ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ФРАКТАЛЬНОГО СЖАТИЯ ИЗОБРАЖЕНИЙ

Выпускная квалификационная работа

обучающейся по направлению подготовки 02.03.01

Математика и компьютерные науки

очной формы обучения, группы 07001303

Глыга Ксении Геннадиевны

Научный руководитель
доцент, Бурданова Е.В.

БЕЛГОРОД 2017

СОДЕРЖАНИЕ

ВВЕДЕНИЕ……………………………………………………………………….…3

1. АНАЛИЗ ПРОБЛЕМЫ И ПОСТАНОВКА ЗАДАЧИ…………………….…7

1.1.Линейные методы встраивания информации в изображение……....10

1.2.Обзор методов фрактального сжатия изображения...………….…....12

1.3.Метод фрактального сжатия изображений………………..………16

1.4. Примеры фрактального изображения…………...………….……..…18

1.5.Постановка задачи………..…………………………………..……..…..21

2. РАЗРАБОТКА МАТЕМАТИЧЕСКОГО АППАРАТА И АЛГОРИТМОВ…………………………………………………………..…...23

2.1.Математические основы фрактального сжатия изображения……...…23

2.2.Разработка алгоритма компрессии……………………………………...28

2.3.Разработка алгоритма декомпрессии……………………………...35

3. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ…………………....36

3.1.Обоснование и выбор среды разработки……………………………36

3.2.Разработка структуры приложения…………………………….…37

3.3.Разработка формата файла…………………………………...……..…42

3.4.Разработка пользовательского интерфейса……………………………43

4. ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ………………...46

ЗАКЛЮЧЕНИЕ……………………………………………………………….…49

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ……………………………..50

ПРИЛОЖЕНИЯ…………………………………………………………….……52

ВВЕДЕНИЕ

Задача защиты информации от несанкционированного доступа решалась во все времена на протяжении истории человечества. Уже в древности выделилось два основных направления решения этой задачи, существующие и сегодня: криптография и стеганография. Целью криптографии является скрытие содержимого сообщений за счет их шифрования. В отличие от этого, при стеганографии скрывается сам факт существования тайного сообщения.

Рождение фрактальной геометрии обычно связывают с выходом в 1977 году книги Б. Мандельброта "Фрактальная геометрия природы". Одна из основных идей книги заключалась в том, что средствами традиционной геометрии (то есть используя линии и поверхности), чрезвычайно сложно представить природные объекты. Фрактальная геометрия задает их очень просто.

Если построение изображений с помощью фрактальной математики можно назвать прямой задачей, то построение по изображению IFS - это обратная задача. Довольно долго она считалась неразрешимой, однако Барнсли, используя Collage Theorem, построил соответствующий алгоритм. (В 1990 и 1991 годах эта идея была защищена патентами.) Если коэффициенты занимают меньше места, чем исходное изображение, то алгоритм является алгоритмом архивации.

Первая статья об успехах Барнсли в области компрессии появилась в журнале BYTE в январе 1988 года. В ней не описывалось решение обратной задачи, но приводилось несколько изображений, сжатых с коэффициентом 1:10000, что было совершенно ошеломительно. Но практически сразу было отмечено, что несмотря на броские названия ("Темный лес", "Побережье Монтере", "Поле подсолнухов") изображения в действительности имели искусственную природу. Это, вызвало массу скептических замечаний,

подогреваемых еще и заявлением Барнсли о том, что "среднее изображение требует для сжатия порядка 100 часов работы на мощной двухпроцессорной рабочей станции, причем с участием человека".

Отношение к новому методу изменилось в 1992 году, когда Арнауд Джеквин, один из сотрудников Барнсли, при защите диссертации описал практический алгоритм и опубликовал его. Этот алгоритм был крайне медленным и не претендовал на компрессию в 10000 раз (полноцветное 24-разрядное изображение с его помощью могло быть сжато без существенных потерь с коэффициентом 1:8 - 1:50); но его несомненным достоинством было то, что вмешательство человека удалось полностью исключить. Сегодня все известные программы фрактальной компрессии базируются на алгоритме Джеквина. В 1993 году вышел первый коммерческий продукт компании Iterated Systems. Ему было посвящено достаточно много публикаций, но о коммерческом успехе речь не шла, продукт был достаточно "сырой", компания не предпринимала никаких рекламных шагов, и приобрести программу было тяжело.

Слово «стеганография» имеет греческие корни и буквально означает «тайнопись». Исторически это направление появилось первым, но затем во многом было вытеснено криптографией. Тайнопись осуществляется самыми различными способами. Общей чертой этих способов является то, что скрываемое сообщение встраивается в некоторый не привлекающий внимание объект (стеганографический контейнер). Затем этот объект открыто транспортируется адресату. При криптографии наличие шифрованного сообщения само по себе привлекает внимание противников, при стеганографии же наличие скрытой связи остается незаметным.

В качестве стеганографического контейнера может быть использован практически любой предмет. В прошлом веке широко использовались так называемые симпатические чернила, невидимые при обычных условиях. Скрытое сообщение размещали в определенные буквы невинных словосочетаний, передавали при помощи внесения в текст незначительных

стилистических, орфографических или пунктуационных погрешностей. С изобретением фотографии появилась технология микрофотоснимков.

Во время Второй мировой войны правительством США придавалось большое значение борьбе против тайных методов передачи информации. Были введены определенные ограничения на почтовые отправления. Так, не принимались письма и телеграммы, содержащие кроссворды, ходы шахматных партий, поручения о вручении цветов с указанием времени и их вида; у пересылаемых часов переводились стрелки. Был привлечен многочисленный отряд цензоров, которые занимались даже перефразированием телеграмм без изменения их смысла.

Скрытие информации перечисленными методами возможно лишь благодаря тому, что противнику неизвестен метод скрытия. Между тем, более надежной была бы система, которая выполняла бы свои функции даже при полной информированности противника о ее структуре и алгоритмах функционирования. Вся секретность системы защиты передаваемой сведений должна заключаться в ключе, то есть в предварительно разделенном между адресатами фрагменте информации.

Развитие средств вычислительной техники в последнее десятилетие дало новый толчок для развития компьютерной стеганографии. Появилось много новых областей применения. Сообщения встраивают теперь в цифровые данные, как правило, имеющие аналоговую природу. Это – речь, аудиозаписи, изображения, видео. Известны также методы встраивания информации в текстовые файлы и в исполняемые файлы программ.

Можно выделить две причины популярности исследований в области стеганографии в настоящее время: законодательное ограничение на использование криптографических средств в некоторых странах мира и появление проблемы защиты прав собственности на информацию, представленную в цифровом виде. Первая причина повлекла за собой большое количество исследований в духе классической стеганографии (то

есть скрытия факта передачи информации), вторая – еще более многочисленные работы в области так называемых водяных знаков.

Цифровой водяной знак (ЦВЗ) – специальная метка, незаметно внедряемая в изображение или другой сигнал с целью тем или иным образом контролировать его использование [2]. В выпускной квалификационной работе будет реализован метод встраивания цифровых водяных знаков в изображении с помощью фрактального сжатия.

Главной целью выпускной квалификационной работы является рассмотрение и изучение фрактального сжатия изображений.

Для решения поставленной цели необходимо в выпускной квалификационной работе изучить следующие задачи:

- выявление проблемы и постановка задачи;
- разработка математического аппарата и алгоритмов;
- разработка программного обеспечения;
- тестирование программного обеспечения.

В первой главе рассказывается о линейных методах встраивания информации в изображение, обзор методов фрактального сжатия изображения, метод фрактального сжатия изображения и постановка задачи.

Во второй главе происходит разработка математического аппарата и алгоритмов, разработка алгоритма компрессии и декомпрессии.

В третье главе идет разработка программного обеспечения, обоснование и выбор среды разработки, разработка структуры приложения, разработка пользовательского интерфейса.

В четвертой главе идет тестирование программного обеспечения.

В выпускной квалификационной работе содержится 51 страниц без приложения, 15 рисунков , 3 таблицы, 14 формул. В процессе создания было использовано 11 литературных источников.

1. АНАЛИЗ ПРОБЛЕМЫ И ПОСТАНОВКА ЗАДАЧИ

Фрактальное сжатие изображений — это алгоритм сжатия изображений с потерями, основанный на применении систем итерируемых функций (IFS, как правило, являющимися аффинными преобразованиями) к изображениям. Данный алгоритм известен тем, что в некоторых случаях позволяет получить очень высокие коэффициенты сжатия (лучшие примеры — до 1000 раз при приемлемом визуальном качестве) для реальных фотографий природных объектов, что недоступно для других алгоритмов сжатия изображений в принципе. Из-за сложной ситуации с патентованием широкого распространения алгоритм не получил.

Основа метода фрактального кодирования — это обнаружение самоподобных участков в изображении. Впервые возможность применения теории систем итерируемых функций (англ. Iterated Function System, сокр. IFS) к проблеме сжатия изображения была исследована Майклом Барнсли и Аланом Слоуном. Они запатентовали свою идею в 1990 и 1991 гг (U.S. Patent 5 065 447). А. Жакен (фр. Arnaud Jacquin) представил метод фрактального кодирования, в котором используются системы доменных и ранговых блоков изображения (англ. domain and range subimage blocks), блоков квадратной формы, покрывающих все изображение. Этот подход стал основой для большинства методов фрактального кодирования, применяемых сегодня. Он был усовершенствован Ювалом Фишером (англ. Yuval Fisher) и рядом других исследователей [4].

В соответствии с данным методом изображение разбивается на множество неперекрывающихся ранговых подизображений (англ. range subimages) и определяется множество перекрывающихся доменных подизображений (англ. domain subimages). Для каждого рангового блока алгоритм кодирования находит наиболее подходящий доменный блок и аффинное преобразование, которое переводит этот доменный блок в данный

ранговый блок. Структура изображения отображается в систему ранговых блоков, доменных блоков и преобразований.

Идея заключается в следующем: предположим, что исходное изображение является неподвижной точкой некоего сжимающего отображения. Тогда можно вместо самого изображения запомнить каким-либо образом это отображение, а для восстановления достаточно многократно применить это отображение к любому стартовому изображению.

По теореме Банаха, такие итерации всегда приводят к неподвижной точке, то есть к исходному изображению. На практике вся трудность заключается в отыскании по изображению наиболее подходящего сжимающего отображения и в компактном его хранении. Как правило, алгоритмы поиска отображения (то есть алгоритмы сжатия) в значительной степени переборные и требуют больших вычислительных затрат. В то же время, алгоритмы восстановления достаточно эффективны и быстры.

Вкратце метод, предложенный Барнсли, можно описать следующим образом. Изображение кодируется несколькими простыми преобразованиями (в нашем случае аффинными), то есть определяется коэффициентами этих преобразований (в нашем случае A, B, C, D, E, F).

Например, на рис.1.1. кривой Коха можно закодировать четырьмя аффинными преобразованиями, мы однозначно определим его с помощью всего 24-х коэффициентов.

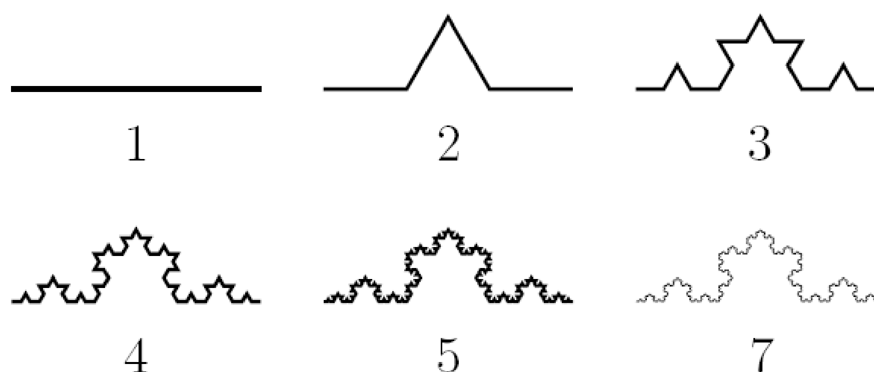


Рис. 1.1. кривая Коха

Далее, поставив чёрную точку в любой точке картинки мы будем применять наши преобразования в случайном порядке некоторое (достаточно большое) число раз (этот метод ещё называют фрактальным пинг-понгом).

В результате точка обязательно перейдёт куда-то внутрь чёрной области на исходном изображении. Проделав такую операцию много раз мы заполним все чёрное пространство, тем самым восстановив картинку.

Основная сложность фрактального сжатия заключается в том, что для нахождения соответствующих доменных блоков, вообще говоря, требуется полный перебор. Поскольку при этом переборе каждый раз должны сравниваться два массива, данная операция получается достаточно длительной. Сравнительно простым преобразованием её можно свести к операции скалярного произведения двух массивов, однако даже скалярное произведение вычисляется сравнительно длительное время.

В этой главе будут рассмотрены алгоритмы встраивания информации в изображение. По способу встраивания информации стегоалгоритмы можно разделить на линейные (аддитивные), нелинейные и другие.

Алгоритмы аддитивного внедрения информации заключаются в линейной модификации исходного изображения, а ее извлечение в декодере производится корреляционными методами. При этом цифровой водяной знак обычно складывается с изображением-контейнером, либо «вплавляется» в него. Эти алгоритмы будут рассмотрены в п.1.1.

В нелинейных методах встраивания информации используется скалярное либо векторное квантование. Обзор соответствующих алгоритмов выполнен в п.1.2.

К другим методам относится и метод встраивания информации с помощью фрактального сжатия, являющийся темой выпускной квалификационной работы. Алгоритм фрактального сжатия рассматривается в п.1.3, а в п 6.4 приведена точная постановка задачи.

1.1. Линейные методы встраивания информации в изображение

В аддитивных методах внедрения цифрового водяного знака (ЦВЗ) представляет собой последовательность чисел wi длины N, которая внедряется в выбранное подмножество отсчетов исходного изображения f. Основное и наиболее часто используемое выражение для встраивания информации в этом случае

$$f^{/}(m,n) = f(m,n)(1+\alpha w_i), \quad (1.1)$$

где α - весовой коэффициент, а f' - модифицированный пиксел изображения.

Другой способ встраивания водяного знака, рассмотренный в [1], использует следующую формулу:

$$f^{/}(m,n) = f(m,n) + \alpha w_i \quad (1.2)$$

или, при использовании логарифмов коэффициентов

$$f^{/}(m,n) = f(m,n)e^{\alpha w_i} \quad (1.3)$$

где $e^{\alpha w_i}$ - это экспонента весового коэффициента, модифицированного пиксель изображения.

При встраивании в соответствии с (1.1) ЦВЗ (цифрового водяного знака) в декодере находится следующим образом:

$$w_i^* = \frac{f^*(m,n) - f(m,n)}{\alpha\, f(m,n)}. \quad (1.4)$$

Здесь под f* понимаются отсчеты полученного изображения, содержащего или не содержащего ЦВЗ w. После извлечения w_i^* сравнивается с подлинным ЦВЗ. Причем в качестве меры идентичности водяных знаков используется значение коэффициента корреляции последовательностей

$$\delta = \frac{w^* w}{\|w\|^* \|w\|}. \qquad (1.5)$$

Эта величина варьируется в интервале [-1; 1]. Значения, близкие к единице, свидетельствуют о том, что извлеченная последовательность с большой вероятностью может соответствовать встроенному ЦВЗ. Следовательно, в этом случае делается заключение, что анализируемое изображение содержит водяной знак.

Для извлечения внедренной информации в аддитивной схеме встраивания ЦВЗ обычно необходимо иметь исходное изображение, что достаточно сильно ограничивает область применения подобных методов.

Существуют слепые методы извлечения ЦВЗ [2], вычисляющие корреляцию последовательности w со всеми N коэффициентами полученного изображения f*:

$$\delta = \frac{\sum_N f(m,n)^* w_i}{N}. \qquad (1.6)$$

Затем полученное значение коэффициента корреляции δ сравнивается с некоторым порогом обнаружения τ,

$$\tau = \frac{\alpha}{3N} \sum_N \left| f(m,n)^* \right|. \qquad (1.7)$$

Основным недостатком этого метода является то, что само изображение в этом случае рассматривается, как шумовой сигнал. Существует гибридный подход (полуслепые схемы), когда часть информации об исходном изображении доступно в ходе извлечения информации, но неизвестно собственно исходное изображение.

Если вместо последовательности псевдослучайных чисел в изображение встраивается другое изображение (например, логотип фирмы), то соответствующие алгоритмы внедрения называются алгоритмами слияния. Размер внедряемого сообщения намного меньше размера исходного изображения. Перед встраиванием оно может быть зашифровано или преобразовано каким-нибудь иным образом. У таких алгоритмов есть два преимущества.

Во-первых, можно допустить некоторое искажение скрытого сообщения, так как человек все равно сможет распознать его.

Во-вторых, наличие внедренного логотипа является более убедительным доказательством прав собственности, чем наличие некоторого псевдослучайного числа.

1.2. Обзор методов фрактального сжатия изображения

1.2.1. Стеганографические методы на основе квантования

Под квантованием понимается процесс сопоставления большого (возможно и бесконечного) множества значений с некоторым конечным множеством чисел. Понятно, что при этом происходит уменьшение объема информации за счет ее искажения. Квантование находит применение в алгоритмах сжатия с потерями. Различают скалярное и векторное квантование. При векторном квантовании, в отличие от скалярного, происходит отображение не отдельно взятого отсчета, а их совокупности (вектора). Из теории информации известно, что векторное квантование

эффективнее скалярного по степени сжатия, обладая большей сложностью. В стеганографии находят применение оба вида квантования.

В кодере квантователя вся область значений исходного множества делится на интервалы, и в каждом интервале выбирается число, его представляющее. Это число есть кодовое слово квантователя и обычно бывает центром интервала квантования. Множество кодовых слов называется книгой квантователя. Все значения, попавшие в данный интервал, заменяются в кодере на соответствующее кодовое слово. В декодере принятому числу сопоставляется некоторое значение. Интервал квантования обычно называют шагом квантователя.

Встраивание информации с применением квантования относится к нелинейным методам. Модель стегосистемы, не требующей наличия исходного сигнала в декодере представлена на рис. 1.2.

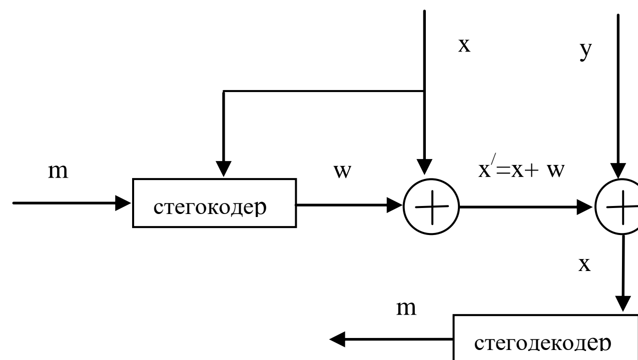


Рис. 1.2. Модель «слепой» стегосистемы

Передаваемое сообщение m имеет ограниченную энергию для выполнения требования его незаметности. Помехами являются исходный сигнал и еще одна гауссовская помеха – шум обработки (квантования). Кодеру исходный сигнал известен, декодер должен извлечь ЦВЗ m без знания обеих составляющих помех.

Наиболее предпочтительно внедрение информации в спектральную область изображения. Если при этом используются линейные методы, то встраивание ЦВЗ производят в средние полосы частот. Это объясняется тем, что энергия изображения сосредоточена, в основном, в низкочастотной (НЧ)

области. Следовательно, в детекторе ЦВЗ в этой области наблюдается сильный шум самого сигнала. В высокочастотных (ВЧ) областях большую величину имеет шум обработки, например, сжатия. В отличие от линейных, нелинейные схемы встраивания информации могут использовать НЧ области, так как мощность внедряемого ЦВЗ не зависит от амплитуды коэффициентов. Это объясняется тем, что в нелинейных алгоритмах скрытия не используется корреляционный детектор, коэффициенты малой и большой амплитуды обрабатываются одинаково.

Итак, как показано на рис.1.2, внедряемый ЦВЗ m определенным образом модулируется и складывается с исходным сигналом x, в результате чего получается заполненный контейнер s(x,m). Этот контейнер может рассматриваться и как ансамбль функций от x, проиндексированных по m, т.е. sm(x). Эти функции обладают следующими свойствами:

- каждая из них должна быть близка, визуально неотличима от x;
- точки одной функции должны находиться на достаточном расстоянии от точек другой функции, чтобы обеспечить возможность детектирования ЦВЗ.

В качестве таких функций может выступать семейство квантователей. Число всевозможных m определяет необходимое число квантователей; индекс m определяет используемый квантователь для представления ЦВЗ m . Для случая m=2 мы получаем бинарный квантователь. На рис.1.3 поясняется принцип встраивания информации с применением модуляции индекса квантования (МИК). Для вложения бита $m, m \in \{1,2\}$, точка изображения отображается в одно из близлежащих кодовых слов.

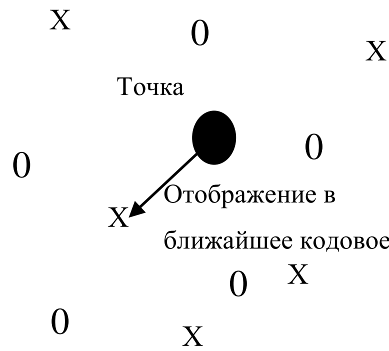


Рис. 1.3. Отображение точки изображения в близлежащее кодовое слово

1.2.2. Стеганографические методы на основе фрактального сжатия

В основе этих методов используются идеи, заимствованные из области кодирования изображений. Важным преимуществом фрактального метода сжатия для многих приложений является его резкая асимметричность. Фрактальное кодирование изображения – долгий и требующий серьезных вычислительных затрат процесс, в то время как декодер реализуется исключительно просто. Так, сжатый этим методом видеофильм может быть воспроизведен даже на 386DX-40.

Основная идея метода сжатия заключается в поиске последовательности аффинных преобразований (поворот, сдвиг, масштабирование), позволяющих аппроксимировать блоки изображения малого размера (ранговые) блоками большего размера (доменами). То есть считается, что изображение самоподобно. Эта последовательность преобразований и передается декодеру. Будучи примененными к любому изображению, эти преобразования дают в результате искомое изображение. Фрактальное кодирование может рассматриваться, как разновидность векторного квантования, причем в качестве кодовой книги выступают различные преобразования. Более подробно идея фрактального сжатия рассматривается в п.1.4.

В качестве ЦВЗ выступает строка бит. Секретным ключом, от которого зависит эффективность всего алгоритма, является в данном случае выбор

рангового блока. Число ранговых блоков есть верхняя граница для числа встраиваемых бит. Доменный пул делится на две части: одной будет соответствовать внедрение единиц, другой – внедрение нулей [1].

ЦВЗ добавляется следующим образом. Для выбранного в соответствии с ключом рангового блока в доменном пуле ищется соответствующий блок. Если надо встроить 1, поиск выполняется в одной части пула, если 0 - в другой части. Для ранговых блоков, в которые не встраивается биты ЦВЗ, поиск осуществляется во всем доменном пуле. После фрактального кодирования изображения осуществляется его декодирование для получения исходного изображения.

Декодер знает секретный ключ и выполняет обратные преобразования, восстанавливая ЦВЗ.

1.3. Метод фрактального сжатия изображений

Фрактальная архивация основана на том, что изображение представляется в более компактной форме — с помощью коэффициентов системы итерируемых функций (Iterated Function System – IFS). Прежде, чем рассматривать сам процесс архивации, рассмотрим, как IFS строит изображение, т.е. процесс декомпрессии.

Строго говоря, IFS представляет собой набор трехмерных аффинных преобразований, переводящих одно изображение в другое. Преобразованию подвергаются точки в трехмерном пространстве (x-координата, y-координата, яркость).

Наиболее наглядно этот процесс продемонстрировал в [4]. Там введено понятие «Фотокопировальной Машины», состоящей из экрана, на котором изображена исходная картинка, и системы линз, проецирующих изображение на другой экран, например, как показано на рис.1.4.

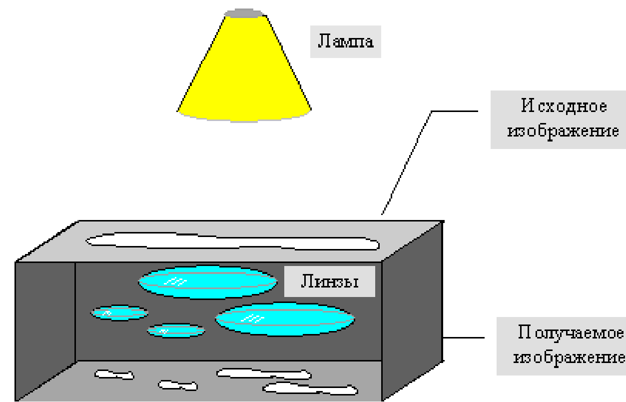


Рис.1.4. Представление алгоритма фрактальной декомпрессии в виде фотокопировальной машины

Линзы могут проецировать часть изображения произвольной формы в любое другое место нового изображения. Области, в которые проецируются изображения, не пересекаются. Линза может менять яркость и уменьшать контрастность. Линза может зеркально отражать и поворачивать свой фрагмент изображения. Линза должна масштабировать (уменьшать) свой фрагмент изображения.

Расставляя линзы и меняя их характеристики, возможно управлять получаемым изображением. Одна итерация работы машины заключается в том, что по исходному изображению с помощью проектирования строится новое, после чего новое берется в качестве исходного. Утверждается, что в процессе итераций мы получим изображение, которое перестанет изменяться. Оно будет зависеть только от расположения и характеристик линз, и не будет зависеть от исходной картинки. Это изображение называется “неподвижной точкой” или аттрактором данной IFS. Существует теорема, гарантирующая наличие ровно одной неподвижной точки для каждой IFS.

Таким образом, фрактальная компрессия — это поиск самоподобных областей в изображении и определение для них параметров аффинных преобразований [8].

В худшем случае, если не будет применяться оптимизирующий алгоритм, потребуется перебор и сравнение всех возможных фрагментов

изображения разного размера. Даже для небольших изображений при учете дискретности получается астрономическое число перебираемых вариантов.

Причем, даже резкое сужение классов преобразований, например, за счет масштабирования только в определенное количество раз, не дает заметного выигрыша во времени. Кроме того, при этом теряется качество изображения. Подавляющее большинство исследований в области фрактальной компрессии сейчас направлены на уменьшение времени архивации, необходимого для получения качественного изображения.

1.4. Примеры фрактального изображения

В окружающей нас природе фрактальные структуры с той или иной степенью подобности можно встретить практически повсеместно. Очевидно, отчасти это связано с тем, что многие органические и неорганические формы формируются аналогично.

В настоящее время к фрактально подобным объектам можно отнести морские раковины, границы морских побережий, горные цепи, зимние узоры на стекле, трещины в некоторых породах, облака и так далее.

Теперь рассмотрим, отдельную веточку дерева. Внимательное ее изучение обязательно натолкнет на мысль, что она со своими сучками и развилками очень похожа на дерево. Такая схожесть отдельной части (ветки) с целым (деревом) говорит в пользу распространенного в природе принципа рекурсивного само подобия. Поэтому разнообразные природные формы можно описать фрактальным алгоритмом.

Рассмотрим один из классификаций фрактальных структур – это геометрические фракталы.

Геометрические фракталы – это ранний тип фракталов. Один из самых наглядных, в нем сразу видна само подобность частей и получаются они из самых простых геометрических построений, как показано на рис. 1.5. и рис. 1.6:

- Задается фигура (нулевое поколение), на основе которой будет строиться фрактал.
- Задается процедура-генератор, которая на основе определенного правила преобразует нулевое поколение.
- Бесконечное повторение процедуры-генератора позволяет получить геометрический фрактал.

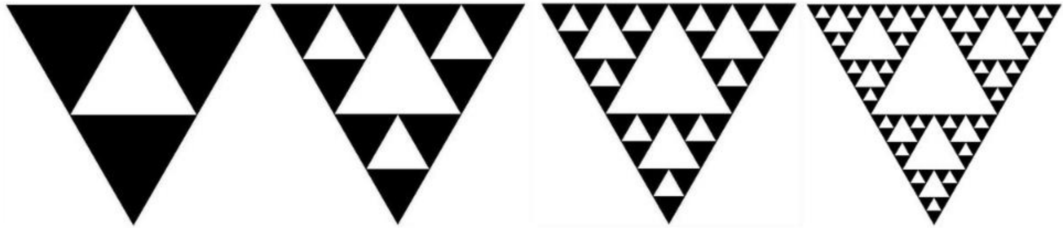


Рис. 1.5. Последовательные итерации построения треугольника Серпинского

Также можно увидеть последовательную итерацию квадратной кривой Коха, как показана на рис.1.6.

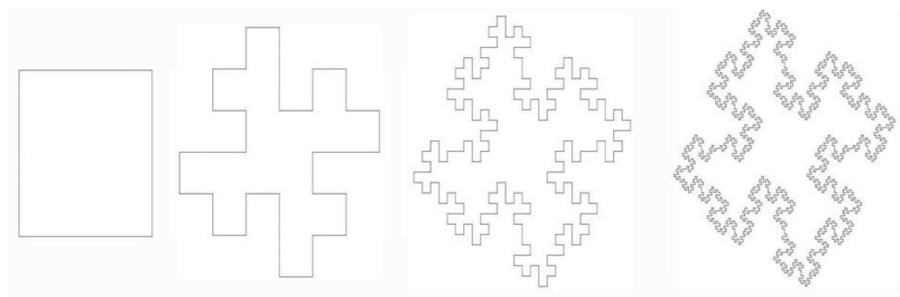


Рис. 1.6. Последовательные итерации построения квадратной кривой Коха

Построение конструктивных фракталов возможно также с помощью системы итерируемых функций (Iterated Function System), или сокращенно IFS, представляющих собой систему функций некоторого фиксированного класса, позволяющих отображать одно многомерное множество на другое. Одна из самых простых IFS включает аффинные преобразования плоскости, и в двумерном пространстве задается на основе 6 коэффициентов:

$$\begin{cases} x = a_1 x_0 + b_1 y_0 + c_1, \\ y = a_2 x_0 + b_2 y_0 + c_2 \end{cases} \tag{1.8}$$

Обычно IFS используется для построения листьев, цветов, веток, деревьев и кустарников при создании реалистичных картин в компьютерном дизайне и играх. На рис. 1.7. приведены примеры изображений, полученных на основе IFS: лист папоротника (а), коралл (б) Для качественного отображения на основе IFS требуется достаточно большое количество итераций. Так, для изображений, представленных на рис. 9, использовано от 100 тыс. до 300 тыс. итераций.

Рис. 1.7. Демонстрация работы алгоритма IFS

Рассмотрим еще некоторые примеры фрактальных структур. На рис. 1.8 и рис.1.9. приведены примеры реальных природных форм, демонстрирующих фрактальную структуру.

На рис. 1.8. показана фотография раковины моллюска. Видно, что каждая отдельная «комнатка» является уменьшенной версией предыдущей.

Рис. 1.8. Раковина моллюска Наутилус – естественный фрактал

На рис. 1.9. показано фото цветной капусты Романеско, форма которой похожа на фрактал (это позволяет назвать ее естественным фракталом). Однако само подобная структура капусты повторяется лишь несколько раз, и прекращается на более мелких уровнях [12].

Рис. 1.9. Цветная капуста Романеско – естественный фрактал

1.5. Постановка задачи

Требуется разработать программное обеспечение, выполняющее следующие функции:

- Открытие файлов с изображением в формате BMP (256 градаций серого);

- Нахождение коэффициентов IFS для фрактального сжатия открытого изображения;
- Возможность встраивания в систему IFS заданной в виде цепочки бит информации;
- Сохранение системы коэффициентов IFS в файл;
- Открытие файла IFS и восстановление исходного изображения.
- Восстановление встроенной в систему IFS битовой информации.
- Разработанное программное обеспечение должно обладать интуитивно понятным графическим интерфейсом.

2. РАЗРАБОТКА МАТЕМАТИЧЕСКОГО АППАРАТА И АЛГОРИТМОВ

В этой главе будут введены и разъяснены базовые понятия, необходимые для понимания алгоритма фрактальной компрессии и декомпрессии, приведены формулы математического аппарата, четко формулирующие задачи для алгоритмов, описаны алгоритмы, реализующие фрактальное сжатие и распаковку.

2.1. Математические основы фрактального сжатия изображения

2.1.1. Фундаментальные определения и теоремы

Метрическое пространство - это пространство χ с введенной на ней метрикой δ. Такое пространство обычно обозначают (χ, δ).

Для любых трех точек a, b, c из пространства χ функция δ является метрикой если соблюдаются четыре условия:

- Для любых двух точек $a, b \in \chi$, $a \neq b$ выполняется следующее неравенство: $0 < \delta(a,b) < \infty$;
- $\delta(a,a) = 0$ для любых $a \in \chi$;
- $\delta(a,b) = \delta(b,a)$ для любых $a, b \in \chi$;
- $\delta(a,b)\ \delta(a,c) + \delta(c,b)$ для любых $a, b, c \in \chi$.

Неподвижная точка. Пусть задана точка $a \in \chi$ и преобразование ω: $\chi \rightarrow \chi$. Если $\omega(a) = a$, то a называют неподвижной точкой преобразования ω.

Сжимающее преобразование. Преобразование ω, работающее в метрическом пространстве (χ,δ) называется сжимающим, если существует положительное действительное число $0 \leq s < 1$, такое что $\delta(\omega(a), \omega(b)) \leq s\delta(a,$

b) для любых точек a, b $\in \chi$. Число s называют сжимаемостью преобразования ω.

Теорема о сжимающем отображении. Пусть дано метрическое пространство χ и преобразование ω, работающее в этом пространстве. Если ω - сжимающее, то существует только одна неподвижная точка $|\omega(a)| = \lim \omega^n(a)$ при $n \to \infty$ для любой $a \in \chi$.

Теорема коллажа. Для точки $a \in \chi$ и сжимающего преобразования ω расстояние между точкой a и неподвижной точкой можно оценить $\delta(a, |\omega(a)|) \leq \delta(a, \omega(a)) / (1 - s)$, где s - это сжимаемость ω.

Пространство Хаусдорфа. Пусть μ - это изображение. Непрерывное непустое подмножество μ в метрическом пространстве (χ,δ) называется пространством Хаусдорфа $H(\chi)$.

Метрика в пространстве Хаусдорфа. Пусть (χ,δ) - метрическое пространство. Для каждого непрерывного подмножества m пространства χ и $\varepsilon > 0$ определим ε-окрестность μ следующим образом $\mu\varepsilon = \{a \in \chi \mid \delta(a,b) \leq \varepsilon$ для какого-то $b \in \mu\}$. Для двух непрерывных подмножеств μ и ν пространства χ метрика Хаусдорфа определяется следующим образом: $h(\mu,\nu) = \inf\{\varepsilon \mid \mu \subset \nu\varepsilon \; \& \; \nu \subset \mu\varepsilon\}$.

Аффинное преобразование - композиция линейного преобразования и параллельного переноса. В двумерном пространстве для полного представления аффинного преобразования достаточно задать 6 коэффициентов.

IFS - Система итерационных функций - это система, состоящая из набора аффинных преобразований w1, w2, … ,wn, работающих в пространстве Хаусдорфа $(H(\chi), h)$.

PIFS - Система сегментированных итерационных функций - это система, состоящая их набора аффинных преобразований w1, w2, …, wn, работающих в пространстве Хаусдорфа. Причем, каждое wi преобразует

только определенную часть пространства, обозначенную $\chi\mu$, то есть wi: $\chi\mu \rightarrow \chi$.

Оператор Хатчинсона. Пусть заданы сжимающие аффинные преобразования w1, w2, …. , wn, работающие в $\mu \in H(\chi)$. Оператор Хатчинсона задается как $W(\mu) = w1(\mu) \cup w2(\mu) \cup \ldots \cup wn(\mu)$, где $W: H(\mu) \rightarrow H(\mu)$. Доказано, что если w1, w2, … , wn - сжимающие, то W - тоже сжимающий [6].

2.1.2. Основные принципы теории фрактального сжатия

Пусть задано пространство Хаусдорфа всех изображений, имеющих определенный размер. Пространство имеет метрику, введенную выше. Следовательно, на нем можно ввести систему итерационных функций (IFS). В качестве оператора преобразований, входящих в систему, берется оператор Хатчинсона, по определению составленный из композиции сжимающих аффинных преобразований пространства Хаусдорфа.

Таким образом, получим систему итерационных функций, которую математически можно записать так:

$$(H(\chi), h, WIFS(\mu)) \tag{2.1}$$

где:

- $H(\chi,h)$ - пространство Хаусдорфа с метрикой h;
- $WIFS(\chi)$ - оператор Хатчинсона, работающий в пространстве Хаусдорфа и - действующий на точку χэтого пространства.

Основываясь на теореме о сжимающем отображении, можно сделать предположение, что некоторое изображение из пространства Хаусдорфа может быть полностью описано только при помощи какого-то конечного набора преобразований, входящих в состав оператора Хатчинсона. Другими

словами, если взять некоторое изображение $\chi 0$ из пространства Хаусдорфа, то, введя каким-то образом систему сжимающих аффинных преобразований и построив из них оператор Хатчинсона $H(\chi)$, вероятно (но необязательно) можно добиться того, что m0 станет неподвижной точкой для этих преобразований. Таким образом, согласно теореме о сжимающем отображении, мы можем получить исходное изображение при помощи многократного действия построенного оператора H на любое изображение из пространства Хаусдорфа [11].

Вполне очевидно, что изображение $\chi 0$ обладает свойствами самоподобия. То есть оно может быть разбито на участки, такие, что каждый из этих участков может быть получен в результате применения какого-то аффинного преобразования ко всему изображению.

Проблема применения системы итерационных функций для сжатия изображений в том, что каждое преобразование из набора отображает все данное пространство в какую-то часть того же пространства. Наивным будет предположить, что реальное изображение обладает, свойством полного самоподобия.

Вследствие этого возникла необходимость во введении системы сегментированных итерационных функций (PIFS). Отличие этой системы от IFS состоит в том, что каждый оператор действует не на все изображение, а только на определенный «кусок». К PIFS, построенной при помощи оператора Хатчинсона, так же как и к IFS, применима теорема о сжимающем отображении. Следовательно, независимо от начального изображения m, система всегда сходится к какому-то стабильному изображению.

2.1.3. Применение теории фрактального сжатия к сжатию реальных изображений

Из предыдущего пункта можно сделать вывод, что некоторое изображение можно определенным образом описать с помощью серии

сжимающих преобразований (в частном случае аффинных), которые, будучи применены многократно на любом начальном изображении, дадут, в конце концов, исходное изображение. Уповая на то, что представление этих преобразований в виде данных по объему гораздо меньше исходного изображения, в файле можно сохранить только сами преобразования, по которым позже, при открытии файла, можно будет полностью восстановить исходную картину.

Но, к сожалению, как было замечено выше, реальные изображения не обладают свойствами полного самоподобия. То есть о применении теории IFS к реальным картинам, фотографиям не может быть и речи. Но, все же, если взять отдельное изображение и выделить некоторые небольшие участки, то можно увидеть некоторое сходство, подобие. Поэтому, сжатие реальных изображений имеет свою специфику, о которой дальше и пойдет речь.

Возьмем изображение orig. Изображение считается закодированным, если оно представлено в виде PIFS. В идеале, мы должны найти оператор Хатчинсона W, который оставляет изображение morig неизменным, то есть такой, что orig = W(orig).

Очевидно, что достичь этого крайне сложно. Во-первых, нет никаких доказательств того, что такой оператор действительно существует для произвольного изображения. Во-вторых, не существует никакого другого метода поиска этого оператора, кроме прямого перебора. То есть нужно исследовать все возможные комбинации сжимающих преобразований (которых, кстати, бесконечно много) и, поочередно применяя их к изображению, найти то, которое его не изменяет.

Поэтому, принимая во внимание визуальную избыточность реальных изображений, можно искать оператор Хатчинсона, который оставляет исходное изображение практически неизменным, насколько это возможно. То есть можно найти такое оператор, для которого неподвижной точкой является изображение.

Итак, реальное изображение кодируется таким образом, что начиная с любого начального изображения, и применяя построенную PIFS, получается стабильное изображение. Конечно, и в этом случае приходится искать оператор Хатчинсона прямым перебором. Но, сделав вышеописанное приближение, мы можем наложить некоторые ограничения на количество преобразований.

2.2.Разработка алгоритма компрессии

Разработку алгоритма можно условно разделить на три этапа:

- Определить конечное множество или совокупность аффинных преобразований.
- Разработать метод дробления изображения на отдельные фрагменты таким образом, чтобы определенные ранее преобразования могли работать на этих фрагментах. Этот этап называют генерацией совокупности доменов и блоков.
- Определить процедуру поиска. То есть для каждого блока нужно найти такой домен и такое преобразование, чтобы ошибка была минимальной.

2.2.1. Терминология и определения

Пиксель - элементарная неделимая составляющая плоского прямоугольного изображения. Каждый пиксель изображения имеет две уникальные целочисленные координаты (x, y), подчиняющиеся условию (2.2).

$$0 \le x < Xmax,\ 0 \le y < Ymax \qquad (2.2)$$

где:

- Xmax - размер изображения по горизонтали;
- Ymax - размер изображения по горизонтали.

Каждому пикселю сопоставляется целое число, лежащее в пределах [0, n], называемое интенсивностью. Блок – квадратный фрагмент изображения определенного размера. Домен - квадратный фрагмент изображения, размеры которого в 2 раза больше размера блока [3].

RMS (среднеквадратичное отклонение). Способ измерения расстояния между двумя блоками, описанный формулой (2.3).

$$RMS(\mu,\nu) = \frac{1}{n^2}\sqrt{\sum_{0\le i,j<n}\left(\mu_{ij} - \nu_{ij}\right)^2} \qquad (2.3)$$

где:

- μ, ν - произвольные блоки одинакового размера n.

Средняя интенсивность блока вычисляется по формуле (2.4):

$$g0(\mu) = \frac{1}{n^2}\sum_{0\le i,j<n}\mu_{ij} \qquad (2.4)$$

Среднеквадратичное отклонение от средней интенсивности вычисляется по формуле (2.5):

$$\delta 0(\mu) = RMS(\mu, g0\mathbf{1}) \qquad (2.5)$$

где:

- g0**1** – блок, совпадающий по размерам с блоком μ, все пиксели которого имеют единичную интенсивность.

2.2.2. Определение конечного множества аффинных преобразований

Как было отмечено ранее, любое аффинное преобразование можно полностью описать с помощью шести чисел. Но чтобы получить более существенный коэффициент сжатия, мы ограничимся дискретным набором аффинных преобразований.

Обозначим совокупность преобразований T и разделим их на три части: геометрическую часть G, поворот S и, наконец, изменение яркости и контрастности пикселей M.

Геометрическая часть G выполняет функцию сжимающего оператора и оператора параллельного переноса. Оператор G, действуя на домен, сжимает его в 2 раза и переносит на место блока.

Оператор поворота S преобразует блок изображения одним из способов, описанных в таблице 2.1.

Таблица 2.1.

Оператор поворота S

Код	Описание	Действие
0	тождественное преобразование	t0(μ[i,j]) = μ[i,j]
1	отражение от горизонтальной оси симметрии	t1(μ[i,j]) = μ[i,n-1-j]
2	отражение от вертикальной оси симметрии	t2(μ[i,j]) = μ[n-1-i,j]
3	отражение от главной диагонали	t3(μ[i,j]) = μ[j,i]
4	отражение от побочной диагонали	t4(μ[i,j]) = μ[n-1-j,n-1-i]
5	поворот на -90°	t5(μ[i,j]) = μ[j,n-1-i]
6	поворот на 180°	t6(μ[i,j]) = μ[n-1-i,n-1-j]
7	поворот на 90°	t7(μ[i,j]) = μ[n-1-j,i]

Последнее преобразование M действует на пиксели в блоке напрямую, изменяя значения их интенсивностей по формуле (2.6):

$$M(\mu) = \alpha\mu + \Delta \qquad (2.6)$$

где:

- α – множитель контрастности;
- Δ – сдвиг яркости

Итак, комбинацию всех этих преобразований можно записать как w = M° S° G. Полученная совокупность преобразований ω и есть искомый набор аффинных преобразований. Нетрудно заметить, что для описания этих преобразований потребуется гораздо меньшее количество данных, чем для хранения растра изображения [6].

2.2.3. Дробление изображения

Теперь следует определить правило разбиения изображения на блоки и домены.

Для того, чтобы снизить время работы процедуры компрессии, принято решение все блоки генерировать в виде квадратов $n \times n$ пикселей. Размеры изображения X и Y должны быть кратны n. Таким образом, исходное изображение можно разбить на (X/n)*(Y/n) непересекающихся блоков.

Домен представляет собой квадрат размером $2{*}n \times 2{*}n$ пикселей. Каждый домен включает в себя четыре блока. То есть, исходное изображение можно разбить на (X/n-1)*(Y/n-1) доменов, которые, в отличие от блоков, могут пересекаться.

Поскольку существуют ситуации, когда приходится разбивать блоки на блоки вдвое меньшего размера, причем эта процедура является рекурсивной,

к размеру n блока предъявляется еще одно требование – он должны являться степенью двойки.

2.2.4. Поиск оптимального аффинного преобразования

Алгоритм поиска состоит в нахождении для каждого блока Bi такой пары (Dj, k), чтобы RMS(Bi, k(Dj)) было минимальным.

В том случае, когда для блока не удается найти подходящего домена в пределах допустимой ошибки, то этот блок можно разделить на четыре блока размерами n/2 n/2 пикселей. Для полученных в результате такого разбиения блоков проделывается точно такая же процедура нахождения домена, что и ранее. Таким образом, в настройках алгоритма должна задаваться максимальная глубина дробления блока.

2.2.5. Встраивание цифровых водяных знаков в изображение

Как упоминалось в пункте 1.3, секретным ключом для встраивания и извлечения секретной информации является способ разделения доменного пула на две части - для единиц и для нулей соответственно. Если текущий блок должен кодировать единицу, поиск соответствующего ему домена должен выполняться только в единичной части, если блок должен кодировать ноль – в нулевой части, если блок не несет информации – в обеих частях.

В данной работе в качестве ключа выбрано разделение пула по принципу шахматной доски. X-координата домена суммируется с его Y-координатой. Если результат четный, домен относится к нулевой части пула, иначе – к единичной.

Как уже упоминалось, максимальное количество кодируемых бит равно количеству ранговых блоков, на которые разбивается исходное изображение. Однако, при использовании дробления изображения это не

совсем так. Поскольку каждый ранговый блок может быть, в свою очередь, разбит на более мелкие блоки, общее количество блоков резко возрастает.

К сожалению, перед началом кодирования их общее количество неизвестно, поэтому все равно приходится ограничиваться количеством исходных блоков.

Если блок разбивается, поиск домена для него не производится. Это значит, что мы должны кодировать информацию в одном из блоков-потомков. При кодировании следующего бита возникает альтернатива – кодировать ли его следующим блоком-потомком, или обязательно дожидаться обработки очередного блока верхнего уровня и только тогда встраивать информацию.

Эти подходы принципиально не отличаются, но первый из них алгоритмически проще, поэтому именно он выбран в данной работе. При этом на алгоритм декодирования накладывается следующее ограничение: абсолютно все блоки должны извлекаться из сжатого изображения точно в таком же порядке, как и при кодировании. Выполнение этого ограничения не требует никаких дополнительных затрат.

2.2.6. Разработка схемы алгоритма компрессии

Схема алгоритма кодирования и встраиванием информации приведена на рисунке 2.1. Основные структуры данные, используемые алгоритмом – это массив векторов блоков blocks[] и массив векторов доменов domains[].

Индекс в массиве соответствует уровню фрактализации. Например, для уровня фрактализации 0 в векторе blocks[0] будет храниться (width*height)/(blocksize*blocksize) блоков. Если в процессе анализа какой-либо блок будет разбит, его потомки должны быть занесены в вектор blocks[1]. Для каждого вектора blocks[k] формируется вектор доменов domains[k], который включает все домены на данном уровне. Рассмотрим алгоритм более подробно представлен в приложении 3.

В блоках 1 и 2 происходит подготовка к кодированию, т.е. проверяется исходное изображение, формируется вектор blocks[0], определяется количество кодируемых бит, очищаются структуры данных и т.п.

В блоке 3 запускается цикл по «i» от 0 максимального разрешенного уровня фрактализации. В блоке 4 анализируется наличие блоков в текущем векторе blocks[i]. Если блоков нет, значит, ни один блок предыдущего уровня не был разбит и кодирование заканчивается.

Блок 5 формирует вектор доменов для данного уровня. Блок 6 запускает цикл по всем блокам данного уровня. В блоке 7 формируется фильтр для досрочного отсеивания доменов, не входящих в нужную половину пула. Если все биты кодовой строки уже закодированы, фильтр пропустит все домены. Если очередной бит кодовой строки равен «0», фильтр настроится так, чтобы пропускать только одну половину доменов, если «1» - то другую.

Блок 8 записывает в переменную экстремума bestRMS недостижимое значение, чтобы его заменило первое же вычисленное RMS. В блоке 8 запускается цикл по доменам для поиска наиболее подходящего для текущего блока. Блок 10 отфильтровывает неподходящие домены, а блок 11 запускает цикл для поиска наилучшего аффинного преобразования домена в блок.

Блоки 12, 13 и 14 реализуют простейшую логику для поиска наилучшего значения RMS. В результате, после выхода из циклов Ц4 и Ц3 в переменной bestRMS сохранено значение наименьшего отклонения. В блоке 17 оно сравнивается со значением настроек, анализируется текущий и максимально допустимый уровень фрактализации и принимается решение о разбиении блока (19). Если блок решено не разбивать, в нем запоминается лучший для него домен и преобразование (18) и осуществляется продвижение к следующему биту кодируемой строки (19).

После закрытия циклов по перебору блоков и уровней фрактализации (блоки 21 и 22) алгоритм завершает свою работу.

2.3. Разработка алгоритма декомпрессии

Как уже упоминалось, фрактальное сжатие является крайне несимметричным методом, т.е. сложность алгоритма компрессии на порядки выше сложности декомпрессии.

Единственное, что нужно помнить об алгоритме декомпрессии – это необходимость перебора блоков всех уровней именно в том порядке, в котором они обрабатывались при сжатии.

На входе алгоритма - указатели на исходное изображение и на изображение, куда должен быть записан результат. Алгоритм выполняет один шаг фрактального преобразования. На практике для полного восстановления изображения требуется 5-10 шагов. Схема алгоритма декомпрессии подробно представлена в приложении 4.

В блоке 1 производится запись во все существующие домены растра из исходного изображения. В блоке 2 выделяется память и определяется количество бит, которые будут восстановлены из изображения.

В блоке 3 запускается цикл по всем уровням фрактализации, а в блоке 4 – цикл для перебора всех блоков на текущем уровне. В блоке 5 проверяется, был ли подобран домен для данного блока. Если не был, значит, при кодировании блок был разбит и его обработка не требуется.

Иначе текущий блок обрабатывается. На основании сведений о домене-источнике и об аффинном преобразовании он генерирует кусок растра и этот кусок записывается в нужную позицию растра-результата (6). Затем, в блоке 7, анализируется, к какой части пула принадлежит домен-источник и в соответствии с этим в восстанавливаемую битовую строку записывается 0 или 1. В блоках 8 и 9 завершаются открытые циклы, а в блоке 10 формируется завершающий символ нуля для корректного отображения строки.

3. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

3.1. Обоснование и выбор среды разработки

Алгоритм фрактального кодирования предъявляет очень высокие требования к ресурсам вычислительной системы. Это говорит о том, что при выборе среды разработки в первую очередь необходимо руководствоваться качеством и производительностью машинного кода, который генерируется средой. После этого вывода можно сразу отказаться от среды разработки Visual Basic, т.к. для выполнения программ, написанных на VB, требуется интерпретатор (виртуальная машина msvbvm60.dll). Таким образом, выбор сводится к следующим возможностям – Borland Delphi, Borland C++ Builder и Microsoft Visual Studio. Первые две среды предназначены скорее для разработки клиентских БД-приложений. Они обладают неплохими возможностями по быстрому созданию современных интерфейсов. Но удобство отладки программы и качество генерирования машинного кода пока значительно уступают Microsoft Visual Studio.

Microsoft Visual Studio обладает, пожалуй, единственным недостатком – сложностью создания интерфейса. Но это полностью компенсируется её огромными возможностями по отладке кода и качеством генерируемой программы.

Итак, в качестве средства разработки выбирается последняя версия Visual Studio – MS VisualStudio.NET (.NET 8.0). Эта среда предоставляет единый интерфейс для написания программ на одном из многих языков, компиляторы которых входят в её состав – C++, C#, managed C++, VB, Java и т.д. Большинство из них ориентировано на поддержку .NET-технологии, поэтому для данного проекта подходят слабо. .NET-технология значительно облегчает процесс разработки приложений, но обладает повышенными требованиями к аппаратуре и установленному на машине конечного пользователя программному обеспечению.

Таким образом, остается сделать выбор между С++ и Managed C++. Т.к. Managed C++ в основном используется для совмещения кода, написанного на традиционном С++ и на одном из языков технологии .NET, его возможности в данном проекте востребованы не будут. Поэтому d качестве языка программирования выбирается С++.

3.2. Разработка структуры приложения

В этом разделе будут рассмотрены все классы, необходимые для реализации алгоритмов фрактального сжатия и стеганографического встраивания/извлечения информации.

В таблице 3.1. приведен список всех классов с кратким описанием их назначения.

Таблица 3.1.

Назначение классов

Класс	Назначение
cIFS	Система итерируемых функций. Агрегирует все данные, необходимые для работы с одной картинкой.
cOptions	Агрегирует все настройки программы, управляет их загрузкой и сохранением в ini-файл.
cDomain	Класс используется для хранения данных одного домена или рангового блока.
sConv	Структура для хранения одного аффинного преобразования
cConversionSet	Класс агрегирует все возможные преобразования.
domain_vector	Вектор STL для хранения объектов класса cDomain

На рис. 3.1. приведена структура приложения. На ней отображено, как взаимодействуют различные классы между собой. В диаграмме использована нотация OMT[5], как одна из двух наиболее популярных на сегодняшний

день (вторая – UML). Простая стрелка означает отношение осведомленности, стрелка с ромбом вначале – отношение агрегирования.

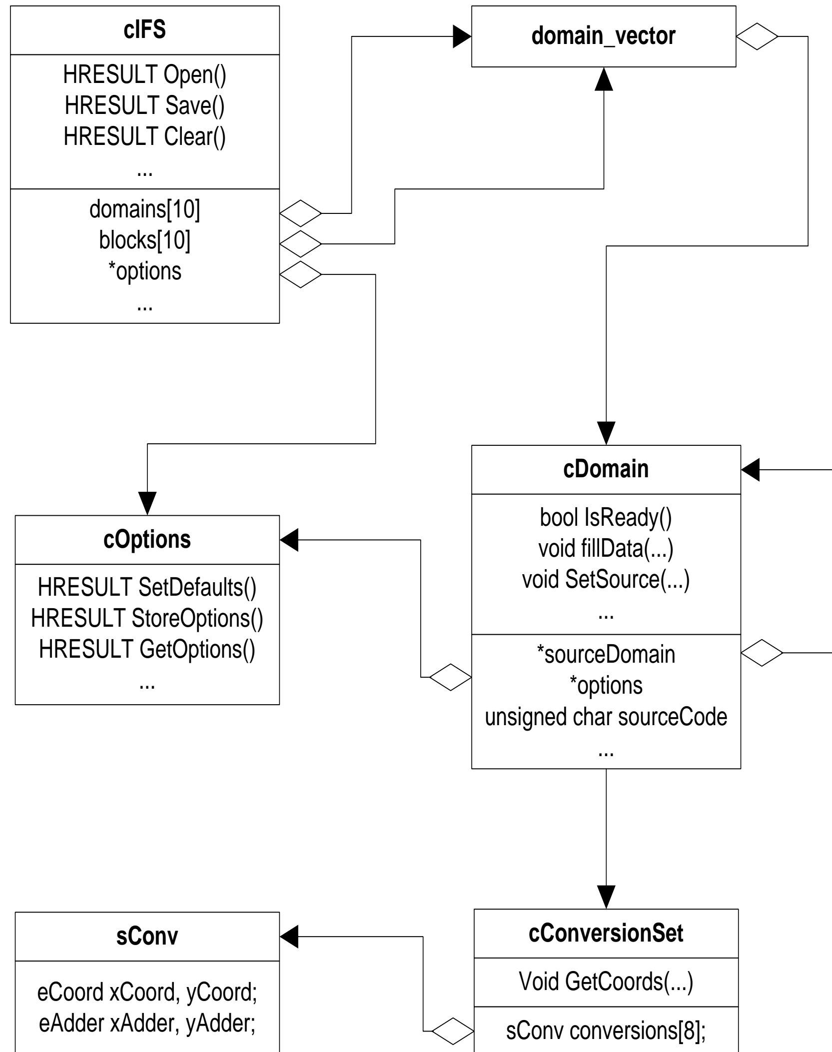


Рис. 3.1. Структура приложения

Рассмотрим структуру более подробно. В программе существует два объекта-одиночки – это объект класса cIFS и объект класса cConversionSet.

Первый представляет собой всю систему итерируемых функций. Он включает в себя (агрегирует) массивы domains[10] и blocks[10]. Каждый элемент этого массива является вектором класса domain_vector. Класс domain_vector реализован на основе соответствующего шаблона STL и объявлен следующим образом:

typedef std::vector<cDomain> domain_vector;*

Вектор представляет собой контейнер с переменным числом объектов и идеально подходит для хранения ранговых блоков и доменов.

Класс cDomain служит для хранения данных одного блока или домена. Он включает в себя растровые данные, указатель на домен-источник (если такой существует), номер аффинного преобразования, по которому происходит преобразование домена-источника в данный блок.

С помощью номера преобразования домен может обратиться к объекту cConversionSet и преобразовать координаты точки соответствующим образом. Именно поэтому в структуре приложения классы cDomain и cConversionSet соединены отношением осведомленности.

Класс cConversionSet агрегирует 8 экземпляров структуры sConv, каждая из которых хранит метаданные, необходимые для соответствующего аффинного преобразования. Такая структура позволяет легко и безболезненно расширить возможности программного обеспечения в случае, например, увеличения количества возможных преобразований.

Основные данные и функции классов cIFS:

- CImage *image - обрабатываемое изображение;
- domain_vector - массивы векторов для доменов и ранговых блоков;
- domains[10], blocks[10] - по одному вектору на уровень фрактализации;
- int maxFract - максимальный уровень фрактализации, достигнутый в процессе кодирования;
- int width, height – размеры изображения;
- int startBlockSize - Размер блока на начальном этапе фрактализации;
- HRESULT CheckImage(…) - Проверяет, соответствует ли обрабатываемая картинка выдвигаемым требованиям (размеры д.б. степенями 2);
- void GetDomains(…) – генерирует вектор доменов или блоков;
- void splitBlock(…) - разбивает указанный блок на 4 и помещает их в соответствующий вектор;

- HRESULT Open(…) - открывает файл с закодированным изображением;
- HRESULT Save(…) - сохраняет закодированное изображение в файл;
- HRESULT Clear() - очищает все вектора;
- HRESULT GetFromImage(…) - генерирует систему IFS на основе переданного изображения (кодирование);
- HRESULT MakeStep(…) - Делает один шаг по восстановлению изображения.

Основные данные и функции класса cDomain:

- Int x,y – координаты расположения домена;
- Int size – размер домена (сторона квадрата);
- int step – с каким шагом вычисляется положение домена (на что нужно умножить x или y чтобы получить координату в пикселах);
- Unsigned char *data – растровые данные, соответствующие блоку или домену;
- cDomain *sourceDomain - домен-источник растровых данных
- unsigned char sourceCode -код аффинного преобразования
- short int source_dk - сдвиг по яркости от домена-источника;
- int getSourceEven() - определяет, в какой части пула расположен домен-источник;
- void SetSource(…) - устанавливает домен-источник, код преобразования и сдвиг по яркости;
- void fillData(…) - записывает свои приватные данные в переменные, переданные по ссылке;
- HRESULT ScaleDown() - вдвое уменьшить домен (сжать растр, изменить size и т.п.);
- float getRMS(cDomain *domain) - получить разницу RMS между собой и указанным доменом;

- void SetRaster(CImage* image, bool scale) - считывает данные в растр из указанного изображения;
- void GetRaster(CImage* image) - возвращает данные из растра в указанное изображение.
- bool SaveToFile(…) - сохраняет себя в файл;
- bool SaveChildren(…) - сохраняет в файл потомков;
- bool ReadFromFile(…) - прочитать свои данные из файла;
- bool ReadChildren(…) - прочитать из файла потомков.

Основные данные и функции класса cOptions:

- int blockSize - начальный размер блока;
- float eps - максимальное значение RMS, при котором НЕ проводится разбивка блока;
- int iMax - максимально допустимый уровень фрактализации.

Основные данные структуры sConv:

- eCoord xCoord - определяет, что берется в качестве x-координаты новой точки (x или y старой точки);
- eCoord yCoord - определяет, что берется в качестве y-координаты новой точки (x или y старой точки);
- eAdder xAdder - Определяет слагаемое, которое добавляется при формировании x-координаты (может быть 0 или n-1);
- eAdder yAdder - определяет слагаемое, которое добавляется при формировании y-координаты (может быть 0 или n-1).

Основные данные структуры cConversionSet:

- sConv conversions[8] - массив всех доступных аффинных преобразований;
- void GetCoords(…) - функция преобразования координат в соответствии с заданным кодом преобразования

3.3. Разработка формата файла

Для хранения сжатого изображения был разработан файловый формат IFS. Этот формат не является оптимальным, его можно усовершенствовать за счет отказа от выравнивания данных на границу байта, т.е. перед записью файла конвертировать все данные в битовую цепочку.

Файловый формат IFS:

- смещение=0, размер=1,поле - maxFract – достигнутый уровень фрактализации;
- смещение=1, размер=1,поле - начальный размер ячейки;
- смещение=2, размер=2,поле - ширина;
- смещение=4, размер=2,поле - высота;
- смещение=6, поле - массив структур BLOCK;
- Формат структуры BLOCK:
- смещение=0, размер=1,поле - код преобразования если код = 0xFF, данный блок является разбитым. Вместо остальных полей в файле последовательно хранятся потомки, также в формате BLOCK..начальный размер ячейки;
- смещение=1, размер=2,поле - сдвиг по яркости;
- смещение=1, размер=2,поле - сдвиг по яркости;
- смещение=3, размер=4,поле - множитель контрастности;
- смещение=7, размер=1,поле - x- координата домена-источника сдвиг по яркости;
- смещение=8, размер=1,поле -y- координата домена-источника;

3.4. Разработка пользовательского интерфейса

Поскольку данная программа выполняет ограниченный набор функций, разработанный пользовательский интерфейс достаточно прост. Он состоит из главного окна, окна настроек и окна с информацией о программе.

3.4.1. Главное окно

Экранный снимок главного окна приведен на рис. 3.2.

Окно включает в себя следующие элементы управления:

- Меню (список функций приведен в таблице 3.9);
- Графический контейнер для хранения исходного изображения;
- Графический контейнер для хранения закодированного изображения;
- Графический контейнер для хранения исходного изображения;
- Кнопку «Open» для открытия исходного изображения;
- Кнопки «Open» и «Save» для открытия и сохранения закодированного изображения;
- Текстовое поле для ввода битовой строки;
- Кнопка «Start Coding» для запуска процесса кодирования;
- Кнопка «Make One Step» для запуска одного шага декодирования;
- Кнопки «White» и «Black» для заливки контейнера графического изображения белым или черным цветом;
- Текстовое поле для отображения извлеченной из изображения битовой строки.

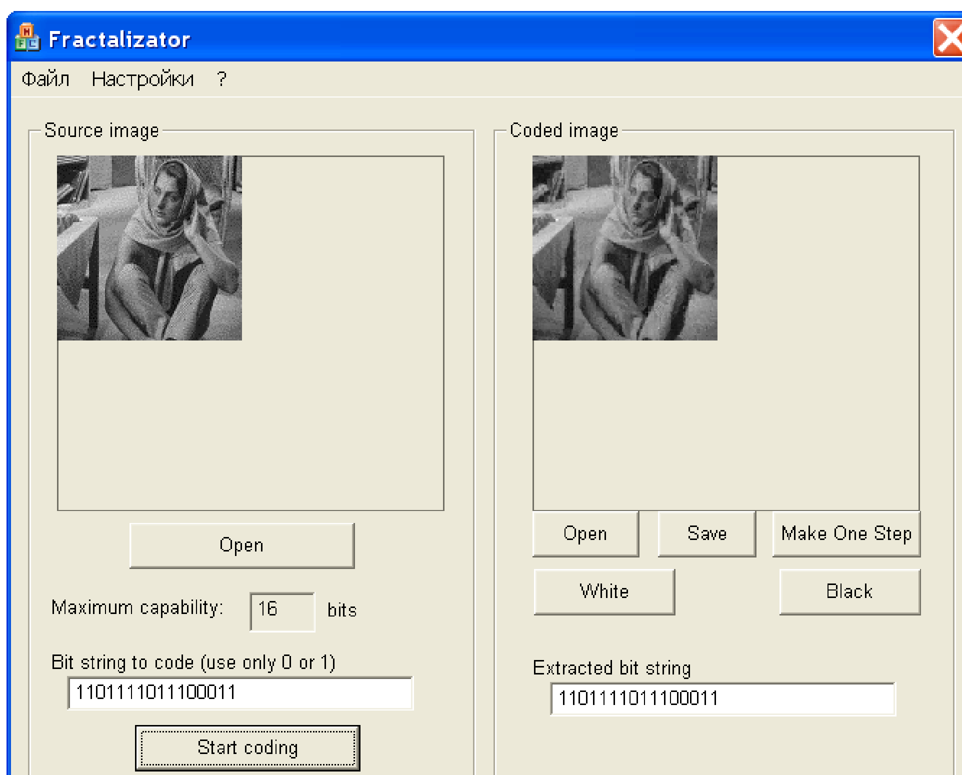


Рис.3.2. Главное окно

Меню главной формы:

- Файл – открыть несжатое изображение - выполняемое действие открытие несжатого изображения в формате BMP;
- Файл – открыть сжатое изображение – выполняемое действие открытие сжатого изображения в формате IFS;
- Файл – Сохранить сжатое изображение – выполняемое действие сохранение сжатого изображения в формате IFS;
- Настройки – настройки сжатия – выполняемое действие вызов окна настроек;
- Файл – выход – выполняемое действие завершение программы;
- ? – О программе – выполняемое действие вызов окна «о программе»

3.4.2. Окно настроек

Окно настроек предназначено для задания настроек сжатия изображения. Все настройки автоматически сохраняются и восстанавливаются из файла fractalizator.ini.

Экранный снимок окна настроек приведен на рис.3.3.

Окно включает в себя следующие элементы управления:

- Кнопки «ОК» и «Отмена» для сохранения и отмены изменений настроек соответственно;
- Текстовое поле для ввода начального размера блока;
- Текстовое поле для ввода максимального уровня разбиения блоков;
- Текстовое поле для ввода пороговой ошибки, при которой осуществляется разбиение блока.

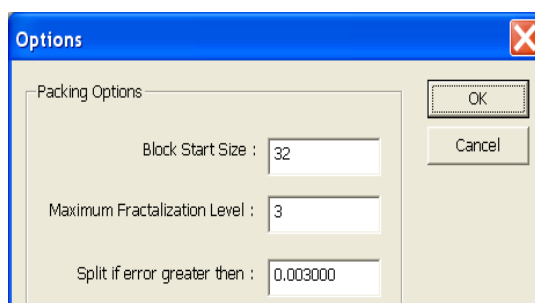


Рис.3.3. Окно настроек

3.4.3. Окно с информацией о программе

Окно с информацией о программе содержит только метки, выводящие различную информацию о программе. Внешний вид окна приведен на рис.3.4.

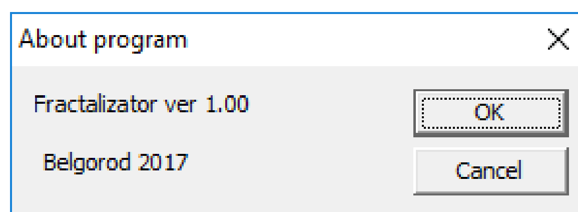


Рис.3.4. Окно настроек

4. ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

После разработки программного обеспечения требуется протестировать его, чтобы убедиться в отсутствии ошибок, а также в наличии всей функциональности, требуемой постановкой задачи.

Проведем последовательно тестирование всех заявленных функций. Для этого предлагается следующий план действий:

- Открытие BMP-файла;
- Ввод битовой строки в соответствующее текстовое поле;
- Запуск процесса кодирования;
- Сохранение закодированного изображения в IFS-файл;
- Сравнение размеров исходного и закодированного файлов;
- Выход из программы;
- Повторный запуск программы;
- Открытие IFS-файла;
- Нажатие по кнопке «Make One Step»;
- Сверить извлеченную битовую строку с исходной;
- Закрасить контейнер сжатого изображения в черный цвет;
- Многократным нажатием на кнопку «Make One Step» проконтролировать процесс восстановления изображения, убедиться, что итерационный процесс сошелся и что конечное изображение близко к исходному;
- Повторить последние два пункта, но с предварительной закраской в белый цвет.

Таким образом, можно удостовериться в корректности работы заявленных функций программного обеспечения. Немаловажное значение имеет тот факт, что процесс восстановления изображения всегда сходится к конечной точке (аттрактору IFS) вне зависимости от исходного изображения.

Таблица 4.1.

Результаты испытания предлагаемого алгоритма

Изображение	Восстановление	Размер файла изначально	Итераций	Коэф-нт сжатия,с	Полный перебор,с	Размер файла после сжатия
Девушка в платке	из абсолютно черного	1.5 мб	8	0,310	1138,86	15,200
Девушка в платке	Из абсолютно белого	1.5 мб	8	0,421	1242,20	15,200
Мона Лиза	из абсолютно черного	4 мб	25	1,946	25386,59	160,200

Последовательность восстановления изображения (не до конца) из абсолютно черного и из абсолютно белого фона показана на рисунках 4.1 и 4.2 соответственно.

Рисунок 4.1. Процесс восстановления изображения из черного фона

После прохождения данного теста разработанное программного обеспечение показало себя полностью работоспособным. Коэффициент сжатия файла составил 0.31, что близко к результату, показанному алгоритмом JPEG.

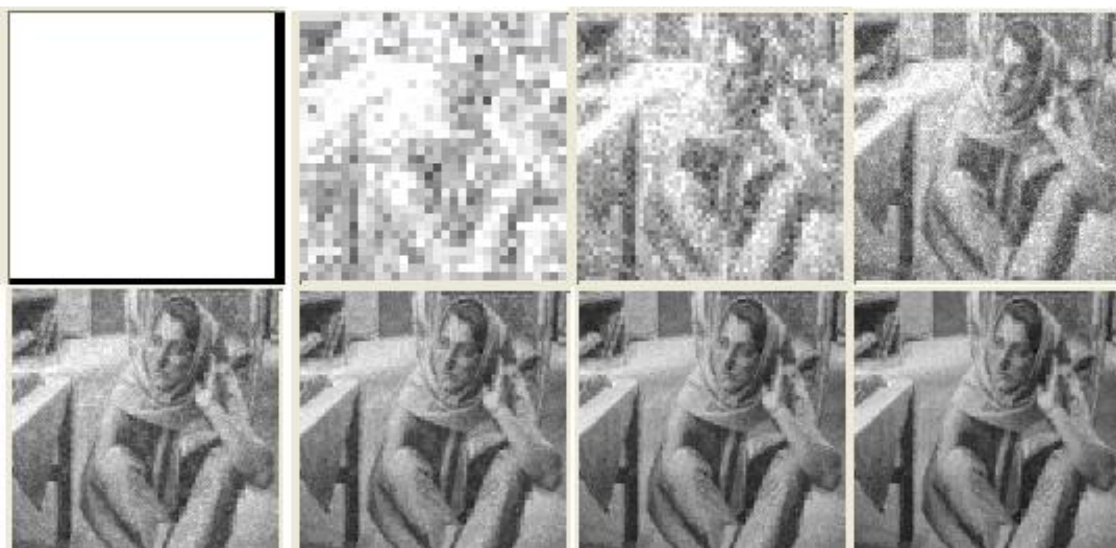


Рисунок 4.2. Процесс восстановления изображения из белого фона

Битовая строка была восстановлена корректно. При этом стоит отметить, что программа всегда восстанавливает количество бит, равное количеству блоков начального размера в исходном изображении. Даже если исходная битовая строка была больше или меньше этого количества. На основании проведенного теста можно сделать вывод о работоспособности программного обеспечения и о соответствии заявленным требованиям.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы были решены все поставленные цели. Было разработано программного обеспечения, реализующее фрактальное сжатие изображения и встраивани извлечения скрытой информации из растрового изображения.

В ходе проведения тестирования была выявлена полная работоспособность разработанного программного обеспечения и соответствие заявленным требованиям. Коэффициент сжатия составил 0.31, битовая строка восстановлена корректно. Исходное изображение на человеческий взгляд полностью идентично восстановленному изображению.

Поскольку основной целью выпускной квалификационной работы было разработка фрактального сжатия изображения и корректное встраивание и извлечение скрытой информации из изображения, коэффициент сжатия не превысил аналогичный коэффициент сжатия, достигнутый JPEG-алгоритмом, однако оказался близок от него.

Этот факт говорит о перспективности и высоком потенциале метода фрактального сжатия изображений. Особенно, если учесть, что в данной работе не были задействованы многие методы увеличения степени сжатия (преобразование в битовую цепочку перед сохранением, обработка однотонных блоков и т.п.).

Внешний вид изображения не меняется при встраивании скрытой информации. Данный метод встраивания цифровых водяных знаков можно с равным успехом применять как для передачи скрытой информации, так и для встраивания информации, необходимой для защиты авторских прав. Этот делает данную разработку перспективной и применимой на практике.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Ватолин Д., Ратушняк А., Смирнов М., Юкин В. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео. - М.: ДИАЛОГ-МИФИ, 2002. - 384 с.

2. Гамма Э., Р.Хелм и другие. Приемы объектно-ориентированного проектирования: Паттерны проектирования. С.-Пб:Питер, Addison-Wesley, издательство ДМК, 2004.

3. Сидоров Д.В., Осокин А.Н. Простой алгоритм ветвления-сжатия полутоновых и цветных изображений // Известия Томского политехнического университета. – 2007 – Т. 311. – № 5. – С. 86-91.

4. Сидоров Д.В. Осокин А.Н. Марков Н.Г. Оценка качества изображений с использованием вейвлетов // Известия Томского политехнического университета. – 2009 – Т.315. – № 5. – С. 104-107.

5. Сэлмон Д. Сжатие данных, изображений и звука Москва:Техносфера, 2006. – 368г.

6. Макаров А.О. Алгоритмы увеличения пространственного разрешения и обработки мультиспектральных спутниковых изображений: Дис... к-та техн. наук: 05.13.01/ А.О. Макаров. –Минск, 2006.

7. Хайтун С.Д. От эргодической гипотезы к фрактальной картине мира: Рождение и осмысление новой парадигмы. Изд. 2-е – М.: ЛЕНАНД, 2016. – 256 с.

8. Цифровая обработка изображений в среде MATLAB Москва: Техносфера, 2006. – 616с.

9. Wang X., Tian B., Liang C., Shi D. Blind Image Quality Assessment for Me asuring Image Blur //Congress on Image and Signal 2008 Congress on Image and Signal Processing, 2008.

10. Балханов В.К. Основы фрактальной геометрии и фрактального исчисле-ния / отв. ред. Ю.Б. Башкуев. – Улан-Удэ: Изд-во Бурятского госуниверситета. 2013. – 224 с.

11. Смирнов В.В., Спиридонов Ф.Ф. Моделирование фракталов в Maple. – Бийск: БТИ АлтГТУ. 2005. – 93 с.

ПРИЛОЖЕНИЯ

Приложение 1. Текст программы

Листинг А.1 – Заголовочный файл IFS.h

```
#pragma once
#include "stdafx.h"

class cOptions;

class cIFS
{
	CImage *image;
	cOptions *Options;
	domain_vector domains[10], blocks[10];
	int maxFract;		//Максимальный уровень фрактализации

	int width, height;
	int startBlockSize;

	HRESULT CheckImage(CImage *image);
	void GetDomains(int size, int step, domain_vector &vect, bool scale);
	void splitBlock(domain_vector &vect, cDomain *block, CImage *image);

public:
	HRESULT Open(char *fileName);
	HRESULT Save(char *fileName);
	HRESULT Clear();
	HRESULT GetFromImage(CImage *image, LPTSTR codestring);
	HRESULT MakeStep(CImage *image1, CImage *image2, LPTSTR
*codestring);

	int getWidth()
		{	return width;		}
	int getHeight()
		{	return height;		}

	cOptions *getOptions()
		{	return Options;		}

	cIFS();
```

```
	~cIFS();
};
```

Листинг А.2 – Модуль IFS.cpp

```
#include "stdafx.h"
#include "IFS.h"
#include "Options.h"
#include "domain.h"
#include "conversions.h"

cConversionSet *conv;

cIFS::cIFS()
{
	Options = new cOptions();
	Options->SetDefaults();
	Options->GetOptions();
	conv = new cConversionSet();
	maxFract = 0;
	width = height = 128;
}

cIFS::~cIFS()
{
	Clear();
}

HRESULT cIFS::Open(char *fileName)
{
	Clear();
	FILE *f = fopen(fileName,"rb");

	unsigned char byte;

	if (f)  {
		////////////////
		// Header
		//Уровень фрактализации
		fread((void*) &byte,1,1,f);
		maxFract = (int) byte;
		//Стартовый размер ячейки
		fread((void*) &byte,1,1,f);
		startBlockSize = (int) byte;
		//Ширина
		fread((void*) &width,1,2,f);
```

```
		//Высота
		fread((void*) &height,1,2,f);
		// Header
		////////////////

		int s = startBlockSize;

		//Нахождение начального массива
		GetDomains(s, s, blocks[0], false);

		//Нахождение всех необходимых массивов доменов
		for (int i = 0; i <= maxFract; i++, s /= 2)	{
			GetDomains(2*s, s, domains[i], true);
		}
		s = startBlockSize;

		domain_vector::iterator begin,end,iter;
		iter = begin = blocks[0].begin();
		end = blocks[0].end();
		int xSize, ySize;
		xSize =  width/s;
		ySize =  height/s;

		for (int y = 0; y < ySize; y++)	{
			for (int x = 0; x < xSize; x++)	{
				assert(iter != end);
				cDomain *block = *iter;
				assert(block);
				block-
>ReadFromFile(f,x,y,s,s,domains,blocks,0,width,height);
				iter++;
			}
		}

		fclose(f);
		return S_OK;
	}
	return E_FAIL;
}

HRESULT cIFS::Save(char *fileName)
{
	FILE *f = fopen(fileName,"wb");

	if (f)	{
```

```
		unsigned char byte;
		////////////////
		// Header
		//Уровень фрактализации
		byte = (unsigned char) this->maxFract;
		fwrite((void*) &byte,1,1,f);

		//Стартовый размер ячейки
		byte = (int) startBlockSize;
		fwrite((void*) &byte,1,1,f);
		//Ширина
		fwrite((void*) &width,1,2,f);
		//Высота
		fwrite((void*) &height,1,2,f);
		// Header
		////////////////

		domain_vector::iterator begin,end,iter;
		begin = blocks[0].begin();
		end = blocks[0].end();
		for (iter = begin; iter != end; iter++)	{
			cDomain *block = *iter;
			block->SaveToFile(f, blocks, 0);
		}

		fclose(f);
		return S_OK;
	}
	return E_FAIL;
}

HRESULT cIFS::CheckImage(CImage *image)
{
	if ( image->GetWidth()%startBlockSize )	{
		return E_FAIL;
	}
	if ( image->GetHeight()%startBlockSize )	{
		return E_FAIL;
	}
	return S_OK;
}

void cIFS::GetDomains(int size, int step, domain_vector &vect, bool scale)
{
	int w = width;		//image->GetWidth();
```

```
	int h = height;		//image->GetHeight();

	for (int y = 0; y*step + size <= h; y++)		{
		for (int x = 0; x*step + size <= w; x++)		{
			cDomain *block = new cDomain(Options, image, size, step, x,
y);
			vect.insert(vect.end(), block);
			if (scale) block->ScaleDown();
		}
	}
}

void cIFS::splitBlock(domain_vector &vect, cDomain *block, CImage *image)
{
	cDomain *b;
	int newSize = block->getSize()/2;
	for (int y = 0; y < 2; y++)		{
		for (int x = 0; x < 2; x++)		{
			b = new cDomain(Options, image, newSize, newSize,
				x + block->getX()*2, y + block->getY()*2);
			vect.insert(vect.end(), b);
		}
	}
}

HRESULT cIFS::GetFromImage(CImage *image, LPTSTR codestring)
{
	HRESULT hr;

	startBlockSize = Options->blockSize;

	hr = CheckImage(image);
	if FAILED(hr) return hr;

	height = image->GetHeight();
	width = image->GetWidth();

	Clear();

	this->image = image;

	int s = startBlockSize;

	//Количество бит, которые осталось закодировать
```

```
	int toCode = (image->GetWidth()*image->GetHeight())/(Options-
>blockSize*Options->blockSize);
	if (toCode > (int) strlen(codestring)) toCode = (int) strlen(codestring);
	int totalCodeLen = toCode;

	//Нахождение начального массива блоков
	GetDomains(s, s, blocks[0], false);

	//Запускаем цикл по всем уровням фрактализации
	for (int i = 0; i <= Options->iMax; i++, s /= 2)	{
		if (blocks[i].empty()) break;

		//Получаем список доменов для данного уровня фрактализации
		maxFract = i;
		GetDomains(2*s, s, domains[i], true);

		domain_vector::iterator dBeg, dEnd, dIter;
		domain_vector::iterator bBeg, bEnd, bIter;

		bBeg = blocks[i].begin();
		bEnd = blocks[i].end();

		dBeg = domains[i].begin();
		dEnd = domains[i].end();


		//Проводим поиск для каждого блока
		for (bIter = bBeg; bIter != bEnd; bIter++)	{
			cDomain *block = *bIter;

			//Максимальное недостижимое значение RMS
			float bestRMS = 65536.0*1024.0*1024.0f;
			int bestCode;
			int best_dk;
			float best_ak;
			cDomain *bestDomain = 0;

			enum eFilter {
				efEven = 0,
				efOdd = 1,
				efNoFilter = 2
			} filter = efNoFilter;

			//if (i == 0)
			{
```

```
			if (toCode > 0)	{
				char ch = codestring[ totalCodeLen - toCode];
				assert( (ch == '1') || (ch == '0'));

				if (ch == '0') filter = efEven;
				else filter = efOdd;
			}
		}

		//Проводим поиск наиболее подходящего домена
		for (dIter = dBeg; dIter != dEnd; dIter++)	{

			cDomain *domain = *dIter;

			bool skipDomain = false;
			if (filter != efNoFilter)	{
				int even = (domain->getX() + domain->getY())%2;
				if (even != filter) skipDomain = true;
			}

			if (!skipDomain)	{
				int y = block->getAverage();
				int x = domain->getAverage();
				float ak = (float) domain->getFluctuation()/
					block->getFluctuation();

				if (ak > 1)	ak = 0.95f;

				int dk = (int) (y - x*ak);

				cDomain *block2 = new cDomain(domain,dk,ak);
				for (int k = 0; k < 8; k++)		{
					cDomain *block3 = new cDomain(block2,
k);

					float RMS = block3->getRMS(block);
					if (RMS < bestRMS)
					{
						bestRMS = RMS;
						bestCode = k;
						bestDomain = domain;
						best_dk = dk;
						best_ak = ak;
					}
					delete block3;
```

```
                                        }
                                        delete block2;
                                }
                        }
                        assert(bestDomain);
                        if ((bestRMS <= Options->eps)||(i == Options->iMax))   {
                                block->SetSource(bestDomain, bestCode, best_dk,
best_ak);
                                if (filter != efNoFilter) toCode --;
                        }
                        else    {
                                splitBlock(blocks[i+1],block,image);
                        }

                }
        }
        return hr;
}

HRESULT cIFS::MakeStep(CImage *image1, CImage *image2, LPTSTR
*codestring)
{
        domain_vector::iterator begin, end, iter;

        //Инициализируем все домены всех уровней растром из исходного
изображения
        for (int i = 0; i <= maxFract; i++)       {
                begin = domains[i].begin();
                end = domains[i].end();

                for (iter = begin; iter != end; iter++)   {
                        cDomain* domain = *iter;
                        domain->SetRaster(image1, true);
                }
        }

        //Количество закодированных бит
        int toCode = (image1->GetWidth()*image1->GetHeight())/(Options-
>blockSize*Options->blockSize);
        (*codestring) = new char[toCode + 1];
        int nextchar = 0;

        for (int i = 0; i <= maxFract; i++)       {
                begin = blocks[i].begin();
                end = blocks[i].end();
```

```
		for (iter = begin; iter != end; iter++)	{
			cDomain* block = *iter;
			if (block->IsReady())	{
				block->GetRaster(image2);

				int even = block->getSourceEven();
				if (nextchar < toCode)	{
					(*codestring)[nextchar] = (even == 0) ? '0' : '1';
					nextchar++;
				}
			}

		}
	}
	(*codestring)[toCode] = 0;
	return S_OK;
}

HRESULT cIFS::Clear()
{
	for (int i = 0; i < 10; i++) {
		domain_vector::iterator begin, end, iter;

		begin = domains[i].begin();
		end = domains[i].end();
		for (iter = begin; iter != end; iter++)	{
			cDomain* domain = *iter;
			delete domain;
		}
		domains[i].clear();

		begin = blocks[i].begin();
		end = blocks[i].end();
		for (iter = begin; iter != end; iter++)	{
			cDomain* block = *iter;
			delete block;
		}
		blocks[i].clear();
	}
	maxFract = 0;
	image = NULL;
	return S_OK;
}
```

Листинг А.3 – Заголовочный файл domain.h

```
#pragma once
#include "stdafx.h"
#include "conversions.h"

class cOptions;

extern cConversionSet *conv;

class cDomain
{
	//static cConversionSet *conv;
	cOptions *Options;
	int x,y;
	int size, step;
	unsigned char *data;

	unsigned char average;
	float fluctuation;

	void CalcCharacteristics();

	cDomain *sourceDomain;
	unsigned char sourceCode;
	short int source_dk;
	float source_ak;
public:
//////////////
//GETS SECTION
	unsigned char *getData()
		{	return data;	}
	int getX()
		{	return x;	}
	int getY()
		{	return y;	}
	int getSize()
		{	return size;	}
	cOptions *getOptions()
		{	return Options;	}
	unsigned char getAverage()
		{	return average;	}
	float getFluctuation()
		{	return fluctuation;	}
	bool IsReady()
		{	return sourceDomain != NULL;	}
	int getSourceEven()
```

```
		{	return (sourceDomain->getX() + sourceDomain->getY())%2;	}
//GETS SECTION
//////////////

	void fillData(int &x, int &y, int &size, int &step);

	void SetSource(cDomain *domain, int code, int dk, float ak);

	HRESULT ScaleDown();			//Сжимает домен в два раза
	float getRMS(cDomain *domain);

	void SetRaster(CImage* image, bool scale);
	void GetRaster(CImage* image);

	bool SaveToFile(FILE *f, domain_vector vector[], int i );
	bool SaveChildren(FILE *f, domain_vector vector[], int i );
	bool ReadFromFile(FILE *f, int x, int y, int size, int step, domain_vector
domains[], domain_vector blocks[], int i, int w, int h);
	bool ReadChildren(FILE *f, domain_vector domains[], domain_vector
blocks[], int i, int w, int h);

	//Конструктор по данным изображения
	cDomain(cOptions *Options, CImage *image, int size,
		int step, int x, int y);

	//Конструктор по сдвигу яркости и множителю контрастности
	cDomain(cDomain* domain, int dk, float ak);

	//Конструктор по коду преобразования
	cDomain(cDomain* domain, int code);

	~cDomain();
};
```

Листинг А.4 – Модуль domain.cpp

```
#include "stdafx.h"
#include "domain.h"
#include "ifs.h"
#include "options.h"
#include "math.h"

cDomain::~cDomain()
{
	delete[] data;
}
```

```
cDomain::cDomain(cOptions *Options, CImage *image,
        int size, int step, int x, int y)
{
        this->Options = Options;
        this->size = size;
        this->step = step;
        this->x = x;
        this->y = y;

        sourceDomain = 0;

        data = 0;

        if (image)    {
                SetRaster(image, false);
                CalcCharacteristics();
        }
}

cDomain::cDomain(cDomain* domain, int dk, float ak)
{
        domain->fillData(x,y,size,step);
        Options = domain->getOptions();

        sourceDomain = 0;

        data = new unsigned char[size*size];
        unsigned char *data2 = domain->getData();

        for (int offs = 0; offs < size*size; offs ++)    {
                int point = (unsigned int)
                        (data2[offs] * ak + dk);
                data[offs] = (point > 0xFF) ? 0xFF :
                        (point < 0x00) ? 0 : point;

                /*
                int point = (unsigned int)
                        ((data2[offs] * ak) + (data2[offs] + dk)) / 2.0f;
                data[offs] = (point > 0xFF) ? 0xFF :
                        (point < 0x00) ? 0 : point;
                */
        }
        CalcCharacteristics();
}
```

```
//Конструктор по коду преобразования
cDomain::cDomain(cDomain* domain, int code)
{
	domain->fillData(x,y,size,step);
	Options = domain->getOptions();

	sourceDomain = 0;

	data = new unsigned char[size*size];
	unsigned char *data2 = domain->getData();

	for (int yy = 0; yy < size; yy++)	{
		for (int xx = 0; xx < size; xx++)	{
			int destOffs = yy*size + xx;
			int srcX, srcY;
			conv->GetCoords(xx, yy, srcX, srcY, size, code);
			int srcOffs = srcY*size + srcX;
			data[destOffs] = data2[srcOffs];
		}
	}
	CalcCharacteristics();
}

void cDomain::fillData(int &x, int &y, int &size, int &step)
{
	x = this->x;
	y = this->y;
	size = this->size;
	step = this->step;
}

void cDomain::CalcCharacteristics()
{
	unsigned long s = 0;
	for (int i = 0; i< size*size; i++)
	{
		s += data[i];
	}
	average = (int) s/(size*size);

	s = 0;
	for (int i = 0; i< size*size; i++)
	{
		int d = (data[i] - average);
```

```
			//int d = (255 - data[i]);
			s += d*d;
		}
		fluctuation = sqrt((float)s) /(size*size);
}

HRESULT cDomain::ScaleDown()
{
		unsigned char *old = data;

		assert((size) && (size%2 == 0));

		if ((!size) || (size%2 != 0))
			return E_FAIL;

		size >>= 1;

		//Нормальная ситуация
		if (!data) return S_FALSE;

		data = new unsigned char[size*size];

		for (int i = 0; i<size; i++){
			int y = i*2;
			for (int j = 0; j<size; j++){
				int x = j*2;
				int offset = y*size*2 + x;

				unsigned int d = old[offset];
				d += old[offset + 1];
				d += old[offset + 2*size];
				d += old[offset + 2*size + 1];

				d >>= 2;
				data[i*size + j] = d;
			}
		}

		delete[] old;

		return S_OK;
}

float cDomain::getRMS(cDomain *domain)
{
```

```
	{
		int sz = domain->getSize();
		assert(sz == size);
	}
	float RMS = 0;

	unsigned char* data2 = domain->getData();
	for (int offs = 0; offs < size*size; offs++)
	{
		int diff = *(data + offs) - *(data2 + offs);
		RMS += (float) diff*diff;
	}
	RMS = (float) sqrt(RMS)/(size*size*256);

	return RMS;
}

void cDomain::SetSource(cDomain *domain, int code, int dk, float ak)
{
	sourceDomain = domain;
	sourceCode = code;
	source_dk = dk;
	source_ak = ak;
	CalcCharacteristics();
}

void cDomain::SetRaster(CImage *im, bool scale)
{
	if (data) delete[] data;

	if (scale) size <<= 1;

	unsigned char *buf = (unsigned char *) im->GetBits();
	int pitch = (im->GetPitch());

	//if (pitch<0) buf += im->GetWidth()*im->GetHeight();

	int offs = step*(y*pitch + x);
	data = new unsigned char[size*size];

	for (int yy = 0; yy < size; yy++)
	{
		memcpy( (void*)(data + yy*size),
				(void*)(buf + offs + yy*pitch), size);
	}
```

```
        if (scale) ScaleDown();
}

void cDomain::GetRaster(CImage *im)
{
        cDomain *d1 = new cDomain(sourceDomain,source_dk, source_ak);
        cDomain *d2 = new cDomain(d1, sourceCode);

        unsigned char *data2 = d2->getData();

        unsigned char *buf = (unsigned char *) im->GetBits();
        int pitch = im->GetPitch();

        int offs = step*(y*pitch + x);

        for (int yy = 0; yy < size; yy++)
        {
                memcpy( (void*)(buf + offs + yy*pitch),
                                (void*)(data2 + yy*size), size);
        }
        delete d1;
        delete d2;
}

bool cDomain::ReadFromFile(FILE *f, int x, int y, int sz, int st, domain_vector
domains[], domain_vector blocks[], int i, int w, int h)
{
//      char s_byte;
        unsigned char byte;

        this->x = x;
        this->y = y;
        this->size = sz;
        this->step = st;

        fread((void*) &byte,1,1,f);
        if (byte == 0xFF)  {
                sourceDomain = NULL;
                ReadChildren(f,domains,blocks, i, w, h);
                return true;
        }

        sourceCode = byte;
```

```
	fread((void*) &source_dk, 2, 1,f);
	fread((void*) &source_ak, 4, 1, f);

	int sx,sy;
	fread((void*) &byte,1,1,f);
	sx = (int) byte;
	fread((void*) &byte,1,1,f);
	sy = (int) byte;

	// у блока и соответствующего ему домена step
	// совпадают
	int offs = sy*((w - size*2)/step + 1) + sx;
	domain_vector::iterator begin,iter,end;
	iter = begin = domains[i].begin();
	end = domains[i].end();
	for (int j = 0; j < offs; j++)	{
		assert(iter != end);
		iter++;
	}
	sourceDomain = *iter;
	assert(sourceDomain);

	return true;
}

bool cDomain::ReadChildren(FILE *f, domain_vector domains[], domain_vector
blocks[], int i, int w, int h)
{
	for (int yy = y*2; yy < y*2 + 2; yy++)	{
		for (int xx = x*2; xx < x*2 + 2; xx++)	{
			cDomain *newBlock = new
cDomain(Options,NULL,size/2,step/2,xx,yy);
			blocks[i+1].insert(blocks[i+1].end(), newBlock);
			newBlock->ReadFromFile(f,xx,yy,size/2,step/2, domains,
blocks, i+1, w,h);
		}
	}
	return true;
}

bool cDomain::SaveToFile(FILE *f, domain_vector vector[], int i )
{
//	char s_byte;
	unsigned char byte;
```

```
	if (!IsReady())	{
		byte = 0xFF;		//Разбитый блок
		fwrite((void*) &byte, 1, 1, f);
		SaveChildren(f, vector, i);
		return true;
	}

	byte = sourceCode;
	fwrite((void*) &byte, 1, 1, f);

	//s_byte = (char) source_dk;
	fwrite((void*) &source_dk, 2, 1, f);
	fwrite((void*) &source_ak, 4, 1, f);

	assert(sourceDomain);
	byte = (unsigned char) sourceDomain->x;
	fwrite((void*) &byte, 1, 1, f);

	byte = (unsigned char) sourceDomain->y;
	fwrite((void*) &byte, 1, 1, f);

	return true;
}

bool cDomain::SaveChildren(FILE *f, domain_vector vector[], int i )
{
	domain_vector::iterator begin, end, iter;
	begin = vector[i+1].begin();
	end = vector[i+1].end();
	for (int yy = y*2; yy < y*2 + 2; yy++)		{
		for (int xx = x*2; xx < x*2 + 2; xx++)		{
			bool finded = false;
			for (iter = begin; iter != end; iter++)	{
				cDomain *domain = *iter;
				if ( (domain->getX() == xx)&&(domain->getY() == yy))
	{
					finded = true;
					domain->SaveToFile(f, vector, i+1);
					break;
				}
			}
			assert(finded);
		}
	}
	return true;
```

```
}
```

Листинг А.5 – Заголовочный файл options.h

```
#pragma once

#define iniFileName ("./packing.ini")
#define Section ("PACKING")

class cOptions
{
public:
//Methods
	HRESULT SetDefaults();
	HRESULT GetOptions();
	HRESULT StoreOptions();
//Options
	int blockSize;
	float eps;
	int iMax;

	cOptions();
	~cOptions();
};
```

Листинг А.5 – Модуль options.cpp

```
#include "stdafx.h"
#include "options.h"

cOptions::cOptions()
{
	;
}

cOptions::~cOptions()
{
	;
}

HRESULT cOptions::SetDefaults()
{
	blockSize = 32;
	eps = 0.003f;
	iMax = 3;

	return S_OK;
```

```
}

HRESULT cOptions::GetOptions()
{
	int ppd;	//PrivateProfileData
	int len;
	char pps[20];

	ppd = (int) GetPrivateProfileInt(Section, "BlockSize", -1, iniFileName);
	if (ppd > 0)  {
		blockSize = ppd;
	}

	ppd = (int) GetPrivateProfileInt(Section, "FractLevel", -1, iniFileName);
	if (ppd >= 0)	{
		iMax = ppd;
	}

	len = (int) GetPrivateProfileString(Section, "Epsilon", "", pps, 10, 
iniFileName);
	if (len > 0)   {
		eps = (float) atof(pps);
	}

	return S_OK;
}

HRESULT cOptions::StoreOptions()
{
	BOOL res;
	char pps[20];			//PrivateProfileString

	try	{
		sprintf(pps,"%d", this->blockSize);
		res = WritePrivateProfileString(Section, "BlockSize", pps, 
iniFileName);
		if (!res) throw(0);

		sprintf(pps,"%d", this->iMax);
		res = WritePrivateProfileString(Section, "FractLevel", pps, 
iniFileName);
		if (!res) throw(0);

		sprintf(pps,"%f", this->eps);
		res = WritePrivateProfileString(Section, "Epsilon", pps, iniFileName);
```

```
		if (!res) throw(0);
	}
	catch(...)	{
		return E_FAIL;
	}
	return S_OK;
}
```

Листинг А.7 – Заголовочный файл conversuions.h

```
#pragma once

enum eCoord	{
	NEG_X, NEG_Y, POS_X, POS_Y
};

enum eAdder	{
	ADD_ZERO, ADD_N_MINUS_1
};

struct sConv
{
	eCoord xCoord, yCoord;
	eAdder xAdder, yAdder;
};

class cConversionSet
{
	sConv conversions[8];
public:
	void GetCoords(const int tarX, const int tarY,
		int &srcX, int &srcY, const int n, const int code );
	cConversionSet();
};
```

Листинг А.8 – Модуль conversions.cpp

```
#include "stdafx.h"
#include "conversions.h"

cConversionSet::cConversionSet()
{
	//Преобразование 0 - оставляет без изменений
	//a[x,y] = a[x, y]
	conversions[0].xCoord = POS_X;
	conversions[0].xAdder = ADD_ZERO;
	conversions[0].yCoord = POS_Y;
```

```
conversions[0].yAdder = ADD_ZERO;

//Преобразование 1 - отражение от горизонтальной оси
//a[x,y] = a[x, n-1-y]
conversions[1].xCoord = POS_X;
conversions[1].xAdder = ADD_ZERO;
conversions[1].yCoord = NEG_Y;
conversions[1].yAdder = ADD_N_MINUS_1;

//Преобразование 2 - отражение от вертикальной оси
//a[x,y] = a[n-1-x, y]
conversions[2].xCoord = NEG_X;
conversions[2].xAdder = ADD_N_MINUS_1;
conversions[2].yCoord = POS_Y;
conversions[2].yAdder = ADD_ZERO;

//Преобразование 3 - отражение от главной диагонали
//a[x,y] = a[y, x]
conversions[3].xCoord = POS_Y;
conversions[3].xAdder = ADD_ZERO;
conversions[3].yCoord = POS_X;
conversions[3].yAdder = ADD_ZERO;

//Преобразование 4 - отражение от побочной диагонали
//a[x,y] = a[n-1-y, n-1-x]
conversions[4].xCoord = NEG_X;
conversions[4].xAdder = ADD_N_MINUS_1;
conversions[4].yCoord = NEG_Y;
conversions[4].yAdder = ADD_N_MINUS_1;

//Преобразование 5 - поворот на -90
//a[x,y] = a[y, n-1-x]
conversions[5].xCoord = POS_Y;
conversions[5].xAdder = ADD_ZERO;
conversions[5].yCoord = NEG_X;
conversions[5].yAdder = ADD_N_MINUS_1;

//Преобразование 6 - поворот на 180
//a[x,y] = a[n-1-x, n-1-y]
conversions[6].xCoord = NEG_Y;
conversions[6].xAdder = ADD_N_MINUS_1;
conversions[6].yCoord = NEG_X;
conversions[6].yAdder = ADD_N_MINUS_1;

//Преобразование 7 - поворот на 90
```

```
        //a[x,y] = a[n-1-y, x]
        conversions[7].xCoord = NEG_Y;
        conversions[7].xAdder = ADD_N_MINUS_1;
        conversions[7].yCoord = POS_X;
        conversions[7].yAdder = ADD_ZERO;
}

void cConversionSet::GetCoords(const int tarX, const int tarY,
                int &srcX, int &srcY, const int n, const int code )
{
        /*
        srcX = tarX;
        srcY = tarY;
        return;
        */

        switch (conversions[code].xCoord)    {
                case POS_X:
                        srcX = tarX;
                        break;
                case POS_Y:
                        srcX = tarY;
                        break;
                case NEG_X:
                        srcX = -tarX;
                        break;
                case NEG_Y:
                        srcX = -tarY;
                        break;
        }
        if (conversions[code].xAdder == ADD_N_MINUS_1)
                srcX += n-1;


        switch (conversions[code].yCoord)    {
                case POS_X:
                        srcY = tarX;
                        break;
                case POS_Y:
                        srcY = tarY;
                        break;
                case NEG_X:
                        srcY = -tarX;
                        break;
                case NEG_Y:
```

```
                        srcY = -tarY;
                        break;
        }
        if (conversions[code].yAdder == ADD_N_MINUS_1)
                srcY += n-1;

}
```

Листинг А.9 – Заголовочный файл stdafx.h

```
        // stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently,
// but are changed infrequently

#pragma once

#include "common.h"

#ifndef VC_EXTRALEAN
#define VC_EXTRALEAN            // Exclude rarely-used stuff from Windows
headers
#endif

// Modify the following defines if you have to target a platform prior to the ones
specified below.
// Refer to MSDN for the latest info on corresponding values for different
platforms.
#ifndef WINVER                  // Allow use of features specific to Windows
95 and Windows NT 4 or later.
#define WINVER 0x0400           // Change this to the appropriate value to
target Windows 98 and Windows 2007 or later.
#endif

#ifndef _WIN32_WINNT            // Allow use of features specific to Windows
NT 4 or later.
#define _WIN32_WINNT 0x0400             // Change this to the appropriate value
to target Windows 98 and Windows 2007 or later.
#endif

#ifndef _WIN32_WINDOWS          // Allow use of features specific to Windows
98 or later.
#define _WIN32_WINDOWS 0x0410 // Change this to the appropriate value to
target Windows Me or later.
#endif
```

```
#ifndef _WIN32_IE                    // Allow use of features specific to IE 4.0 or later.
#define _WIN32_IE 0x0400    // Change this to the appropriate value to target IE 5.0 or later.
#endif

#define _ATL_CSTRING_EXPLICIT_CONSTRUCTORS      // some CString constructors will be explicit

// turns off MFC's hiding of some common and often safely ignored warning messages
#define _AFX_ALL_WARNINGS

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxdisp.h>        // MFC Automation classes


#include <afxdtctl.h>               // MFC support for Internet Explorer 4 Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>                      // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT

#include "atlimage.h"
#include "assert.h"

#include <vector>
#include <map>
#include <afxdhtml.h>

//My STL typedefs
class cDomain;
typedef std::vector<cDomain*> domain_vector;
```

Листинг А.10 – Модуль stdafx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
// Fractalizator.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

Листинг А.11 – Заголовочный файл Fractalizator.h

```
// Fractalizator.h : main header file for the PROJECT_NAME application
//

#pragma once

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols


// CFractalizatorApp:
// See Fractalizator.cpp for the implementation of this class
//

class CFractalizatorApp : public CWinApp
{
public:
	CFractalizatorApp();

// Overrides
	public:
	virtual BOOL InitInstance();

// Implementation

	DECLARE_MESSAGE_MAP()
};

extern CFractalizatorApp theApp;
```

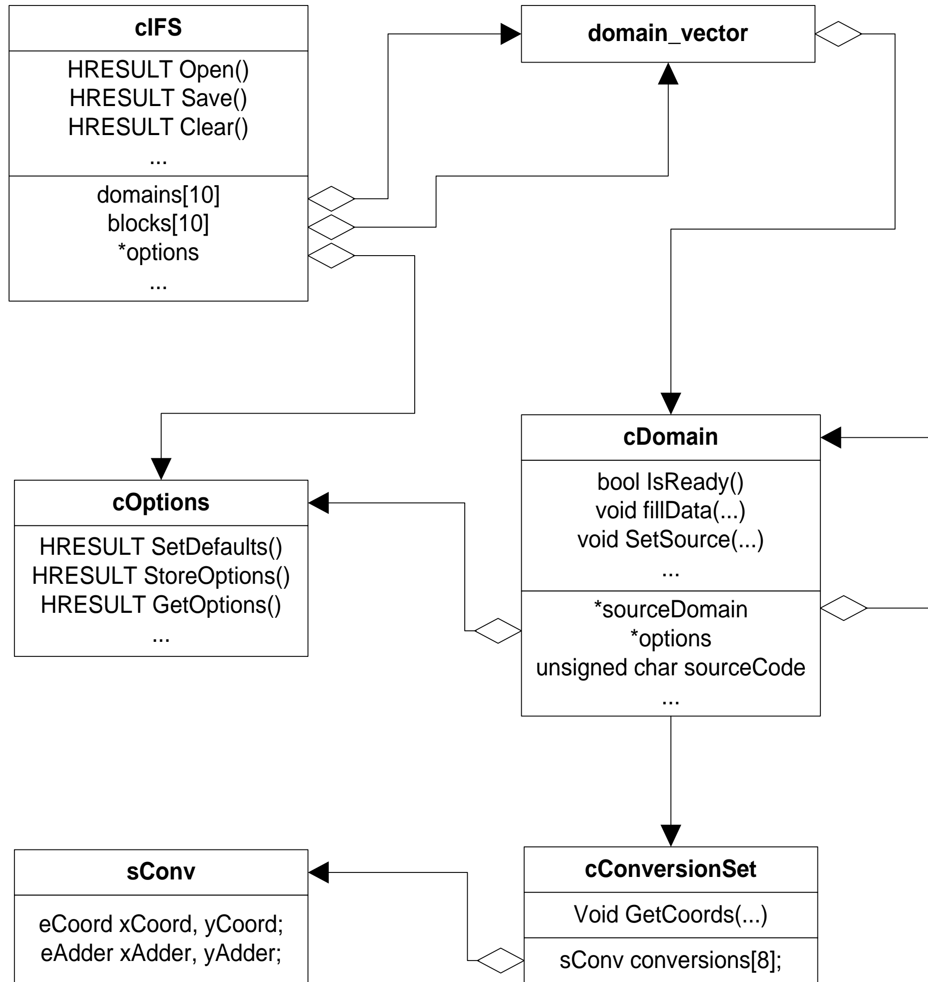
Листинг А.12 – Модуль Fractalizator.cpp

```
// Fractalizator.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "Fractalizator.h"
#include "FractalizatorDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif
```

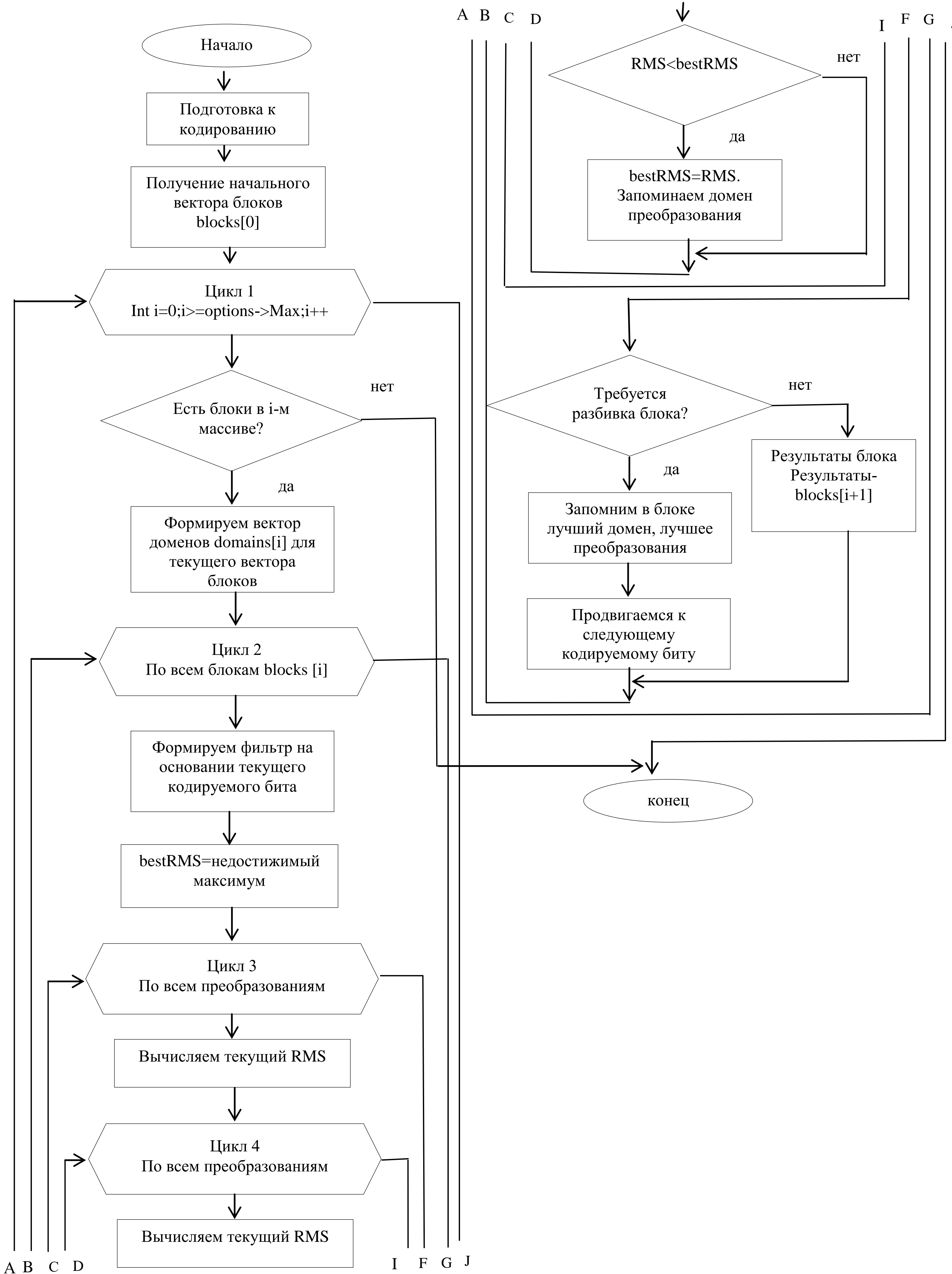
```
// CFractalizatorApp

BEGIN_MESSAGE_MAP(CFractalizatorApp, CWinApp)
	ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()


// CFractalizatorApp construction

CFractalizatorApp::CFractalizatorApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}


// The one and only CFractalizatorApp object

CFractalizatorApp theApp;


// CFractalizatorApp initialization

BOOL CFractalizatorApp::InitInstance()
{
	// InitCommonControls() is required on Windows XP if an application
	// manifest specifies use of ComCtl32.dll version 6 or later to enable
	// visual styles.  Otherwise, any window creation will fail.
	InitCommonControls();

	CWinApp::InitInstance();

	AfxEnableControlContainer();

	// Standard initialization
	// If you are not using these features and wish to reduce the size
	// of your final executable, you should remove from the following
	// the specific initialization routines you do not need
	// Change the registry key under which our settings are stored
	// TODO: You should modify this string to be something appropriate
	// such as the name of your company or organization
	SetRegistryKey(_T("Local AppWizard-Generated Applications"));

	CFractalizatorDlg dlg;
	m_pMainWnd = &dlg;
```

```
	INT_PTR nResponse = dlg.DoModal();
	delete dlg.IFS;
	if (nResponse == IDOK)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with OK
	}
	else if (nResponse == IDCANCEL)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with Cancel
	}

	// Since the dialog has been closed, return FALSE so that we exit the
	//  application, rather than start the application's message pump.
	return FALSE;
}
```

Приложение 2. Структурная схема программного обеспечения



Приложение 3. Схема алгоритма компрессии



Приложение 4. Схема алгоритма декомпрессии