизведения самоподобных структур в изображении. Уменьшение объёма достигается путем представления этих структур в более компактном виде. Используя этот метод, можно достичь высокой степени сжатия, не жертвуя при этом качеством.

Актуальность. При текущем уровне развития мультимедийных технологий, вопрос эффективного сжатия изображений без сильной потери качества является очень актуальным, и в таких обстоятельствах метод сжатия при помощи фракталов может быть хорошим вариантом.

Объект исследования: сжатие данных

Предмет исследования: методы сжатия изображений

Цель работы: сравнение фрактального сжатия изображений со стандартным методом JPEG. Для выполнения цели нужно решить следующие задачи:

- описать методы сжатия данных;
- описать математические основы фрактального сжатия изображений;
- осуществить программную реализацию алгоритма фрактального сжатия изображений, оценить его эффективность по сравнению с устоявшимися методами сжатия.

В первом разделе работы рассматриваются теоретические сведения о фракталах, приводятся определения основных свойств и описываются основная применения фрактального подхода в различных областях. Особое внимание уделяется фрактальному сжатию изображений, которому посвящена дальнейшая работа.

Во втором разделе описывается общий принцип фрактального сжатия изображений, а также его математическая модель. Подробно описываются преобразования, через которые проходит исходное изображение на этапе кодирования и этапе декодирования.

В третьем разделе представлена программная реализация алгоритма, рассматриваются основные модули программы и их функционал, приводятся результаты работы и оценка эффективности по сравнению с массово распространенным форматом JPEG. Также приведены способы улучшения алгоритма.

1 Обзор методов сжатия данных

1.1 Описание методов сжатия данных

Сжатие данных – это процесс уменьшения объема информации для экономии места на носителях или ускорения передачи по каналам связи. Для сжатия пригодны следующие виды данных:

– текстовые данные,

– аудиоданные,

– видеоданные,

– изображения.

Для сжатия разных видов данных существуют разные методы, отличающиеся эффективностью и сложностью реализации, глобально методы сжатия разделяют на два вида: с потерями и без потерь. Для текстовых данных используются методы сжатия без потерь, к которым относят:

– RLE (Run-Length Encoding). Повторы символов заменяются числом повторений,

– кодирование Хаффмана. Часто встречающиеся символы кодируются короткими цепочками битов, редко встречающиеся – длинными цепочками. Составляется дерево кодов, из которого можно восстановить исходный текст.

Для сжатия аудиоданных применяют как методы с потерями, так и без:

– MP3 (MPEG-1 Layer 3). Является методом с потерями, при сжатии удаляются звуки, которые не воспринимаются человеческим ухом,

– FLAC (Free Lossless Audio Codec). Метод сжатия без потерь. Он делит звук на фрагменты, предсказывает значения следующих на основе предыдущих, сохраняет только разницу и эффективно кодирует её. В итоге файл становится меньше, но звучит так же, как оригинал.

Для сжатия видеоданных, могут применяться те же методы, что и для изображений, но есть и специфические:

– межкадровое кодирование. Алгоритм находит различия между кадрами и сохраняет только разницу, используется в видеокодеке H.264.

Для сжатия статичных изображений используются следующие методы:

а) без потерь (используются в форматах PNG, GIF):

1) удаление повторяющихся пикселей через RLE,

2) предсказание цвета следующих пикселей и сохранение разницы;

б) с потерями (JPEG, WebP, фрактальное сжатие)

1) разделение изображения на блоки и преобразование в частоты,

2) округление малозаметных деталей,

3) сжатие полученных коэффициентов.

В рамках этой выпускной квалификационной работы будут рассмотрены именно методы сжатия изображений, в частности алгоритм фрактального сжатия и его эффективность.

1.2 Фрактальный подход

История фракталов берет свое начало в XIX веке, в то время математика столкнулась с объектами, обладающими свойством самоподобия и бесконечной сложностью. Первые такие объекты были обнаружены при изучении математических парадоксов. Например, Карл Вейерштрасс описал функцию, которая была непрерывной, но нигде не имела производной, а Жордан Кантор разработал множество, состоящее из бесконечного количества удаляемых отрезков, и при этом сохраняющее свою структуру.

Исследование этих объектов продолжилось в XX веке. Хельге фон Кох изобразил кривую, усложняющуюся с каждой последующей итерацией, но сохраняющей общую форму. В последствии ее назовут кривой Коха или снежинкой Коха, из-за общей схожести фигуры с настоящей снежинкой. В этот же период Вацлав Серпинский изобразил геометрические конструкции, со

схожими свойствами самоподобия. Среди самых известных выделяется треугольник Серпинского.

Значительный вклад в развитие теории фракталов внес Бенуа Мандельброт. В середине XX века он начал исследовать процессы, выходящие за рамки традиционной евклидовой геометрии. Наблюдая за физическими и природными явлениями, он обнаружил, что в некоторых из них, вроде облаков, форм береговых линий и характере скопления галактик, присутствует самоподобие. В 1975 году он ввел термин фрактал – объект, частично или полностью состоящий из своих уменьшенных копий. А в 1980 году он описал один из самых известных, на сегодняшний день, фракталов – множество Мандельброта.

Фракталы отличаются от классических геометрических фигур рядом признаков, среди которых: самоподобие, рекурсивность, дробная размерность и сложная структура при простых правилах построения. Рассмотрим их далее, с приведением примеров.

Самоподобие. Это означает, что части объекта похожи на весь объект вцелом. Если части повторяют целую фигуру в точности, то самоподобие называют строгим. Если части лишь частично сохраняют форму целого, то это называется статическим самоподобием.

Примеры:

- снежинки – каждая ветвь повторяет форму всей фигуры,
- романеско – структура соцветий повторяется на разных уровнях увеличения,
- береговые линии – на любом масштабе сохраняется извилистая форма.

Рисунок 1 – Самоподобная кристаллическая структура

Рекурсивность. Фракталы строятся по принципу рекурсии. В ходе построения одна и та же операция повторяется многократно и из простых фигур получаются более сложные.

Примеры:

– древесные ветви и корни – каждая ветвь делится на меньшие ветви аналогичной формы,
– кровеносные сосуды и бронхи – аналогично ветвям деревьев, из крупных сосудов выходят более мелкие.

Рисунок 2 – Рекурсивность на примере ветвей

Дробная размерность. В то время как классические геометрические фигуры при приближении не обретают новых деталей, фракталы могут бесконечно обретать новые детали. Это свойство называют дробной размерностью.

Примеры:

– облака – при постепенном приближении, структура облаков будет становиться более комплексной,
– горные хребты – контуры гор выглядят более детализированными при приближении, появляются новые детали, которые нельзя было разглядеть ранее.

Рисунок 3 – Дробная размерность на примере гор

Сложная структура при простых правилах построения. Фракталы формируются на основе простых математических правил, которые при многократном повторении приводят к сложным узорам.

Пример из природы:

– раковины моллюсков – повторяющаяся спиральная структура возникает из-за регулярного роста.

Рисунок 4 – Повторяющаяся спиральная форма раковины

Благодаря своим уникальным свойствам, фракталы нашли применение в различных отраслях науки, техники и искусства. Обычно они используются для упрощения вычислений, которые потребовали бы слишком много времени для реализации традиционными методами.

Компьютерная графика и визуализация.

Фракталы широко используются для создания реалистичных изображений природных объектов. Их алгоритмы помогают воспроизводить сложные формы с минимальными затратами.

– генерация ландшафтов – в компьютерной графике фрактальные алгоритмы используются для моделирования гор, облаков, деревьев и береговых линий,
– создание текстур – в 3D моделировании для создания комплексных поверхностей, например древесной коры, так же могут использоваться фрактальные вычисления.

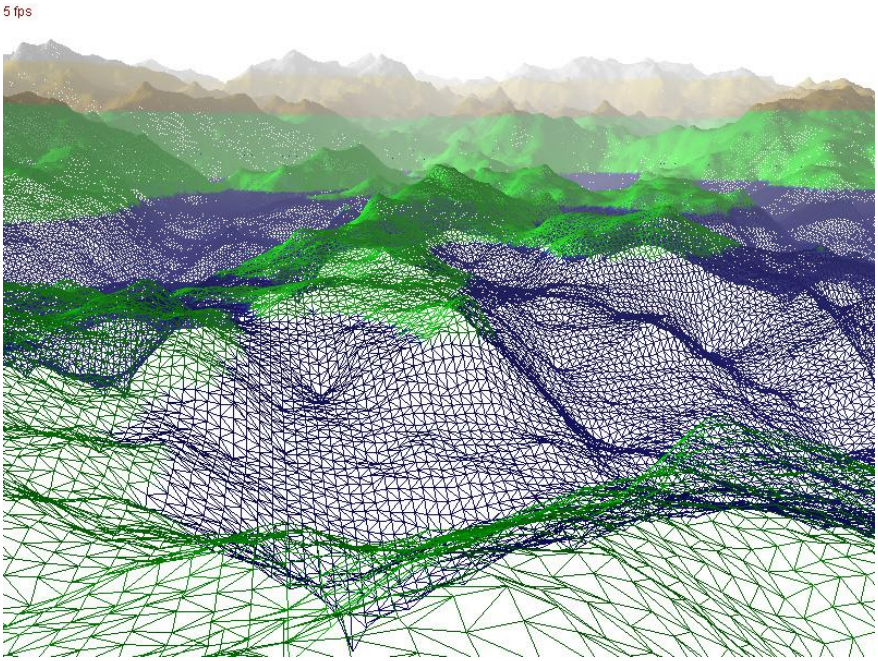

Рисунок 5 – Ландшафт, сгенерированный с помощью фракталов

Обработка изображений и сжатие данных.

Фрактальное сжатие изображений основано на принципе самоподобия. Для уменьшения занимаемого на носителе места, изображение хранится в виде ряда математических преобразований.

– фрактальное сжатие – позволяет значительно уменьшить объем памяти, занимаемый данными, при этом не слишком жертвуя качеством,
– шумоподавление и увеличение разрешения – алгоритмы на основе фракталов помогают очищать изображения от шумов и восстанавливать детали при увеличении масштаба.

Естественные науки.

Фракталы используются для описания сложных систем и явлений, которые не подчиняются традиционной геометрии.

– моделирование потоков – в аэродинамике и гидродинамике фрактальные модели помогают описывать поведение воздушных и водных потоков,

– анализ структуры материалов – многие материалы (горные породы, металлы, биологические ткани) обладают фрактальной структурой, поэтому для ее изучения используются фрактальные алгоритмы.

Биология и медицина.

Фрактальные структуры встречаются и в живых организмах, их применяют при диагностике и моделировании биологических процессов.

– анализ структуры кровеносных сосудов и нервных сетей – помогает в диагностике сердечно-сосудистых заболеваний,
– фрактальная диагностика рака – исследования показывают, что злокачественные опухоли имеют сложную структуру, схожую с фрактальной, что можно использовать для предупреждения и изучения заболевания.

Телекоммуникации и радиосвязь.

Фрактальные структуры используются при проектировании эффективных антенн для передачи и приёма сигналов.

– фрактальные антенны – применяются с недавнего времени, их уникальная форма обеспечивает хорошие широкополосные характеристики при относительно небольшой площади,
– оптимизация сетей связи – фрактальные алгоритмы помогают моделировать сложные структуры распределенных систем.

Рисунок 6 – Фрактальная антенна

Это не все применения фракталов на практике, но в рамках этой работы будет рассмотрено применение фракталов в качестве инструмента для сжатия изображений.

Выводы: фрактальный подход представляет собой удобный инструмент для описания и моделирования сложных структур. Благодаря свойствам самоподобия и рекурсивности, его можно использовать в задачах, где традиционные методы менее эффективны.

На практике фрактальный подход применяется в компьютерной графике, сжатии изображений, обработке сигналов, биомедицине и других областях. Особенно перспективным является фрактальное сжатие, так как оно позволяет уменьшать объем данных без существенной потери качества.

Таким образом, фрактальный подход актуален для решения прикладных задач и может быть полезен в разработке современных технологий обработки информации.

2 Математические основы фрактального сжатия изображений

2.1 Основные принципы

Фрактальное сжатие изображений основано на идее, что любое изображение содержит области, которые можно выразить через математические преобразования других областей. Этот метод отличается от традиционных алгоритмов (JPEG, PNG) тем, что не хранит пиксельные значения напрямую, а представляет изображение в виде набора аффинных преобразований.

Самоподобие изображения.

Главный принцип фрактального сжатия заключается в том, что в любом изображении можно найти участки, схожие по структуре с другими областями, но, чтобы их получить нужно сделать ряд преобразований по отношению к исходной области, таким образом, вместо хранения всей информации о пикселях, достаточно будет хранить информацию об исходных областях и действиях, которые нужно с ними провести. К примеру, если рассмотреть изображение листа папоротника, его фрагменты будут напоминать уменьшенные копии всего листа.

Разбиение изображения на доменные и ранговые блоки.

Для сжатия изображение разбивается на два типа блоков:

– ранговые блоки – небольшие фрагменты, которые необходимо закодировать.
– доменные блоки – более крупные фрагменты, над которыми производятся преобразования, пока они не будут схожи с ранговыми блоками.

Для каждого рангового блока ищется соответствующий ему доменный блок, который можно преобразовать таким образом, чтобы он максимально соответствовал оригинальному фрагменту. Допустим, в фотографии дерева есть множество похожих листьев. Вместо хранения информации обо всех

листьях, алгоритм находит один доменный блок, который можно трансформировать и использовать для кодирования множества ранговых блоков, содержащих в себе изображения листьев.

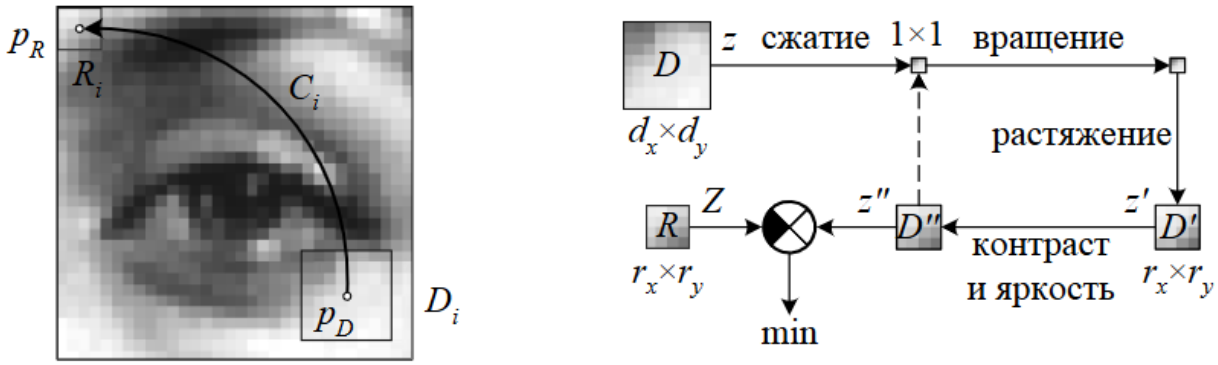

Рисунок 7 – Преобразования доменного блока в ранговый

Аффинные преобразования.

Чтобы доменный блок можно было превратить в соответствующий ранговый, применяются аффинные преобразования:

– масштабирование,

– вращение,

– отражение,

– смещение по координатам,

– изменение яркости и контраста.

В память вместо информации о пикселях, будут записываться эти инструкции.

Коллажная теорема Барнсли.

Этот принцип утверждает, что любое изображение можно представить как совокупность его собственных преобразованных фрагментов. В процессе декодирования происходит многократное применение этих преобразований, что приводит к восстановлению оригинального изображения. Если взять фотографию облака и уменьшить её, а затем многократно

копировать, изменяя масштаб и расположение, можно воссоздать структуру, напоминающую исходное изображение.

Генеративный характер декодирования.

Фрактальное сжатие является разрушающим методом, так как исходные пиксели не сохраняются, а восстанавливаются на основе математической модели. Декодирование осуществляется путем итеративного применения найденных аффинных преобразований, что делает процесс похожим на генерацию изображения. Если запустить алгоритм восстановления фрактального изображения, начав с любого случайного изображения, через несколько итераций оно примет форму оригинала. Это похоже на то, как из небольшого набора инструкций можно "вырастить" сложную структуру

2.2 Математическая модель метода

Допустим, что нам нужно сжать изображение, при помощи фракталов, первым шагом для достижения этого будет разделение изображения на непересекающиеся ранговые блоки R_n.

Простейшим способом будет разбиение на одинаковые прямоугольные области фиксированного размера.

R_{11}	R_{12}		…		R_{1n}
R_{21}					
⋮					
R_{m1}					R_{mn}

Рисунок 8 – Разбиение на одинаковые прямоугольники

Такой подход позволит не хранить информацию о позициях блоков в сжатом файле.

В более эффективных подходах, однако, используется адаптивное разбиение на блоки нефиксированного размера. К примеру, в методе квадродерева, за размерность блока берется половина размера исходного изображение, последующие размеры блоков так же могут уменьшаться вдвое.

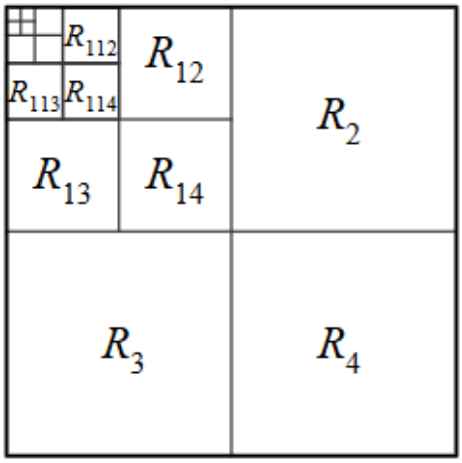

Рисунок 9 – Разбиение методом квадродерева

За уменьшение общего числа блоков придется заплатить хранением информации о позициях и размерах блоков, а также сниженным коэффициентом сжатия. В таком случае нумерацию лучше вести многоуровневым списком индексов.

Вторым шагом выполнения алгоритма будет поиск доменных блоков. Для того чтобы преобразование доменных блоков в ранговые прошло успешно, первые должны быть больше по размеру. Доменные блоки могут перекрывать друг друга и не покрывать полную область изображения. Для того, чтобы поиск занимал адекватное количество времени, размеры искомых блоков и варианты их расположения можно ограничить.

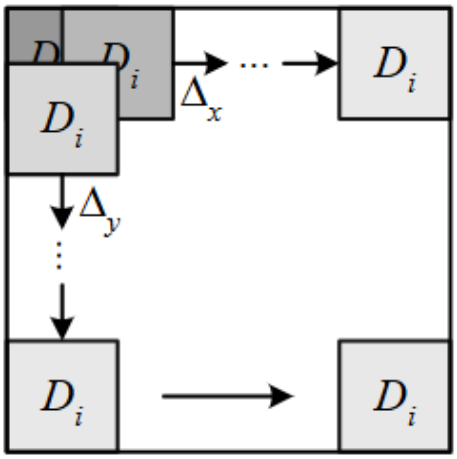

Рисунок 10 – Вариант поиска доменных блоков сверху вниз и слева направо

Хорошего сжатия можно достичь при условии, что удалось найти доменный блок D_i аффинно-оптические преобразования которого близки к ранговому блоку R_i, доменный блок, при этом, может быть большего размера. При неудачном поиске ранговый блок дробится на более мелкие части до тех пор, пока фрактальное сжатие эффективнее простого хранения информации о пикселях.

Сопоставление доменных и ранговых блоков – итерационный процесс, целью которого является приведение размеров блока D к размерам блока R с помощью геометрических преобразований, вращения и масштабирования. Также с помощью оптических преобразований z'=f(D') оттенков пикселей блока D' найти оттенки пикселей z'', приближенных к окраске Z=f(R) пикселей блока R.

Так как имеется вероятность, что доменные или ранговые блоки будут неквадратными, необходимо выполнить последовательность геометрических преобразований:

– перенос центра доменного блока $p_D = [x_D; y_D]$ в начало координат матрицей $T(-p_D)$,

– масштабирование блока до единичного матрицей $M(d_x^{-1}, d_y^{-1})$,

– возможное отражение единичного блока матрицей $M(\pm 1,1)$,

– вращение блока на градусную меру кратную 90 матрицей $R(90^o \cdot \nu)$, где $\nu \in \{0,1,2,3\}$,

– масштабирование блока до размеров $r_x \times r_y$ матрицей $M(r_x,r_y)$,

– перенос полученного блока D' из начала координат в центр рангового блока $p_R = [x_R; y_R]$ матрицей $T(p_R)$.

Результирующая матрица сложного математического преобразования будет иметь вид:

$$C = T(-P_D)\tilde{M}\left(\pm d_x^{-1} d_y^{-1}\right)\tilde{R}(90° \cdot \nu)\tilde{M}\left(r_x,r_y\right)T\left(p_R\right). \quad (1)$$

С помощью этой формулы можно вычислить координаты рангового блока $p' = [x'; y'] \in D' \equiv R$ по координатам соответствующего домена $p = [x; y] \in D$:

$$\tilde{p}' = \tilde{p}C. \quad (2)$$

Или в координатной форме с шестью искомыми коэффициентами матрицы геометрического преобразования C:

$$\begin{cases} x' = c_{11}x + c_{21}y + c_{31}, \; y' = c_{12}x + c_{22}y + c_{32}, \\ c_{11} = \dfrac{r_x}{\pm d_x}\cos(90° \cdot \nu), \; c_{12} = \dfrac{r_y}{\pm d_x}\sin(90° \cdot \nu), \\ c_{21} = \dfrac{-r_x}{d_y}\sin(90° \cdot \nu), \; c_{22} = \dfrac{r_y}{d_y}\cos(90° \cdot \nu), \\ c_{31} = x_R - c_{11}x_D - c_{21}y_D, \; c_{32} = y_R - c_{12}x_D - c_{22}y_D. \end{cases} \quad (3)$$

Обратное преобразование R в D определяется векторным или координатными уравнениями:

$$\tilde{p} = \tilde{p}'^{C^{-1}}, \quad (4)$$

$$\begin{cases} x = \dfrac{c_{22}x' - c_{21}y' + c_{21}c_{32} - c_{22}c_{31}}{|C|}, \\ y = \dfrac{-c_{12}x' + c_{11}y' + c_{12}c_{31} - c_{11}c_{32}}{|C|}, \\ |C| = c_{11}c_{22} - c_{12}c_{21} = \pm\dfrac{r_x r_y}{d_x d_y}. \end{cases} \quad (5)$$

Из-за неравности блоков, полученных в результате преобразований, высока вероятность нецелых координат точек. Принадлежность точки [x; y] пикселю, находящемуся в строке i и столбце j, устанавливается соотношениями:

$$i = \left\lceil \frac{y}{M} \right\rceil + \{y=0\}, \quad (6)$$

$$j = \left\lceil \frac{x}{M} \right\rceil + \{x=0\}. \quad (7)$$

Всего есть восемь вариантов матрицы (1) и уравнений (2), (3), (4), (5), полученных из двух вариантов матрицы отражения $M(\pm 1,1)$ и четырех вариантов матрицы вращения $R(90^o \cdot \nu)$. Все их нужно испытать и выбрать вариант с минимальным отличием блока D' от рангового блока R.

Цвет пикселя (i, j) блока D' можно получить следующими способами.

Простейшее прямое вычисление цвета пикселя домена D, в который попала точка $p_{ij} \equiv [x_j; y_i]$, восстановленная по формулам (4) и (5)

$$z'_{ij} = z_{ij} = f(x_j, y_i). \quad (8)$$

Простое усреднение цветов четырех пикселей в которые попали восстановленные по формуле точки $p_{i-1,j-1}$, $p_{i-1,j}$, $p_{i,j-1}$, p_{ij}.

$$z'_{ij} = \frac{z_{i-1,\ j-1} + z_{i-1,j} + z_{i,j-1} + z_{ij}}{4}. \quad (9)$$

Простое усреднение всех цветов пикселей, полностью или частично попавших в область между точками $p_{i-1,j-1}$, $p_{i-1,j}$, $p_{i,j-1}$, p_{ij}.

$$z'_{ij} = \frac{1}{L}\sum_{l=1}^{L} z_l \,. \tag{10}$$

Наиболее трудоемким будет взвешенное усреднение цветов пикселей домена, попавших в прямоугольник между точками $p_{i-1,j-1}$, $p_{i-1,j}$, $p_{i,j-1}$, p_{ij}.

$$z'_{ij} = \|C\| \sum_{l} S_l z_l \,. \tag{11}$$

В формуле (11) $0<S_l\leq 1$ – накрытая площадь пикселя l. Площадь полностью накрытого пикселя равна единице, а сумма всех накрытых площадей будет равна модулю определителя матрицы C.

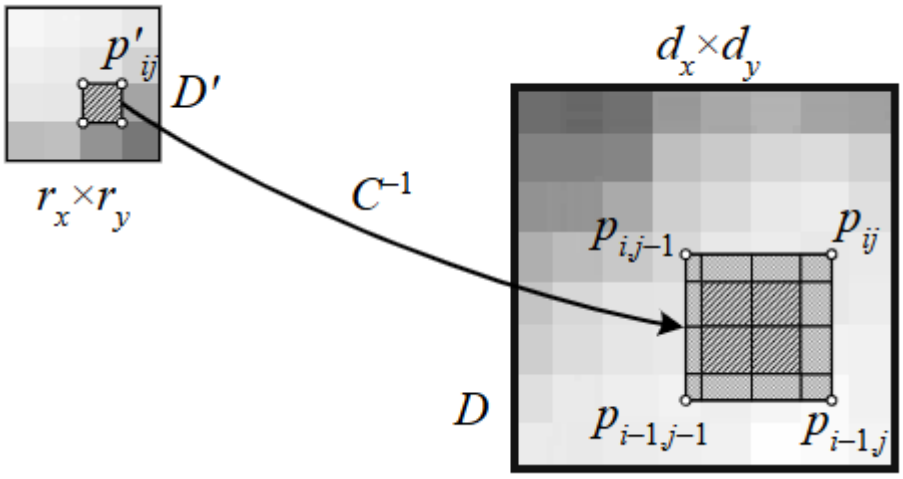

Рисунок 11 – Визуализация покрытой площади пикселей

В результате вычислений получается распределение цветов z'_{ij} всех $r_x r_y$ пикселей в блоке D', стоящем на месте рангового блока R.

Любое неоднородное изображение можно охарактеризовать двумя следующими оптическими параметрами: средний уровень интенсивности – яркость, средний диапазон изменения цвета – контраст. Для оптимального оптического сближения блоков R и D' подвергнем окраску виртуального блока аффинному преобразованию D'→D"

$$z'' = \alpha z' + \beta. \qquad (12)$$

В формуле (12) коэффициент α отвечает за контрастность, а β за яркость пикселей блока. Эти коэффициенты получаются минимизацией дисперсии ошибки доменно-рангового соответствия в расчёте на один пиксель:

$$\sigma^2 = \frac{1}{r_x r_y} \sum_{i,j} \left(z''_{ij} - Z_{ij} \right)^2 \rightarrow \min_{\alpha,\beta}. \qquad (13)$$

Стандартное решение этой задачи при $\Delta = r_x r_y \sum z'^2_{ij} - \left(\sum z'_{ij} \right)^2 \neq 0$:

$$\alpha = \frac{r_x r_y \sum z'_{ij} Z_{ij} - \left(\sum z'_{ij} \right)\left(\sum Z_{ij} \right)}{\Delta}, \qquad (14)$$

$$\beta = \frac{\sum Z_{ij} - \alpha \sum z'_{ij}}{r_x r_y}. \qquad (15)$$

Для $\Delta = 0$ (например, если доменный блок однотонный) решением будет выглядеть следующим образом:

$$\alpha = 0, \qquad (16)$$

$$\beta = \frac{1}{r_x r_y} \sum Z_{ij}. \qquad (17)$$

Суммирование производится по всем пикселям $r_x r_y$ блоков D' и R. Средние арифметические интенсивностей цветов сравниваемых блоков обозначаются следующим образом:

$$\overline{z'} = \frac{1}{r_x r_y} \sum z'_{ij}, \tag{18}$$

$$\overline{Z} = \frac{1}{r_x r_y} \sum Z_{ij}. \tag{19}$$

Получается альтернативная форма записи для формул (14) и (15) оптических коэффициентов:

$$\alpha = \frac{\sum (z'_{ij} - \overline{z'})(Z_{ij} - \overline{\overline{Z}})}{\sum (z'_{ij} - \overline{\overline{z'}})^2}, \tag{20}$$

$$\beta = \overline{Z} - \alpha \overline{\overline{z'}}. \tag{21}$$

Оптическое преобразование доменного блока D' примет вид:

$$z'' = \alpha(z' - \overline{z'}) + \overline{Z}. \tag{22}$$

Подставляя оптические преобразования из формул с (14) по (17) включительно в формулы (12) и (13), получим численное значение дисперсии σ^2 для пары блоков {D, R} и одной из восьми матриц преобразования C.

$$\sigma \le \varepsilon \cdot 2^K. \tag{23}$$

Если выполняется условие (23), где ε – относительный порог доменно-рангового соответствия, в файле сохраняются параметры аффинно-

оптического преобразования пары блоков {D, R} и осуществляется переход в начало первого шага кодирования уже следующего рангового блока.

Если при всех восьми преобразованиях доменно-ранговое соответствие (23) не достигается, запоминаются параметры лучшей реализации со значениями σ^2_{min} и производится переход ко второму шагу – испытанию следующего доменного блока D из множеств допустимых блоков. В отсутствие успеха поиска, ранговый блок R дробится на более мелкие части и убирается из списка необработанных. Очевидно, что повторять эту процедуру бесконечно не получится, иначе число параметров преобразования будет так велико, что алгоритм потеряет в компрессии. Следовательно, в каких-то частях изображения возможны потери. При достижении минимальных размеров рангового блока R, которому не нашлось доменного блока, в выходной файл записываются цветовые коды пикселей рангового блока либо параметры наиболее близкой пары {D, R} со значением $\sigma^2_{min.}$

Когда доступные для обработки ранговые блоки кончаются, алгоритм заканчивает свою работу, на диск записывается файл с сохраненными в нем параметрами доменно-рангового соответствия.

Эффективность сжатия оценивается соотношением объёма исходного файла в формате BMP и полученного закодированного файла. В связи с большим количеством ранговых блоков, чтобы повысить коэффициент сжатия, в файл необходимо сохранять минимум данных, по которым можно восстановить расположение ранговых и отведенных им доменных блоков, коэффициенты оптических и аффинных преобразований, а в заголовке – размеры исходного файла, его тип (цветное изображение ил в градациях серого), разрядность кодирования цвета, параметры выбора доменных и ранговых блоков и т. д.

При создании рангового блока методом квадродерева достаточно хранить лишь номер, по которому программа декодирования восстановит размеры r_x, r_y, а также положение центра p_R. Аналогичным образом при переборе доменов достаточно запомнить лишь их номер для восстановления

параметров d_x, d_y и p_D. Всё, что нужно сохранить для вычисления коэффициентов с c_{11} по c_{32} в формулах (1) – (5), это знак числа ±1 в матрице отражения $M(\pm 1, 1)$ и значение ν в матрице вращения $R(90^o \cdot \nu)$, на что требуется всего 3 бита. Также, для каждой доменно-ранговой пары нужно запомнить параметры преобразования контрастности α и яркости β.

Еще большего сжатия можно достичь, если архивировать известными методами полученный текстовый файл параметров фрактальной компрессии. В результате всей работы можно достичь коэффициента сжатия в 20000.

2.3 Декодирование изображения

Процесс декодирования заключается в итеративном применении «линз», а именно геометрических и оптических преобразований, для закрашивания ранговых блоков цветами z''_{ij} в особом массиве данных размером с холст из соответствующих доменных блоков. Так как каждый ранговый блок меньше своего доменного, несколько пикселей доменного блока D могут преобразоваться в один и тот же пиксель блока R, что слишком затратно для алгоритма декомпрессии.

Чтобы исключить лишние вычисления во время декодирования изображения более эффективным будет обратный порядок преобразований – от рангового к доменному.

Для каждой площадки рангового пикселя $p'_{ij} \in R$ с помощью преобразований (4) и (5) вычисляется нужное количество точек домена $p_{\nu\mu} \in D$ в соответствии с использованным при компрессии вариантом вычисления цвета по (8) – (11).

С холста изображения считываются цвета найденных точек $z_{\nu\mu} = f(p_{\nu\mu})$ и по формулам (8) – (11) вычисляется цвет рангового пикселя z'_{ij}.

Выполняется оптическое преобразование по формуле (12), полученное значение цвета z''_{ij} присваивается пикселю p'_{ij}.

Когда восстановленные ранговые блоки покроют холст изображение, оно выводится на экран и алгоритм переходит к следующей итерации.

Особенностью фрактального декодирования является несимметричность по отношению к сжатию, это значит, что вычислительная сложность декодирования сильно меньше сложности кодирования, поэтому при разработке эффективных алгоритмов фрактального сжатия, основные силы сосредотачивают именно на оптимизации процесса кодирования.

Выводы: в этой части были рассмотрены математические принципы фрактального сжатия изображений. Метод основан на поиске самоподобных участков изображения и замене их набором аффинных и оптических преобразований. Были описаны этапы разбиения изображения на блоки, сопоставления доменных и ранговых областей, вычисления трансформаций и критериев соответствия.

3 Программная реализация

3.1 Описание структуры программы

Для написания программной части был использован язык Python и совместимые с ним методы визуализации процесса.

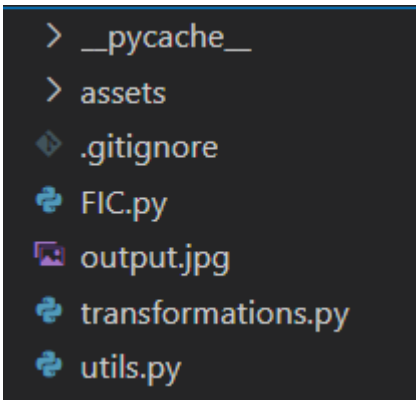

Рисунок 12 – Структура программы

Структура программы состоит из следующих компонентов:

– папка assets содержит изображение, на котором будет проверяться алгоритм,

– FIC.py. Главный исполняемый файл, содержащий основную логику алгоритма,

– utils.py содержит вспомогательные функции для обработки изображения и деления его на блоки,

– transformations,py реализует геометрические преобразования, используемые в алгоритме фрактального сжатия.

В главном исполняемом файле реализованы следующие функции:

Compress_image. Является ядром фрактального кодирования. Основная задача функции – поиск набора трансформаций, которые с определенной точностью могут воссоздать исходные части изображения, на основе

самоподобных фрагментов. Сама функция сохраняет не изображение, а только инструкции по его восстановлению из набора аффинных преобразований. Внутри функции производится формирование базы данных, содержащей всех возможных кандидатов на роль нужных доменных блоков. Так же оригинальное изображение разбивается на сетку ранговых блоков.

```python
def compress_image(img, source_size, destination_size, step):

    transformed_blocks = generate_all_transformed_blocks(img, source_size, destination_size, step)
    i_block_count, j_block_count = img.shape[0] // destination_size, img.shape[1] // destination_size

    transformations = find_best_transformations(
        img, destination_size, transformed_blocks, i_block_count, j_block_count, step
    )

    return transformations
```

Рисунок 13 – Реализация модуля compress_image

Generate_all_transformed_blocks. В функции производится перебор доменных блоков с определенным шагом, далее над ними производятся геометрические преобразования: масштабирование до размеров рангового блока, повороты. На выходе получается список блоков и их метаданных.

```python
def generate_all_transformed_blocks(img, source_size, destination_size, step):

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

    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
            source_block = transf.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, transf.apply_transformation(source_block, direction, angle))
                )
    return transformed_blocks
```

Рисунок 14 – Реализация модуля generate_all_transformed_blocks

Find_best_transformations. Эта функция отвечает за поиск оптимальных преобразований для каждого рангового блока, является одним из ключевых этапов, именно на нем, при помощи аффинных преобразований сопоставляются доменные и ранговые блоки для достижения наименьшего значения среднеквадратичной ошибки. По выполнении алгоритма, для каждого блока будет получена следующая информация: координаты источника, тип геометрической трансформации, яркость, контраст. В конце будет возвращена полная таблица значений.

```
def find_best_transformations(img, destination_size, transformed_blocks, i_count, j_count, step):

    transformations = []

    for i in range(i_count):
        transformations.append([None] * j_count)

        for j in range(j_count):
            destination_block = img[
                i * destination_size : (i + 1) * destination_size,
                j * destination_size : (j + 1) * destination_size,
            ]
            best_transformation = find_best_transformation(transformed_blocks, destination_block)
            transformations[i][j] = best_transformation

    return transformations
```

Рисунок 15 – Реализация модуля find_best_transformations

Estimate_contrast_and_brightness. В этой функции оцениваются значения яркости и контраста, которых нужно достичь доменному блоку чтобы приблизиться к ранговому. На вход модуля идут ранговый и доменные блоки, на выходе получаются значения.

```python
def estimate_contrast_and_brightness(source_block, destination_block):

    source_mean = np.mean(source_block)
    destination_mean = np.mean(destination_block)

    source_variance = np.var(source_block)
    covar = np.cov(source_block.flatten(), destination_block.flatten())[0][1]

    contrast = covar / source_variance if source_variance > 0 else 1.0
    brightness = destination_mean - (contrast * source_mean)

    return contrast, brightness
```

Рисунок 16 – Реализация модуля estimate_contrast_and_brightness

Find_best_transformation. В этом модуле будет вычисляться лучшая трансформация, поиском наименьшей ошибки. Из списка transformed_blocks перебираются все варианты блоков кандидатов. Из прошлой описанной функции к блоку кандидату применяются контраст и яркость, а потом вычисляется среднеквадратичная ошибка. Чем ошибка меньше, тем больше совпадение. В ходе работы модуля находится блок-кандидат с наименьшим отклонением от рангового блока.

```python
def find_best_transformation(transformed_blocks, destination_block):

    min_distance = float("inf")
    best_transformation = None

    for k, l, direction, angle, source_block in transformed_blocks:
        contrast, brightness = estimate_contrast_and_brightness(source_block, destination_block)
        source_block = contrast * source_block + brightness
        distance = np.sum(np.square(destination_block - source_block))

        if distance < min_distance:
            min_distance = distance
            best_transformation = (k, l, direction, angle, contrast, brightness)

    return best_transformation
```

Рисунок 17 – Реализация модуля find_best_transformation

Decompress_image. Функция реализует декодирующую часть алгоритма фрактального сжатия. На основе раннее сохраненных инструкций восстанавливается изображение путем итеративного применения фрактальных операторов. Трансформации применяются ко всем блокам итеративно через функцию apply_transformations_to_blocks. После указанного количества итераций, список iterations содержит историю процесса восстановления. Возвращается список изображений: от первого приближения до финального результата. Обычно хватает небольшого числа итераций, в противном случае можно столкнуться со случаем, когда изменения в приближениях будут минимальны.

```
def decompress_image(transformations, source_size, destination_size, step, num_iterations=8):

    factor = source_size // destination_size
    height, width = len(transformations) * destination_size, len(transformations[0]) * destination_size
    iterations = [initialize_random_image(height, width)]
    current_img = np.zeros((height, width))

    for iteration in range(num_iterations):
        current_img = apply_transformations_to_blocks(
            transformations, current_img, iterations, source_size, destination_size, step, factor
        )
        iterations.append(current_img)
        current_img = np.zeros((height, width))
    return iterations
```

Рисунок 18 – Реализация модуля decompress_image

Apply_transformations_to_blocks. Функция отвечает за реализацию одной итерации фрактальной декомпрессии. Она обходит все ранговые блоки на изображении и применяет к ним соответствующие аффинные преобразования, ранее вычисленные и сохраненные в списке transformations в процессе компрессии. Доменные блоки уменьшаются, степень уменьшения зависит от параметра factor, который задается в начале пользователем.

```python
def apply_transformations_to_blocks(
    transformations, current_img, iterations, source_size, destination_size, step, factor
):
    for i, row in enumerate(transformations):
        for j, t in enumerate(row):
            if t is not None:
                k, l, flip, angle, contrast, brightness = t
                source_block = transf.reduce(
                    iterations[-1][k * step : k * step + source_size, l * step : l * step + source_size],
                    factor,
                )
                destination_block = transf.apply_transformation(
                    source_block, flip, angle, contrast, brightness
                )
                current_img[
                    i * destination_size : (i + 1) * destination_size,
                    j * destination_size : (j + 1) * destination_size,
                ] = destination_block

    return current_img
```

Рисунок 19 – Реализация модуля apply_transformations_to_blocks

Помимо описанных функций, в программе имеются вспомогательные, выполняющие второстепенные функции. В файле utils.py содержится функция compress_with_jpeg, реализующая JPEG подобное сжатие, для дальнейшего сравнения, а также функции plot_decompressed_images и plot_psnr_comparation, отвечающие за вывод результатов в отдельные окна.

```python
def plot_decompressed_images(decompressed_images, rows, cols):
    fig, axes = plt.subplots(rows, cols, figsize=(8, 5))

    for i in range(rows):
        for j in range(cols):
            index = i * cols + j
            if index < len(decompressed_images):
                axes[i, j].imshow(decompressed_images[index], cmap="gray")
                axes[i, j].axis("off")
                axes[i, j].set_title(f"Iteration {index+1}")

    plt.tight_layout()
    plt.show()
```

Рисунок 20 – Реализация модуля plot_decompressed_images

```python
def plot_psnr_comparation(fractal_img, fractal_psnr, jpeg_img, jpeg_psnr):
    fig, axes = plt.subplots(1, 2, figsize=(6, 4))
    axes[0].imshow(fractal_img, cmap="gray")
    axes[0].set_title(f"Fractal PSNR: {fractal_psnr:.2f} dB")
    axes[0].axis("off")

    axes[1].imshow(jpeg_img, cmap="gray")
    axes[1].set_title(f"JPEG PSNR: {jpeg_psnr:.2f} dB")
    axes[1].axis("off")

    plt.tight_layout()
    plt.show()
```

Рисунок 21 – Реализация модуля plot_psnr_comparation

В итоге получается консольная программа, выполняющая фрактальное сжатие выбранного изображения, выводит итерации декомпрессии и сравнивает результат с алгоритмом JPEG при помощи коэффициента PSNR (Peak Signal-to-Noise Ratio).

3.2 Тестирование программы

Для того чтобы проверить работоспособность программы и корректность алгоритма, были выбраны три чёрно-белых изображения с разрешением 256 на 256 точек. В коде задаются параметры преобразования.

```python
# Parameters
file_path = "assets/lena.gif"
reduction_factor = 2
source_size = 8
destination_size = 4
step = 4
```

Рисунок 22 – Входные параметры

File_path указывает на расположение преобразуемого файла, reduction_factor отвечает за степень уменьшения выходного изображения, значение 2 означает, что выходное изображение будет уменьшено по двум осям в два раза, итоговый размер будет 128 на 128 точек. Destination_size определяет размеры ранговых блоков, значение 4 означает, что ранговые блоки будут размерами 4 на 4 точки, суммарное количество ранговых блоков в таком случае $32^2 = 1024$. Destination_size и step отвечают за размер доменных блоков и шаг с которым они берутся, итоговое число доменных блоков размером 8 на 8, берущихся с шагом 4, равно 961. Над доменными блоками могут производится операции поворота (4 направления) и отражения (2 направления), итоговое число преобразованных доменных блоков $961 \cdot 8 = 7688$. Общее число сравнений 1024 ранговых блоков и 7688 равно 7872512. Использование изображений низкого разрешения – вынужденная мера, так как при повышении этой характеристики, вычислительная сложность алгоритма повышается. Например, при описанных выше параметрах, время выполнения алгоритма могло достигать 17 минут.

Ниже приведены результаты выполнения алгоритма для разных входных изображений:

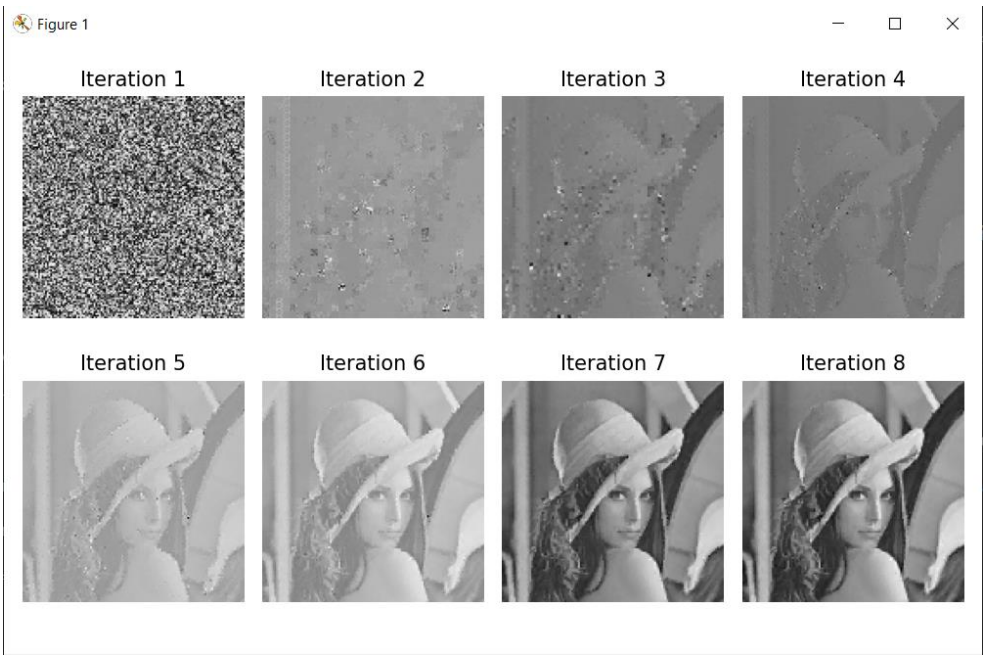

Рисунок 23 – Прогресс алгоритма декомпрессии первого изображения

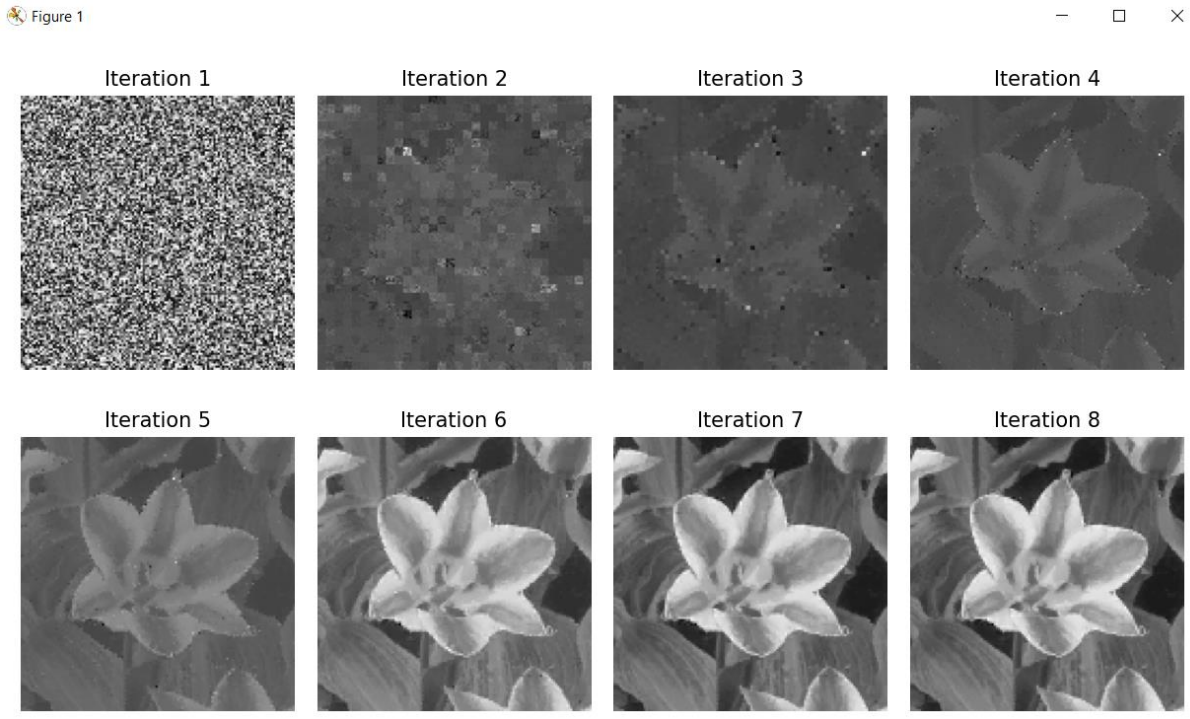

Рисунок 24 – Прогресс алгоритма декомпрессии второго изображения

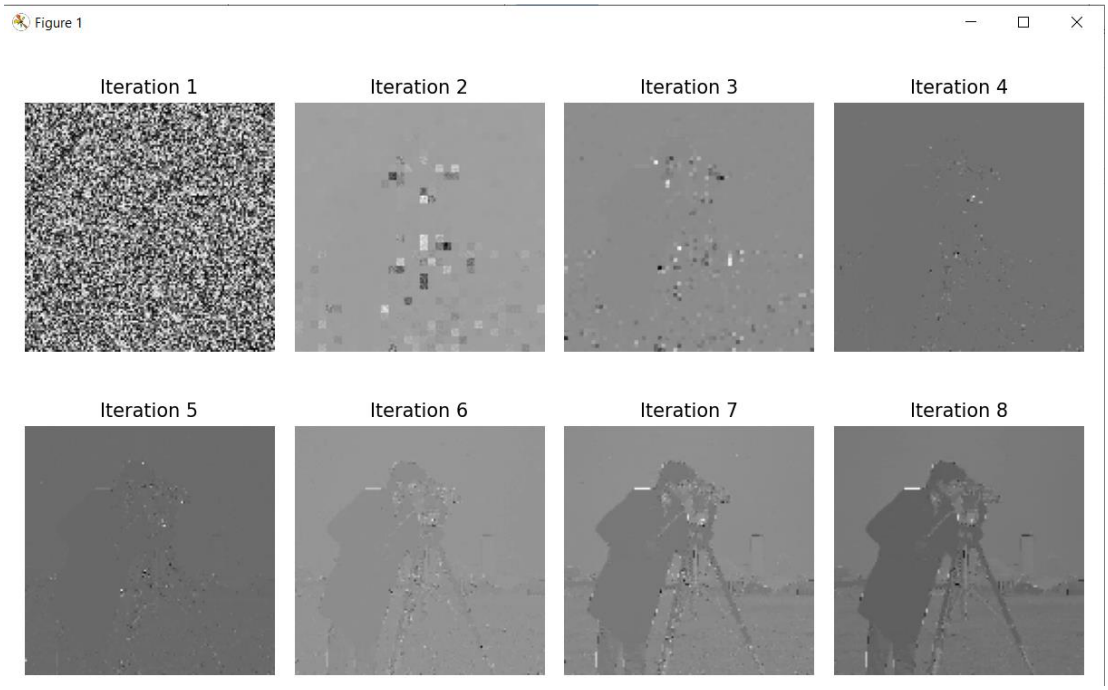

Рисунок 25 – Прогресс алгоритма декомпрессии третьего изображения

В результате выполнения программа выводит указанное количество результатов итераций процесса декомпрессии, в ходе которого можно наблюдать снижение уровня шума и постепенное приближение к исходному изображению. Как и задумывалось, первая итерация представляет собой случайное изображение. У результирующего изображения можно наблюдать небольшие артефакты на границах ранговых блоков, эффект становится заметнее если увеличить размеры ранговых блоков. В ходе тестирования было замечено очень низкое качество выходного изображения в третьем опыте, возможная причина может быть в том, что на изображении есть большие участки с шумом, для которых алгоритм не смог подобрать самоподобные.

3.3 Анализ эффективности алгоритма

Для проверки эффективности описанного алгоритма, проведем его сравнение с устоявшимся алгоритмом JPEG. Для этого рассчитаем

коэффициент PSNR для обоих алгоритмов и сверим итоговые размеры выходных файлов.

Формула PSNR выглядит следующим образом:

$$\mathrm{PSNR}=20\cdot\log_{10}\left(\frac{\mathrm{MAX_I}}{\sqrt{\mathrm{MSE}}}\right). \quad (24)$$

MSE – среднее значение квадратов разностей между пикселям исходного и сжатого изображения. PSNR измеряется в децибелах, значения выше 40 дБ указывают на изображение без видимых искажений, в диапазоне от 30 до 40дБ получаются у изображений с незначительными искажениями. Ниже 30 дБ ошибки будут видны невооруженным взглядом.

Рисунок 26 – Сравнение результатов с JPEG для первого изображения

После декомпрессии первого изображения получаем результат PSNR в 30,94 дБ у фрактального метода и 42,06 дБ у JPEG. Размер выходного изображения у метода JPEG 13 кБ и 7,3 кБ у фрактального подхода.

Хоть и было достигнуто заметное уменьшение размера результирующего изображения, по качеству фрактальный метод уступает, и изображения наблюдается недостаток контраста и помехи на границах контуров.

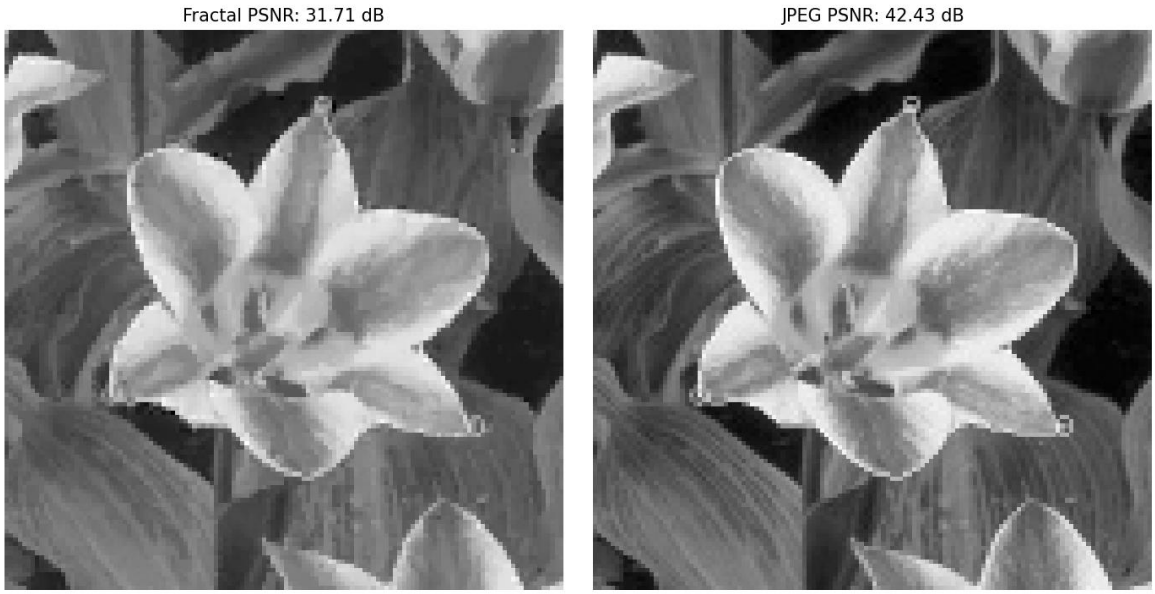

Рисунок 27 – Сравнение результатов с JPEG для второго изображения

В результате обработки второго изображения было достигнуто чуть более высокое значение PSNR – 31,89. Размер выходного изображения после фрактального подхода 8,65 кБ, а после JPEG 14кБ. При детальном рассмотрении можно обратить внимание на размытые текстуры лепестков в левом изображении, но без тщательного осмотра сложно выявить недостатки фрактального алгоритма.

Рисунок 28 – Сравнение результатов с JPEG для третьего изображения

Из-за специфики третьего изображения, фрактальный подход оказался крайне неэффективным, хоть размер итогового файла и оказался ниже, по сравнению с JPEG, качество слишком сильно уступает для практического применения.

Выводы: в этом разделе была рассмотрена структура программы, в которой был реализован алгоритм фрактального сжатия изображений, было проведено тестирование и оценка эффективности, показавшая, что алгоритм не универсален, так как слишком неэффективен в специфических сценариях.

Заключение

В рамках выпускной квалификационной работы были рассмотрены основные моменты, связанные с фрактальным сжатием изображений.

В первом разделе были рассмотрены основные положения о природе фракталов, о том, как принципы, связанные с ними, использует в современных областях науки и жизни. Отдельное внимание было уделено фрактальному методу компрессии цифровых изображений.

Во втором разделе была описана математическая модель метода, рассмотрены основные принципы, по которым работает алгоритм, отдельно разобраны процессы компрессии и декомпрессии, обозначены особенности методы, которые позже будут использованы для разработки программной реализации.

В третьем разделе была представлена программная реализация алгоритма фрактального сжатия данных, описаны основные компоненты программы, проведено тестирование и оценка эффективности в сравнении с актуальным и распространенным методом JPEG.

В результате работы была получена функционирующая реализация алгоритма фрактального сжатия, на основе испытаний можно было сделать вывод, что метод фрактального сжатия является эффективным в специфических ситуациях, но вычислительная сложность и непостоянство качества результирующих изображений показывают, что метод не может стать массово распространенным, пока в процессе компрессии не будет получена высокая производительность.

Список используемой литературы и используемых источников

1. Аксёнова, Н. А. Компьютерная графика : учебно-методическое пособие / Н. А. Аксёнова, А. В. Воруев, О. М. Демиденко. – Гомель : ГГУ имени Ф. Скорины, 2023. – 130 с.

2. Даутова, И. С. Мультимедийные технологии : учебное пособие / И. С. Даутова, С. .. Кошевая, М. В. Симонова. – Краснодар : КубГТУ, 2024. – 259 с.

3. Катунин, Г. П. Основы мультимедийных технологий / Г. П. Катунин. – 3-е изд., стер. – Санкт-Петербург : Лань, 2023. – 784 с.

4. Кордонская, И. Б. Инженерная и компьютерная графика : учебник / И. Б. Кордонская, Е. А. Богданова. – Самара : ПГУТИ, 2020. – 264 с.

5. Королькова, И. А. Растровая компьютерная графика : учебное пособие / И. А. Королькова, С. А. Зайцев, Ф. В. Киселев. – Москва : МУИВ, 2024. – 318 с.

6. Лаптева, А. В. Теория информации : учебное пособие / А. В. Лаптева. – Екатеринбург : УрГЭУ, 2023. – 98 с.

7. Медведев, М. В. Цифровая обработка изображений : учебно-методическое пособие / М. В. Медведев. – Казань : КНИТУ-КАИ, 2020. – 100 с.

8. Мицук, С. В. Информационные технологии в процессе подготовки современного специалиста : сборник научных трудов / С.В. Мицук. – Липецк : Липецкий ГПУ, 2022. – 255 с.

9. Никулин, Е. А. Компьютерная графика. Фракталы : Учебное пособие для вузов / Е. А. Никулин. – 2-е изд., стер. – Санкт-Петербург : Лань, 2021. – 100 с

10. Осокин, А. Н. Теория информации : учебное пособие / А. Н. Осокин, А. Н. Мальчуков. – Томск : ТПУ, 2014. - 2006 с

11. Пантелеев, Е. Р. Алгоритмы сжатия данных без потерь / Е. Р. Пантелеев, А. Л. Алыкова. – 3-е изд., стер. – Санкт-Петербург : Лань, 2023. – 172 с.

12. Старовойтов, В. В. Получение и обработка изображений на ЭВМ : учебно-методическое пособие / В. В. Старовойтов, Ю. И. Голуб. – Минск : БНТУ, 2018. – 204 с.

13. Шилина, О. И. Цифровая обработка изображений : учебно-методическое пособие / О. И. Шилина, Д. А. Наумов, Е. А. Уварова. – Рязань : РГРТУ, 2021. – 265 с.

14. Шихеева, В. В. Фрактальная геометрия. Детерминированные фракталы: учебник / В. В. Шихеева. – Москва : МИСИС, 2019. – 270 с.

15. Яковлев, Б. С. Цифровые технологии обработки изобразительной информации : учебник / Б. С. Яковлев, Е. Н. Пальчун. – Тула : ТулГУ, 2023. – 236 с.

16. Falconer K. Fractal Geometry: Mathematical Foundations and Applications / K. Falconer. – New York : Wiley, 1990. – 288 с.

17. Fisher Y. Fractal Image Compression: Theory and Application. – Springer Science & Business Media, 2012. – 281 p.

18. Miano, J.M. Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP / J.M. Miano. – Boston : Addison-Wesley Professional, 1999. – 288 с.

19. Pennebaker,, W.B., JPEG: Still Image Data Compression Standard (Digital Multimedia Standards S) / W.B., Pennebaker,, J. L. Mitchell. – 1993 Edition. – NYC : Springer, 1993. – 656 с.

20. Salomon D. Data Compression: The Complete Reference. – 4th ed. – Springer, 2007. – 1092 p.