

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

Институт информатики, математики и электроники
Факультет информатики
Кафедра информационных систем и технологий

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

«ИССЛЕДОВАНИЕ АЛГОРИТМОВ ФРАКТАЛЬНОГО СЖАТИЯ
ИЗОБРАЖЕНИЙ»

по направлению подготовки 09.04.01 Информатика
и вычислительная техника
(уровень магистратуры)
профиль «Программное обеспечение мобильных устройств»

Студент _____ В.Б. Сахибназарова
(подпись, дата)

Руководитель ВКР,
к.т.н., доцент кафедры ИСТ _____ М.А. Кудрина
(подпись, дата)

Самара 2018

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

Институт информатики, математики и электроники
Факультет информатики
Кафедра информационных систем и технологий

УТВЕРЖДАЮ

Заведующий кафедрой ИСТ

_____ С.А. Прохоров

« ____ » _____ 20__ г.

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ МАГИСТРА

студенту(ке) _____ Сахибназаровой Виктории Бахтиёровне _____

группа № 6222-090401D _____

Тема работы: Исследование алгоритмов фрактального сжатия
изображений _____

Исходные данные к работе:

- 1) объект исследования: алгоритмы фрактального сжатия изображений; _____
- 2) язык программирования: C#; _____
- 3) среда программирования Microsoft Visual Studio 2015. _____
- 4) операционная система: Windows _____

Перечень вопросов, подлежащих разработке в работе:

- 1) анализ предметной области: изучение основного алгоритма фрактального сжатия изображения; изучение различных вариантов реализации основного алгоритма; изучение модификаций алгоритма, направленных на повышение скорости фрактального сжатия;
- 2) разработка проекта системы по методологии UML;
- 4) разработка программного обеспечения, реализующего алгоритмы фрактального сжатия; проведение тестирования и отладки;
- 5) исследование зависимости скорости сжатия исходного изображения и качества декодируемого изображения от использованного алгоритма;
- 6) оформление документации ВКР.

Руководитель работы

к.т.н., доцент кафедры ИСТ _____ М.А. Кудрина
(подпись)

« ____ » _____ 20 ____ г.

Задание принял к исполнению _____ В.Б. Сахибназарова
(подпись)

« ____ » _____ 20 ____ г.

РЕФЕРАТ

Пояснительная записка 86 страниц, 33 рисунка, 27 таблиц, 20 источников, 2 приложения.

Презентация: 23 слайда Microsoft Power Point.

ФРАКТАЛЬНОЕ СЖАТИЕ ИЗОБРАЖЕНИЙ, КОМПРЕССИЯ, ДЕКОМПРЕССИЯ, РАНГОВЫЙ БЛОК, ДОМЕННЫЙ БЛОК, СРЕДНЕКВАДРАТИЧЕСКОЕ ОТКЛОНЕНИЕ

Целью выпускной работы является исследование вариантов реализации классического алгоритма фрактального сжатия изображений, а также изучение методов повышения скорости фрактального сжатия.

В рамках данной работы проведено исследование зависимости времени сжатия изображения от используемого варианта алгоритма, от примененного метода ускорения, от входных параметров сжатия и от типа изображения. Сравниваются закономерности скорости выполнения алгоритмов для изображений в оттенках серого и цветных изображений. Выявляется зависимость качества декодируемого изображения от параметров примененного алгоритма. Определяется наиболее эффективное сочетание варианта реализации алгоритма фрактального сжатия и метода ускорения для разных типов изображений.

Разработан логический проект системы в среде StarUML 5.0.

Разработана программная система, позволяющая осуществлять как фрактальное сжатие (выбрав сжимаемое изображение, вариант реализации алгоритма компрессии и метод ускорения), так и декомпрессию.

Система реализована с помощью средств языка программирования C# в среде разработки Microsoft Visual Studio Community 2015.

СОДЕРЖАНИЕ

Введение.....	7
1 Описание и анализ предметной области	9
1.1 Математическое обоснование фрактального сжатия изображений.....	9
1.2 Классический алгоритм фрактального сжатия.....	10
1.3 Преобразования доменных блоков	11
1.3.1 Аффинные преобразования.....	12
1.3.2 Преобразование яркости.....	14
1.3.3 Цветовые преобразования	15
1.4 Алгоритм декомпрессии	16
1.5 Постановка задачи	20
2 Модификации алгоритма фрактального сжатия изображений	21
2.1 Основной алгоритм фрактального сжатия.....	21
2.1.1 Первый подходящий доменный блок без разбиения.....	23
2.1.2 Первый подходящий доменный блок с разбиением.....	25
2.1.3 Доменный блок с минимальным СКО	27
2.2 Методы повышения скорости выполнения фрактального сжатия изображений	29
2.2.1 Предварительная классификация блоков	29
2.2.2 Метод эталонного блока.....	30
3 Реализация системы.....	32
3.1 Информационно-логический проект системы.....	32
3.2 Программная реализация системы	35
3.3 Выбор и обоснование комплекса программных средств	37
3.3.1 Выбор операционной системы.....	37
3.3.2 Выбор языка программирования и средства разработки	38
4 Исследования.....	39
4.1 Исследуемые параметры.....	39
4.2 Сжатие изображений типа «Портрет».....	41
4.3 Сжатие изображений с малым количеством деталей	45
4.4 Сжатие изображений с большим количеством деталей	48

4.5	Сжатие изображений с текстом	52
4.6	Сводная таблица результатов исследований сжатия изображений четырех типов.....	55
4.7	Сжатие изображений разных размеров	57
4.8	Сжатие цветных изображений	61
	Заключение	67
	Список использованных источников	68
	Приложение А Руководство пользователя	71
	Приложение Б Листинг программы	74

ВВЕДЕНИЕ

В настоящее время сложно представить себе область деятельности человека, не включающую в себя, хоть в малой степени, необходимость обмена информацией по сети Интернет. При использовании сети важно учитывать два критерия: скорость передачи информации и объем передаваемых данных. Необходимо передать как можно больше информации в сообщении наименьшего размера. В случае передачи графической информации используются различные методы сжатия изображений для уменьшения объема передаваемых данных.

В данной работе рассматривается алгоритм фрактального сжатия изображений, основанный на использовании системы итерируемых функций Iterated Function System (IFS).

Применение IFS к построению фрактальных изображений, стало результатом исследований Майкла Барнсли. Метод базируется на самоподобии элементов изображения и заключается в моделировании рисунка несколькими меньшими фрагментами его самого. Специальные уравнения позволяют переносить, поворачивать и изменять масштаб участков изображения; таким образом, эти участки служат компоновочными блоками остальной части картины[1].

Сама система итерируемых функций представляет собой набор трехмерных аффинных преобразований, переводящих одно изображение в другое. Преобразованию подвергаются точки в трехмерном пространстве (X координата, Y координата, яркость) [2]. Примером изображения, основанном на IFS-системе, является чёрный папоротник, в котором каждый лист в действительности представляет собой миниатюрный вариант самого папоротника.

Дальнейшие исследования ученых были направлены на поиск метода, позволяющего находить для любого изображения систему аффинных преобразований, воспроизводящую изображение с заданной точностью.

Первым решение данной задачи нашёл студент Барнсли, Арно Жакан (Arnaud Jacquin). Предложенный метод получил название «Система итерируемых кусочно-определённых функций» (Partitioned Iterated Function System – PIFS) [3]. Согласно его схеме, отдельные части изображения подобны не всему изображению, а только его частям.

Именно предложенное решение положило начало алгоритму фрактального сжатия, известному сегодня. Согласно ему для осуществления фрактального сжатия (или фрактальной компрессии) исходное изображение делится на подобласти, которые представляют из себя квадраты, называемые *ранговыми блоками*. Ранговые блоки пересекаться не могут. Также на исходном изображении выделяют *доменные блоки* (домены) – являющиеся совокупностью 4-х ранговых блоков. Домены могут пересекаться. Все ранговые блоки и домены – это квадраты со сторонами, параллельными сторонам исходного изображения. И затем, для каждого рангового блока ищется соответствующий ему доменный блок.

Достоинствами фрактальной компрессии являются степень сжатия на уровне JPEG при сравнительно одинаковом качестве, быстрый процесс декодирования и независимость восстанавливаемого изображения от разрешения (хранится структура изображения, а не данные о пикселях). Недостатками являются большие временные затраты сжатия и невозможность гарантировать ту или иную степень потерь (качество декодированного изображения зависит от самоподобия сжимаемого).

Целью данной выпускной квалификационной работы является изучение различных вариантов реализации алгоритма фрактального сжатия изображений, исследование подходов, позволяющих увеличить скорость фрактального сжатия и выявление зависимости качества декодируемого изображения от параметров примененного алгоритма.

1 Описание и анализ предметной области

1.1 Математическое обоснование фрактального сжатия изображений

Есть отображение $W: \Delta \rightarrow \Delta$, где Δ – множество всех возможных изображений. W является объединением отображений w_i :

$$W(R) = \bigcup_i w_i(d_i),$$

где R – изображение,

d_i – какие-то (возможно, перекрывающиеся) области изображения D .

Каждое преобразование w_i переводит d_i в r_i . Таким образом:

$$W(R) = \bigcup_i r_i.$$

Представим изображение в виде функции двух переменных $f(x, y)$. На множестве всех таких функций введём метрику (расстояние между изображениями) следующим образом:

$$\delta(f, g) = \max_{x,y} (|f(x, y) - g(x, y)|).$$

Согласно теореме Банаха, существует определённый класс отображений, для которых существует константа $0 \leq c < 1$ такая, что для любых изображений f и g выполняется неравенство:

$$\delta(W(f), W(g)) \leq c \cdot \delta(f, g).$$

Такие отображения называются *сжимающими*, и для них справедливо следующее утверждение:

Если к какому-то изображению F_0 мы начнём многократно применять отображение W таким образом, что

$$F_i = W(F_{i-1}),$$

то в пределе, при i , стремящемся к бесконечности, мы получим одно и то же изображение вне зависимости от того, какое изображение мы взяли в качестве F_0 :

$$\lim_{i \rightarrow \infty} F_i = F.$$

Это конечное изображение F называют *аттрактором*, или *неподвижной точкой отображения W* . Также известно, что если преобразования w_i являются сжимающими, то их объединение W тоже является сжимающим.

Как будет описано далее, процесс декодирования изображения будет осуществляться именно путем многократно применения отображения W к базовому изображению. Именно наличие аттрактора у отображения W позволяет при декомпрессии в качестве базового использовать любой изображение.

1.2 Классический алгоритм фрактального сжатия

По своей сути, фрактальное сжатие (или фрактальная компрессия) – это процесс поиска самоподобных областей изображения и определения для них параметров аффинных и яркостных преобразований.

Для реализации алгоритма компрессии выполняются следующие шаги [2]:

1) исходное изображение разбивается на подобласти, которые представляют из себя квадраты, называемые *ранговыми блоками*. Ранговые блоки пересекаться не могут;

2) на исходном изображении выделяются *домены* – совокупности четырех ранговых блоков. Домены могут пересекаться. Все ранговые блоки и домены – это квадраты со сторонами, параллельными изображению;

3) для каждого рангового блока производится попытка найти на изображении домен, такой чтобы этот домен можно было преобразовать в ранговый блок при помощи аффинных преобразований;

4) перевод домена в ранговый блок производится с помощью поворота домена на 0° , 90° , 180° , 270° и с помощью вертикального и горизонтального зеркальных преобразований;

5) при переводе доменной области в ранговую, ее линейный размер уменьшается в 2 раза;

6) производится изменение яркости и контрастности пикселей доменного блока (формулы преобразования приведены в разделе 1.3.2);

7) совпадение преобразованного домена с ранговым блоком может производиться при помощи среднеквадратичного отклонения:

$$\text{СКО} = \frac{1}{N^2} \sum_{i=1}^N \sum_{j=1}^N (d_{ij} - r_{ij})^2 < \varepsilon \quad (1)$$

где d_{ij} – точка в домене; r_{ij} – точка в блоке; ε – пороговое значение «похожести», N – размер стороны доменного и рангового блоков;

8) если для некоторого рангового блока не было найдено ни одного удовлетворяющего условию (1), то ранговый блок разбивается на 4 подобласти, и для каждой из них ищутся подходящие домены.

После сжатия изображения для каждого рангового блока сохраняется следующая информация:

- координаты верхнего левого угла найденного доменного блока;
- номер аффинного преобразования доменного блока;
- степень разбиения (1, если ранговый блок не разбивался на 4 подобласти; к степени разбиения прибавляется 1 при каждом следующем разбиении рангового блока);
- координаты верхнего левого угла рангового блока;
- коэффициенты изменения яркости и контрастности.

1.3 Преобразования доменных блоков

В процессе поиска подходящего доменного блока к доменам применяются следующие преобразования:

- уменьшение размера блока в 2 раза;
- аффинные преобразования;
- преобразования яркости.

Уменьшение размера блока осуществляется при помощи усреднения четырех соседних пикселей:

$$x'_{kl} = \frac{1}{4} (x_{2k \ 2l} + x_{2k+1 \ 2l} + x_{2k \ 2l+1} + x_{2k+1 \ 2l+1}),$$

где x'_{kl} - значение пикселя уменьшенного блока (k и l изменяются в диапазоне от 0 до N – размера стороны уменьшаемого блока), x - значение пикселя уменьшаемого блока.

1.3.1 Аффинные преобразования

Аффинное преобразование – это линейное преобразование плоскости или пространства (отображение плоскости или пространства в себя) [4], при котором параллельные прямые переходят в параллельные прямые, пересекающиеся – в пересекающиеся, скрещивающиеся – в скрещивающиеся.

В общем виде аффинные преобразования на плоскости описываются следующими формулами [2]:

$$\begin{cases} X = Ax + By + C, \\ Y = Dx + Ey + F, \end{cases} \quad (2)$$

где A, B, C, D, E, F – некие константы, (x, y) - координаты точки на плоскости до преобразования, (X, Y) – координаты точки на плоскости после преобразования. Преобразование (2) можно записать в матричной форме:

$$\begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} = \begin{pmatrix} A & B & C \\ D & E & F \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

При поиске самоподобных областей во фрактальном сжатии используются такие аффинные преобразования, как поворот на 0, 90, 180, 270 градусов и отражение относительно осей.

Координаты точки после поворота на φ градусов рассчитываются по формулам:

$$\begin{cases} X = x \cos \varphi - y \sin \varphi, \\ Y = x \sin \varphi + y \cos \varphi. \end{cases}$$

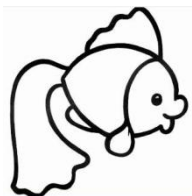
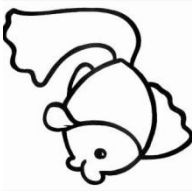
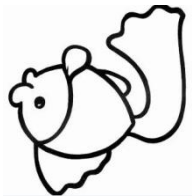

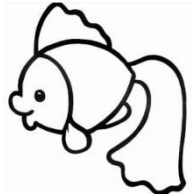
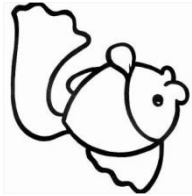
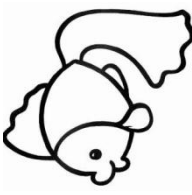
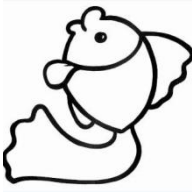
Отражения относительно осей описываются формулами:

$$\begin{cases} X = \alpha x, \\ Y = \beta y, \end{cases}$$

где при $\alpha = \pm 1, \beta = \pm 1$. При $\alpha = 1, \beta = -1$ - отражение относительно оси X , а при $\alpha = -1, \beta = 1$ - отражение относительно оси Y .

В конечном итоге каждый доменный блок подвергнется одному из восьми аффинных преобразований, приведенным в таблице 1.

Таблица 1 – Аффинные преобразования доменного блока

№	Название	Формулы	Пример
1	Поворот на 0°	$\begin{cases} x' = x \\ y' = y \end{cases}$	
2	Поворот на 90°	$\begin{cases} x' = -y \\ y' = x \end{cases}$	
3	Поворот на 180°	$\begin{cases} x' = -x \\ y' = -y \end{cases}$	
4	Поворот на 270°	$\begin{cases} x' = y \\ y' = -x \end{cases}$	
5	Отражение относительно оси Y	$\begin{cases} x' = -x \\ y' = y \end{cases}$	
6	Отражение относительно оси X	$\begin{cases} x' = x \\ y' = -y \end{cases}$	
7	Поворот на 90° и отражение относительно оси Y	$\begin{cases} x' = -y \\ y' = -x \end{cases}$	
8	Поворот на 90° и отражение относительно оси X	$\begin{cases} x' = y \\ y' = x \end{cases}$	

1.3.2 Преобразование яркости

При поиске подходящего доменного блока для достижения максимального соответствия блоков часто нам необходимо не только поворачивать доменный блок, но и изменять его яркость, контрастность или, в случае сжатия цветного изображения, оттенок.

Согласно цветовой модели RGB, цвет пикселя представлен тремя компонентами: красной, зеленой и синей. Значения компонент находятся в диапазоне от 0 до 255. В случае сжатия цветного изображения, соответствующий доменный блок ищется для каждой цветовой составляющей рангового блока отдельно. Если же сжимаемое изображение представлено в оттенках серого, т.е. все его цветовые компоненты имеют одинаковое значение, то для успешного сжатия достаточно найти подходящий доменный блок для одной цветовой составляющей рангового блока.

Для изменения цветовых составляющих блока используют *контрастность* s и *яркость* o – яркостные характеристики преобразования доменного блока к ранговому блоку.

Оптимальные контрастность и яркость минимизируют выражение [5]:

$$\sum_{i=1}^N \sum_{j=1}^N ((s d_{ij} + o) - r_{ij})^2,$$

в котором r_{ij} и d_{ij} это соответственно значения цветовых компонент пикселей ранговой и доменной областей, а N – длина стороны рангового и доменного блоков. Коэффициенты s и o могут быть вычислены с использованием метода наименьших квадратов по формулам:

$$s = \alpha / \beta,$$
$$o = \bar{r} - \frac{\alpha}{\beta} \bar{d},$$

где

$$\alpha = \sum_{i=1}^N \sum_{j=1}^N (d_{ij} - \bar{d})(r_{ij} - \bar{r}),$$

$$\beta = \sum_{i=1}^N \sum_{j=1}^N (d_{ij} - \bar{d})^2,$$

$$\bar{d} = \frac{1}{N^2} \sum_{i=1}^N \sum_{j=1}^N d_{ij},$$

$$\bar{r} = \frac{1}{N^2} \sum_{i=1}^N \sum_{j=1}^N r_{ij},$$

и N – размер стороны рангового (доменного) блока.

С учетом яркостных преобразований, условие (1) принимает следующий вид:

$$\frac{1}{N^2} \sum_{i=1}^N \sum_{j=1}^N ((sd_{ij} + o) - r_{ij})^2 < \varepsilon \quad (3)$$

1.3.3 Цветовые преобразования

В данной работе алгоритмы фрактального сжатия применяются к двум типам изображений: изображение в оттенках серого и цветное изображение. Цветное изображение представляется двумя цветовыми моделями: RGB и YIQ.

У изображений в градациях серого все три цветовые компоненты пикселя имеют одинаковое значение ($R = G = B$). Для преобразования цветного изображения в оттенки серого используется формула [2]:

$$H = 0,299 \cdot R + 0,587 \cdot G + 0,114 \cdot B$$

где R , G и B - значения красной, зеленой и синей компонент пикселя.

В цветовой модели YIQ цвет представляется тремя компонентами: Y - яркостная компонента, I и Q - цветоразностные компоненты [6]. Яркостная компонента содержит в оттенках серого изображение в оттенках серого, а оставшиеся две компоненты содержат информацию для восстановления требуемого цвета.

Для перехода от цветовой модели RGB к YIQ используют формулы:

$$Y = 0,299 \cdot R + 0,587 \cdot G + 0,114 \cdot B,$$

$$I = 0,596 \cdot R - 0,274 \cdot G - 0,322 \cdot B,$$

$$Q = 0,211 \cdot R - 0,522 \cdot G + 0,311 \cdot B.$$

Для возвращения от цветовой модели YIQ к RGB применяют преобразования:

$$R = Y + 0,956 \cdot I + 0,623 \cdot Q,$$

$$G = Y - 0,272 \cdot I - 0,648 \cdot Q,$$

$$B = Y - 1,105 \cdot I + 1,705 \cdot Q.$$

1.4 Алгоритм декомпрессии

Для осуществления декомпрессии необходимо задать базовое изображение, k – количество итераций декодирования (обычно достаточно 16 итераций), фрактальный код (т.е. набор коэффициентов аффинного преобразования, преобразования яркости, степень разбиения, координаты соответствующего домена, для каждого рангового блока).

Алгоритм фрактальной декомпрессии включает следующие шаги:

1) задаем значения исходных данных:

- базовое изображение;
- количество итераций декодирования - k ;
- файл с коэффициентами преобразований доменных блоков;

2) для каждого рангового блока на базовом изображении выделяем соответствующий параметрам доменный блок и преобразовываем его:

- уменьшаем размер в 2 раза;
- применяем аффинное преобразование;
- применяем преобразование яркости;
- копируем преобразованный доменный блок на место

текущего рангового блока в базовом изображении;

3) повторяем п.2 данного алгоритма k раз.

Схему алгоритма фрактальной декомпрессии изображения можно видеть на рисунке 1.



Рисунок 1 – Схема алгоритма декомпрессии

Пример зависимости результирующего изображения от количества итераций приведен на рисунке 2.

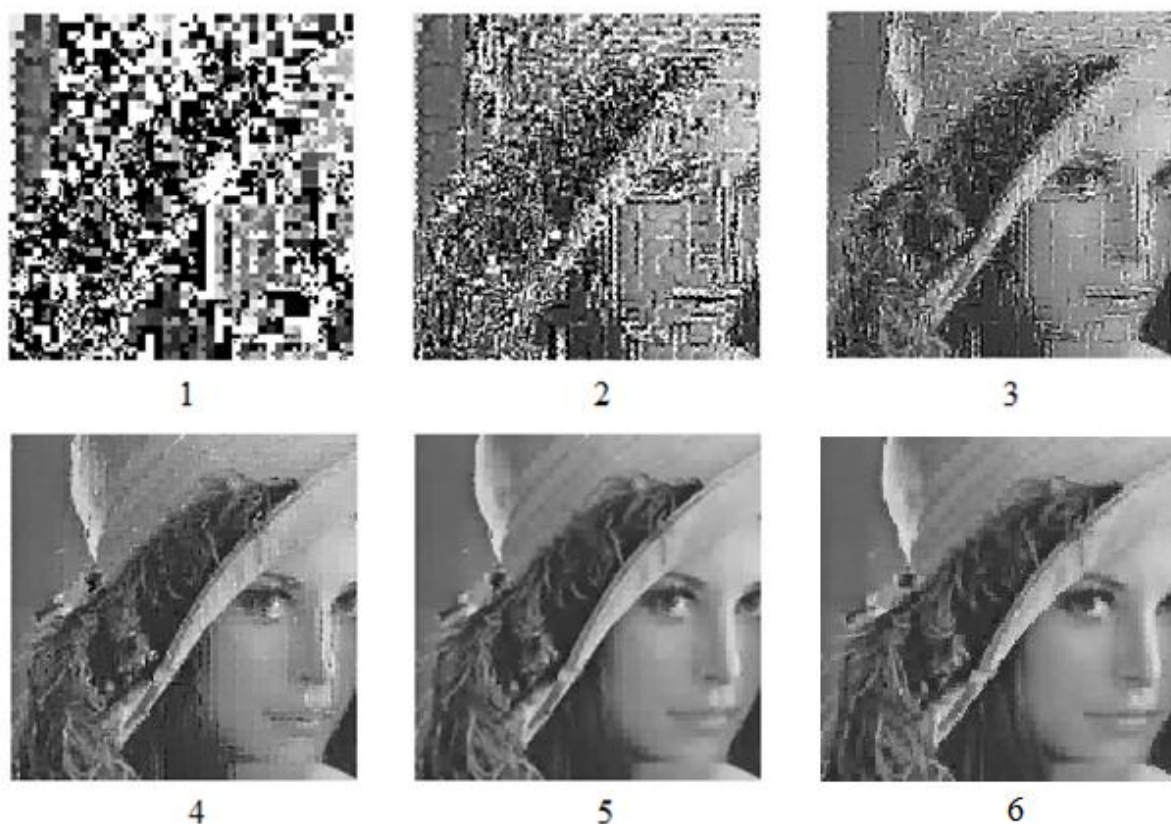


Рисунок 2 – Процесс декомпрессии изображения

Как видно из рисунка 2, с каждой итерацией декомпрессии (т.е. с каждым повторным применением аффинных и яркостных преобразований к базовому изображению) декодируемое изображение становится все более похожим на исходное. При этом, после выполнения определенного числа итераций декомпрессии, декодированное изображение перестает меняться при последующих применениях преобразований.

Пример результатов декодирования изображения, при выборе различных базовых изображения декомпрессии продемонстрированы на рисунке 3-а, 3-б, 3-в и 3-г.

Из данных рисунков можно видеть, что вне зависимости от базового изображения (белое, черное, в крупную неровную, или мелкую клетку) декодированное изображение принимает один и тот же вид.

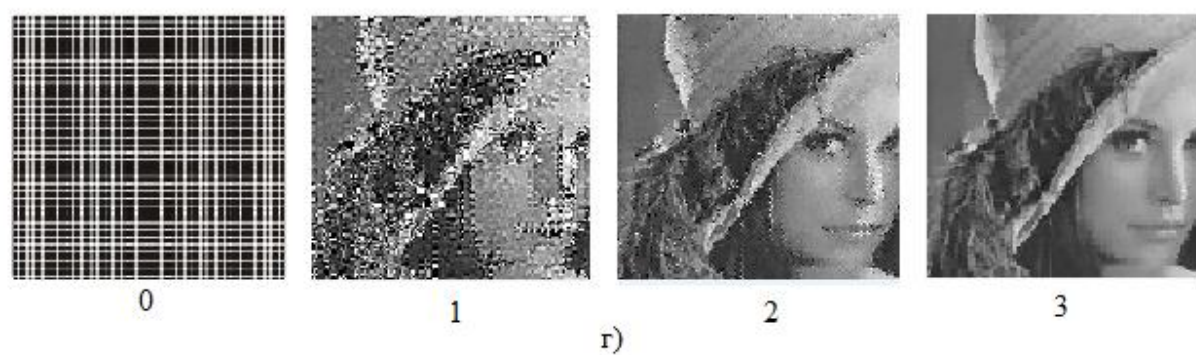
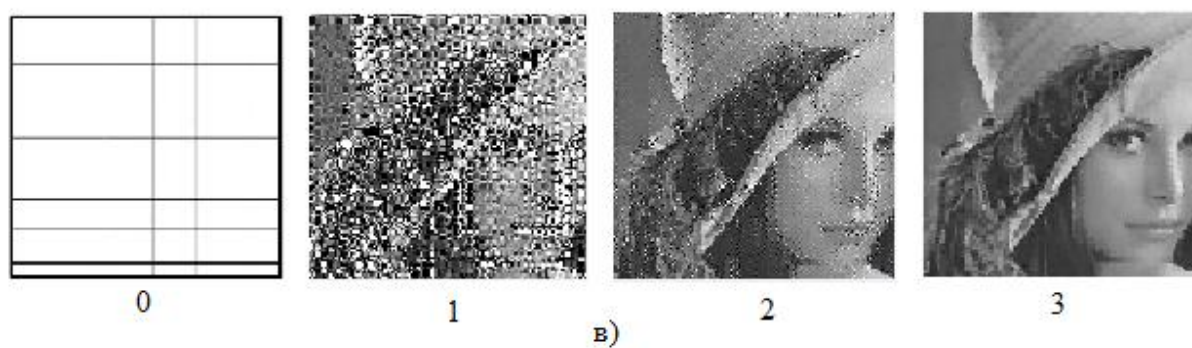
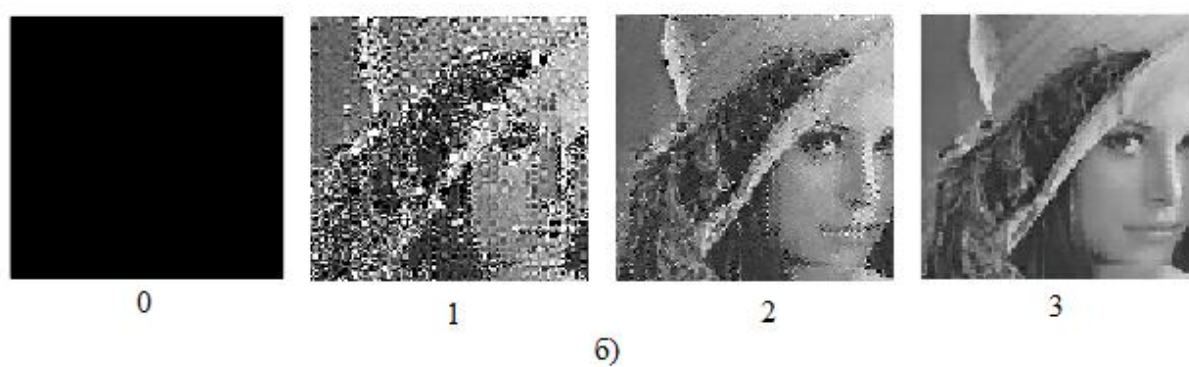
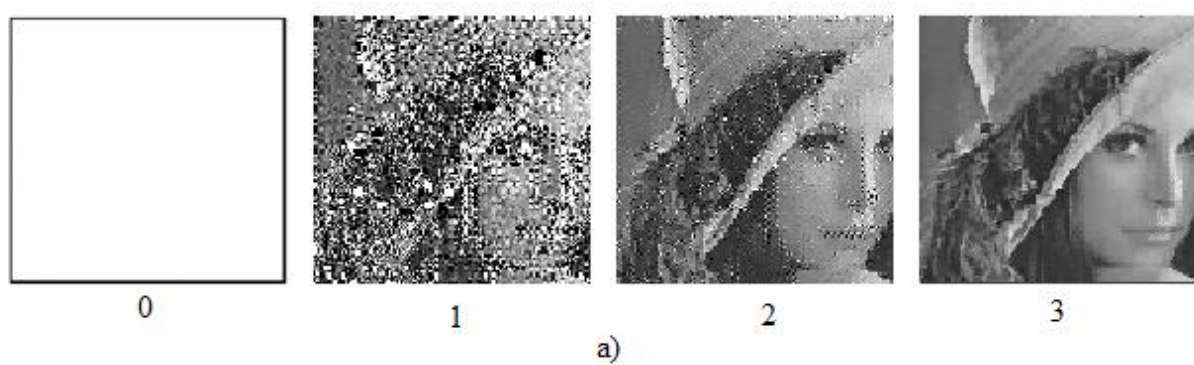


Рисунок 3 – Процесс декомпрессии при разных базовых изображениях

1.5 Постановка задачи

Целью выпускной квалификационной работы является исследование вариантов реализации алгоритма фрактального сжатия изображений, а также изучение подходов, позволяющих ускорить выполнение данного алгоритма. В соответствии с целью поставлены следующие задачи:

- изучить основной алгоритм фрактального сжатия и варианты его реализации;
- изучить методы повышения скорости выполнения фрактального сжатия изображений;
- разработать и реализовать программное и информационное обеспечение, позволяющее применять изученные алгоритмы и методы;
- исследовать зависимости скорости сжатия исходного изображения и качества декодируемого изображения от использованного алгоритма.

2 Модификации алгоритма фрактального сжатия изображений

2.1 Основной алгоритм фрактального сжатия

Общий алгоритм фрактального сжатия включает следующие шаги:

- 1) исходное изображение разбивается на ранговые блоки;
- 2) для каждого рангового блока:
 - ищется доменный блок, соответствующий ранговому блоку;
 - сохраняются параметры преобразований подошедшего доменного блока;

Схема алгоритма представлена на рисунке 4.

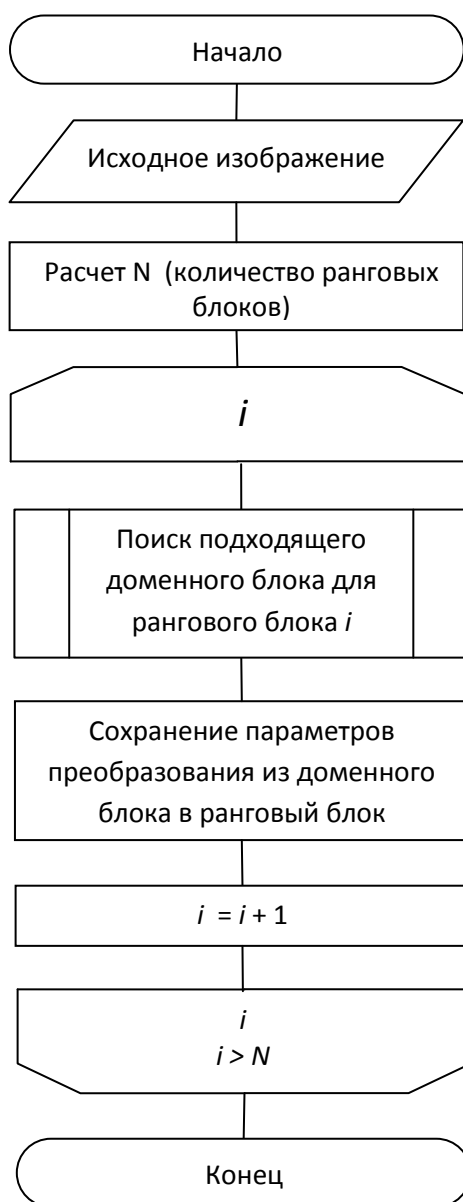


Рисунок 4 – Схема общего алгоритма фрактального сжатия

Подходящий доменный блок может выбираться несколькими способами:

1) до первого найденного доменного блока, удовлетворяющего условию (1). Если ни один доменный блок не удовлетворяет условию (3):

а) берем доменный блок с минимальным СКО (алгоритм А1);

б) разбиваем ранговый блок на 4 блока и для каждого из них ищем подходящий доменный блок (алгоритм А2).

2) доменный блок с минимальным СКО (алгоритм Б).

Каждый из перечисленных способов включает в себя расчет минимального СКО между ранговым блоком и аффинными преобразованиями доменного блока. Схема расчета представлена на рисунке 5.

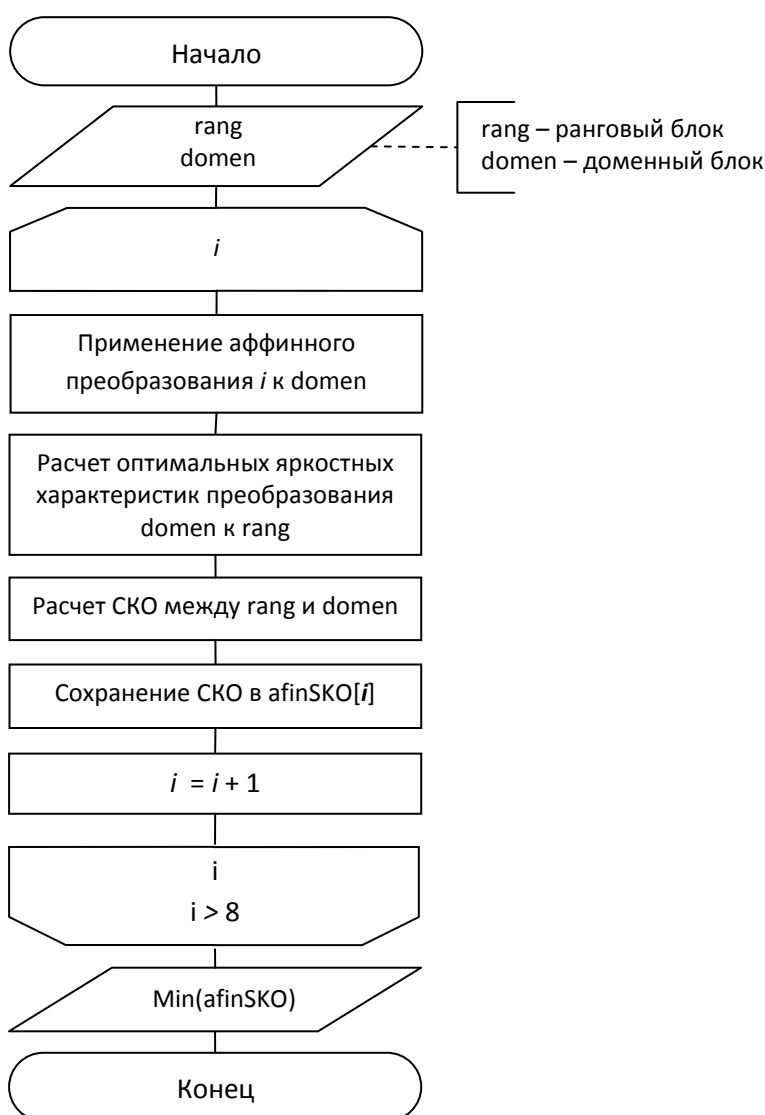


Рисунок 5 – Схема расчета минимального СКО между ранговым блоком и аффинными преобразованиями доменного блока

2.1.1 Первый подходящий доменный блок без разбиения

Входные параметры алгоритма: исходное изображение, ранговый блок, коэффициент компрессии ϵ .

Используемые параметры: $minSKO$ (значение СКО, соответствующее минимальному СКО из всех, рассчитанных для заданного рангового блока), $minX$ (координата X верхнего левого угла доменного блока, соответствующего $minSKO$), $minY$ (координата Y верхнего левого угла доменного блока, соответствующего $minSKO$), $minAfin$ (номер аффинного преобразования доменного блока, соответствующего $minSKO$).

Шаги алгоритма:

- 1) задаем значение исходных данных;
- 2) задаем начальные значения для $minSKO$, $minX$, $minY$, $minAfin$;
- 3) на исходном изображении выделяем непроверенный доменный блок;
- 4) уменьшаем его в 2 раза;
- 5) рассчитываем минимальное СКО (min) между ранговым блоком и аффинными преобразованиями доменного блока;
- 6) если минимальное СКО меньше коэффициента компрессии то сохраняем параметры преобразования текущего доменного блока, иначе – переходим к п.7;
- 7) если найденное минимальное СКО меньше значения входного параметра $minSKO$ то переходим к п.8, иначе – к п.9;
- 8) в параметр $minSKO$ присваиваем значение min , в $minX$, $minY$, $minAfin$ сохраняем соответствующие параметры доменного блока;
- 9) если на исходном изображении остались непроверенные доменные блоки, то переходим в п.2, иначе – в п.10;
- 10) сохраняем параметры преобразования доменного блока, соответствующего $minSKO$.

Схема алгоритма поиска первого подходящего доменного блока без разбиения представлена на рисунке 6.

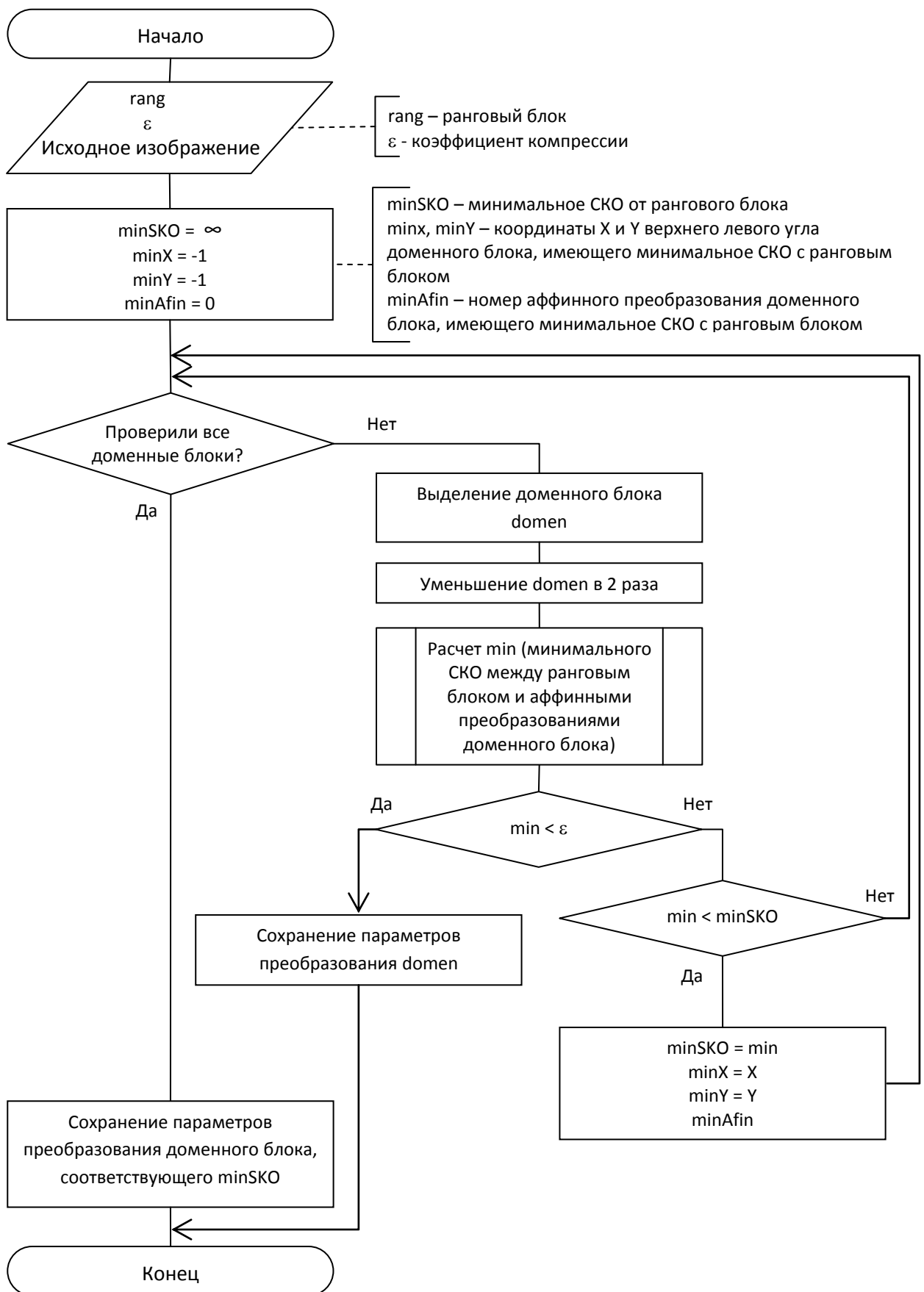


Рисунок 6 – Схема алгоритма A1

2.1.2 Первый подходящий доменный блок с разбиением

Входные параметры алгоритма: исходное изображение, ранговый блок, коэффициент компрессии ϵ .

Используемые параметры: $minSKO$ (значение СКО, соответствующее минимальному СКО из всех, рассчитанных для заданного рангового блока), $minX$ и $minY$ (координаты X и Y верхнего левого угла доменного блока, соответствующего $minSKO$), $minAfin$ (номер аффинного преобразования доменного блока, соответствующего $minSKO$).

Шаги алгоритма:

- 1) задаем значение исходных данных
- 2) задаем начальные значения для $minSKO$, $minX$, $minY$, $minAfin$;
- 3) на исходном изображении выделяем непроверенный доменный блок;
- 4) уменьшаем его в 2 раза;
- 5) рассчитываем минимальное СКО (min) между ранговым блоком и аффинными преобразованиями доменного блока;
- 6) если минимальное СКО меньше коэффициента компрессии то сохраняем параметры преобразований текущего доменного блока, иначе – переходим к п.7;
- 7) если найденное минимальное СКО меньше значения входного параметра $minSKO$ то переходим к п.8, иначе – к п.9;
- 8) в параметр $minSKO$ присваиваем значение min , в $minX$, $minY$, $minAfin$ сохраняем соответствующие параметры доменного блока;
- 9) если на исходном изображении остались непроверенные доменные блоки, то переходим в пункт 2, иначе – в п.10;
- 10) если можем разделить ранговый блок на 4 подблока, то переходим к п.11, иначе – п.12;
- 11) делим ранговый блок на 4 подблока и для каждого из них выполняем данный алгоритм;
- 12) сохраняем параметры преобразований доменного блока, соответствующего $minSKO$.

Схема алгоритма представлена на рисунке 7.

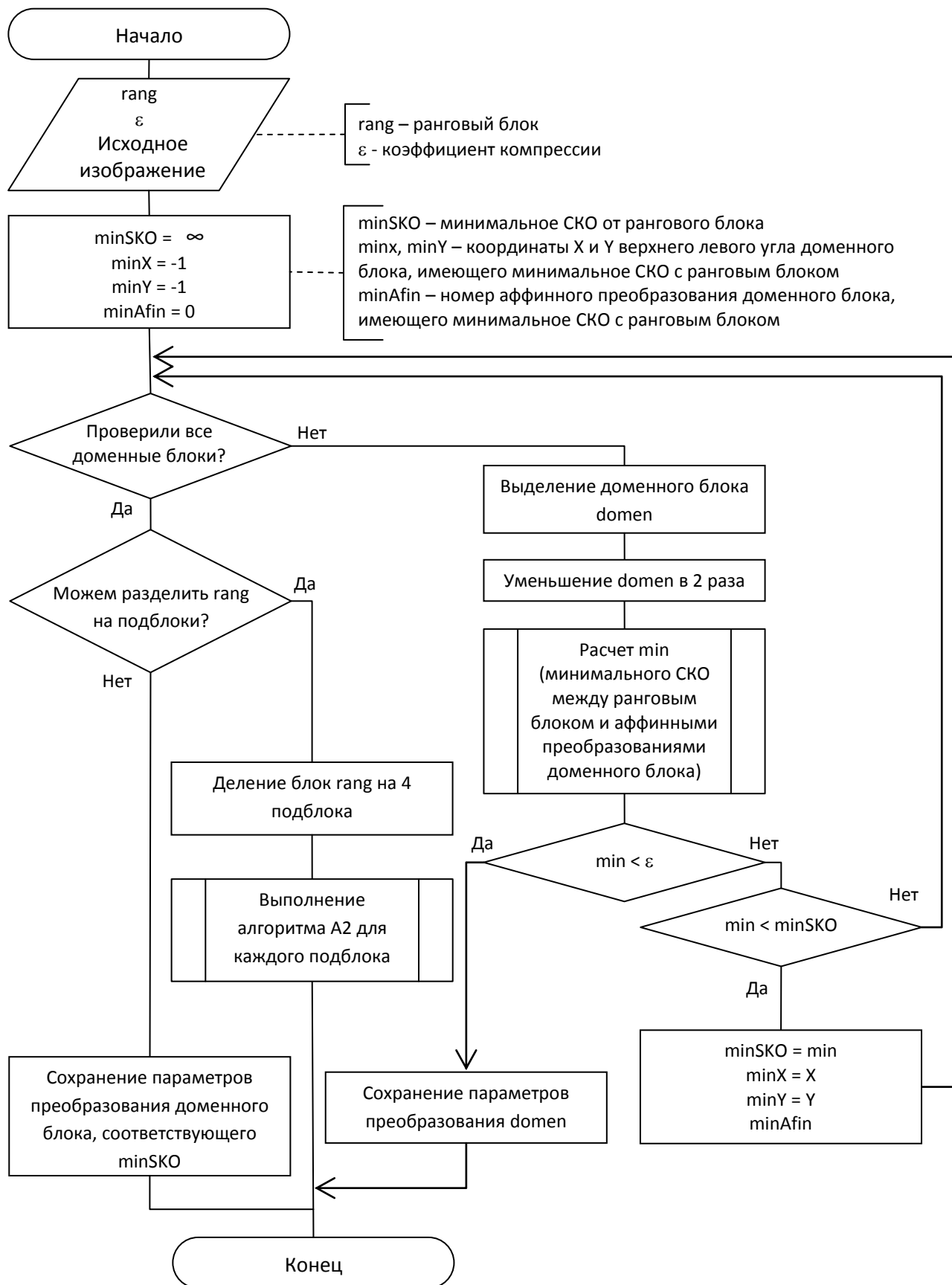


Рисунок 7 – Схема алгоритма A2

2.1.3 Доменный блок с минимальным СКО

Входные параметры алгоритма: исходное изображение, ранговый блок.

Используемые параметры: $minSKO$ (значение СКО, соответствующее минимальному СКО из всех, рассчитанных для заданного рангового блока), $minX$ (координата X верхнего левого угла доменного блока, соответствующего $minSKO$), $minY$ (координата Y верхнего левого угла доменного блока, соответствующего $minSKO$), $minAfin$ (номер аффинного преобразования доменного блока, соответствующего $minSKO$).

Шаги алгоритма:

- 1) задаем значение исходных данных;
- 2) задаем начальные значения для $minSKO$, $minX$, $minY$, $minAfin$;
- 3) на исходном изображении выделяем непроверенный доменный блок;
- 4) уменьшаем его в 2 раза;
- 5) рассчитываем минимальное СКО (min) между ранговым блоком и аффинными преобразованиями доменного блока;
- 6) если найденное минимальное СКО меньше значения входного параметра $minSKO$ то переходим к п.7, иначе – к п.8;
- 7) в параметр $minSKO$ присваиваем значение min , в $minX$, $minY$, $minAfin$ сохраняем соответствующие параметры доменного блока;
- 8) если на исходном изображении остались непроверенные доменные блоки, то переходим в пункт 2, иначе – в п.9;
- 9) сохраняем параметры преобразований доменного блока, соответствующего $minSKO$.

Схема алгоритма поиска доменного блока с минимальным СКО представлена на рисунке 8.

В случае алгоритмов А1 и Б сжимаемое изображение равномерно разбивается ранговыми блоками (рисунок 9-а), а в случае алгоритма А2 разбиение в зависимости от содержания изображения и коэффициента ε (пример разбиения приведен на рисунке 9-б).

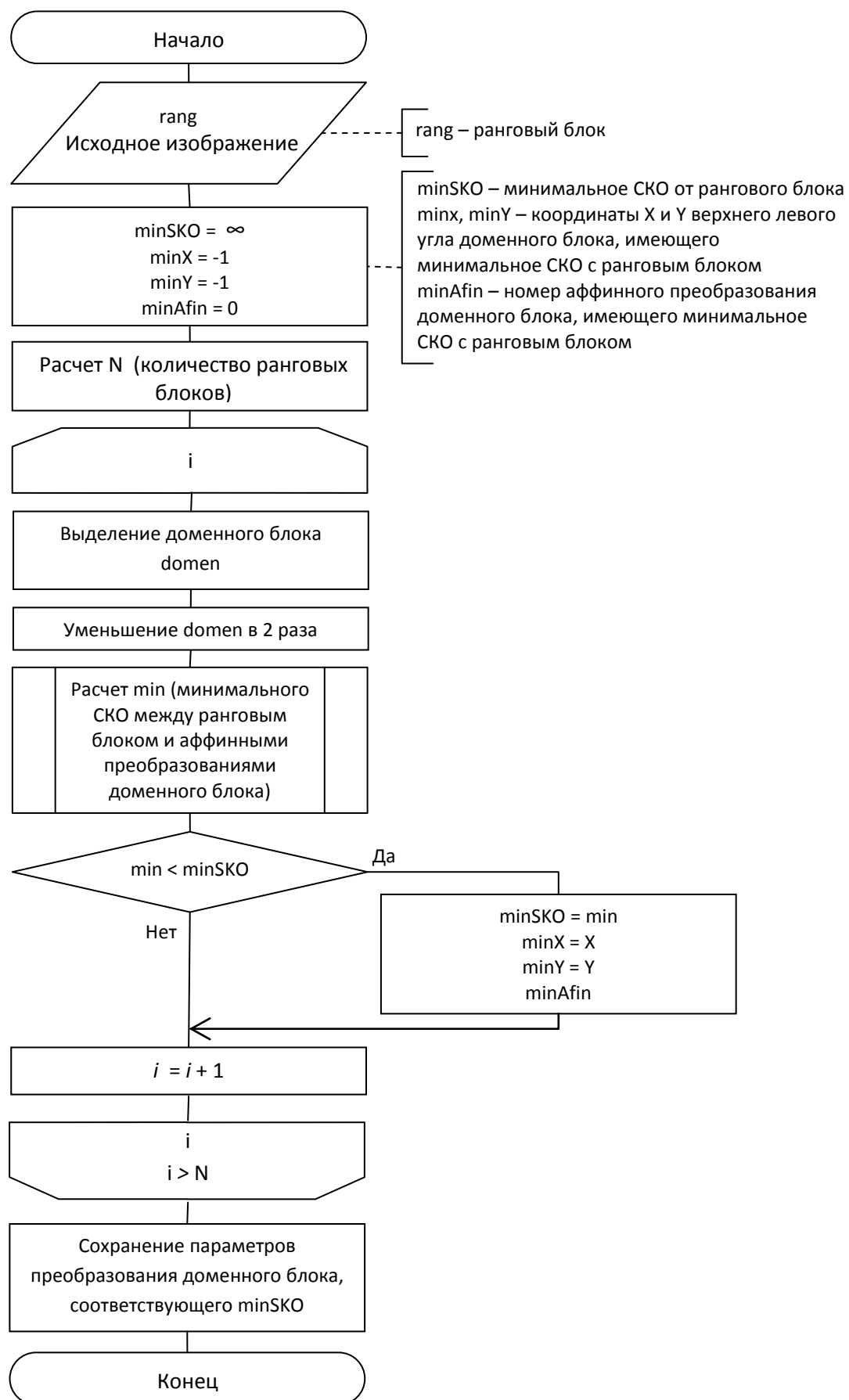


Рисунок 8 – Схема алгоритма Б

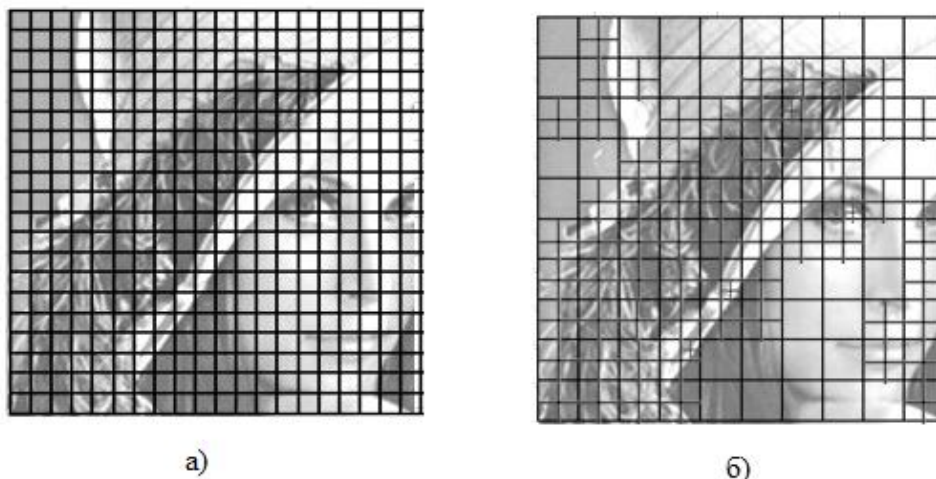


Рисунок 9 – Разбиение изображения на подблоки

2.2 Методы повышения скорости выполнения фрактального сжатия изображений

2.2.1 Предварительная классификация блоков

В случае использования предварительной классификации блоков, прежде чем приступить к выполнению компрессии, каждому ранговому и доменному блоку, выделяемому на изображении, присваивается определенный класс; в дальнейшем, поиск подходящего доменного блока осуществляется только среди доменных блоков, имеющих тот же класс, что и ранговый блок [7].

В данной работе для классификации блоков используются следующие подходы:

- 1) классификация по значению центра массы блока;
- 2) классификация по разнице граничных яркостных значений блока.

Центр масс блока рассчитывается по формуле:

$$\text{ЦМ} = \frac{1}{N^2} \sum_{i=1}^N \sum_{j=1}^N x_{ij}, \quad (4)$$

в которой N – количество пикселей в блоке, x_{ij} – значение цветовой компоненты пикселя.

Разница граничных яркостных значений блока вычисляется по формуле:

$$\text{РГЗ} = \max(x_{ij}) - \min(x_{ij}). \quad (5)$$

Так как значения компонент находятся в диапазоне от 0 до 255, и формула 4 и формула 5 будут возвращать значения в том же диапазоне.

В общем случае данный диапазон можно разбить на 5 интервалов, каждый из которых будет соответствовать одному классу. После вычисления ЦМ или РГЗ для рангового или доменного блока, в зависимости от полученного значения, блоку присваивается соответствующий класс.

2.2.2 Метод эталонного блока

Входные параметры алгоритма: исходное изображение, эталонный блок, коэффициент компрессии ε . Пример эталонного блока приведен на рисунке 10.

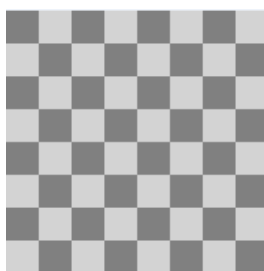


Рисунок 10 – Пример эталонного блока

Используемые параметры: $domensSKOs[j][s]$ (матрица значений среднеквадратических отклонений аффинных преобразований доменного блока от эталонного блока, где j - номер доменного блока, s - номер аффинного преобразования).

Шаги алгоритма:

- 1) задаем значение исходных данных;
- 2) рассчитываем СКО между эталонным блоком и каждым аффинным преобразованием всех доменных блоков; сохраняем рассчитанные значения в матрицу $domensSKOs$;
- 3) выделяем ранговый блок;
- 4) рассчитываем $rangSKO$ (значение СКО между эталонным и ранговым блоками);
- 5) рассчитываем min (минимальный модуль разницы между $rangSKO$ и значениями матрицы $domensSKOs$);
- 6) если min меньше ε , то переходим к п.7, иначе – к п.8;

7) сохраняем параметры преобразования доменного блока, соответствующего min ; переходим к п.9;

8) для текущего доменного блока выполняем алгоритм A1, A2 или Б;

9) если закодированы не все ранговые блок, то переходим к п.3, иначе – заканчиваем выполнение алгоритма.

Схема алгоритма представлена на рисунке 11.

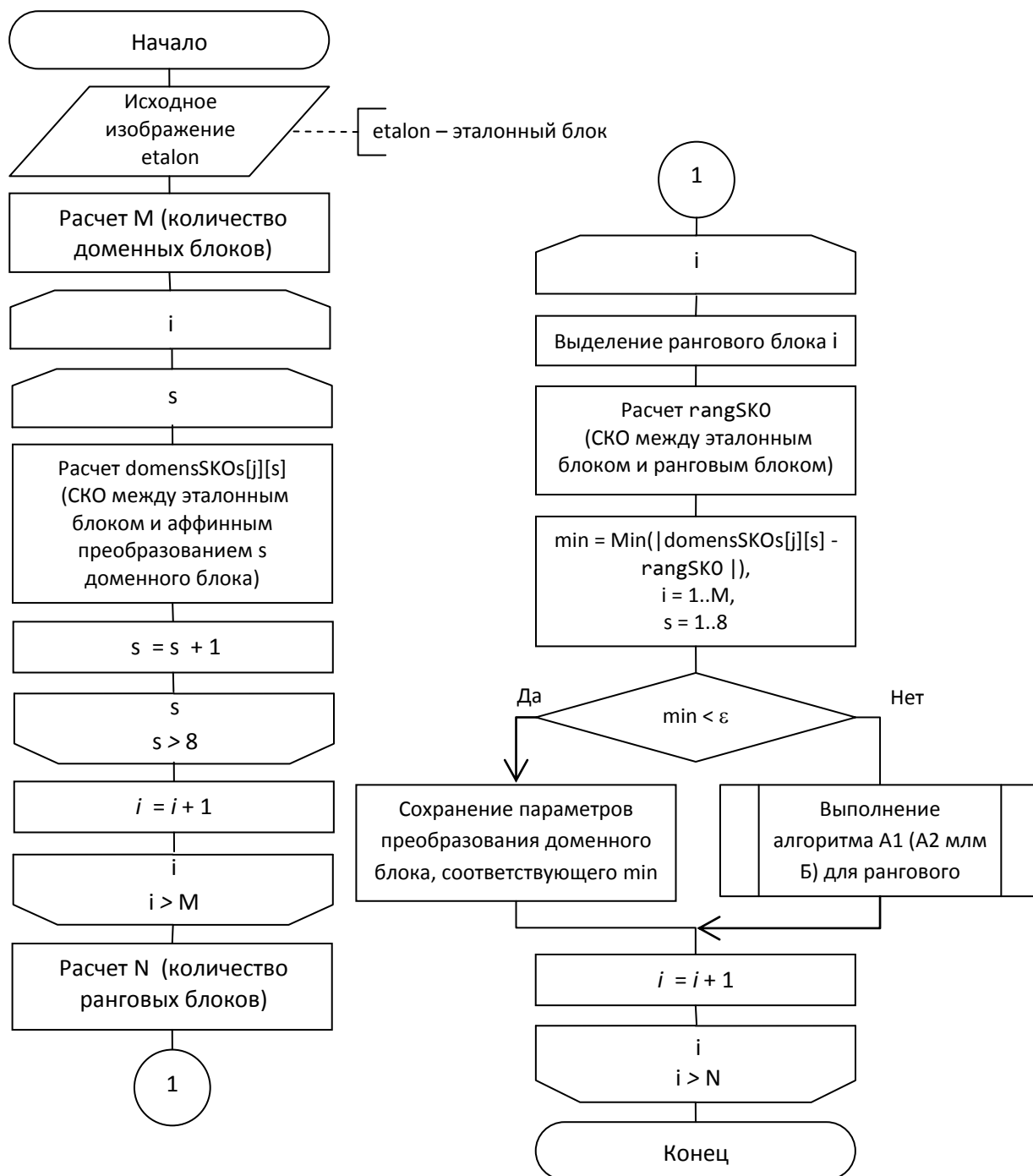


Рисунок 11 – Схема метода эталонного блока

3 Реализация системы

3.1 Информационно-логический проект системы

В данной работе для описания информационно логического проекта системы используется язык UML.

Унифицированный язык моделирования (Unified Modeling Language – UML) – это стандартный инструмент для разработки «чертежей» программного обеспечения. Его можно использовать для визуализации, спецификации, конструирования и документирования артефактов программных систем. UML подходит для моделирования любых систем – от информационных систем масштаба предприятия до распределенных Web-приложений и даже встроенных систем реального времени [8]. Преимуществом методологии UML является объектно-ориентированное представление моделей системы.

Для разрабатываемой системы далее представлены следующие диаграммы, предоставляемые нотацией языка UML: *диаграмма вариантов использования* представляющая собой наиболее общую концептуальную модель сложной системы, которая является исходной для построения всех остальных диаграмм и *диаграмма состояний*, предназначенная для моделирования поведения системы.

Визуальное моделирование начинается с модели в форме так называемой диаграммы вариантов использования (*use case diagram*), которая описывает функциональное назначение системы и является исходным концептуальным представлением в процессе ее проектирования и разработки. На ней изображаются отношения между актерами и вариантами использования.

Актер (actor) – согласованное множество ролей, которые играют внешние сущности по отношению к вариантам использования при взаимодействии с ними (это может быть любой объект, субъект или система, взаимодействующая с моделируемой бизнес-системой извне, т.е. человек, техническое устройство, программа и т.п.).

Вариант использования – внешняя спецификация последовательности действий, которые система или другая сущность могут выполнять в процессе

взаимодействия с актерами (он определяет набор действий, совершаемый системой при диалоге с актером).

На рисунке 12 приведена диаграмма вариантов использования для разрабатываемой системы.

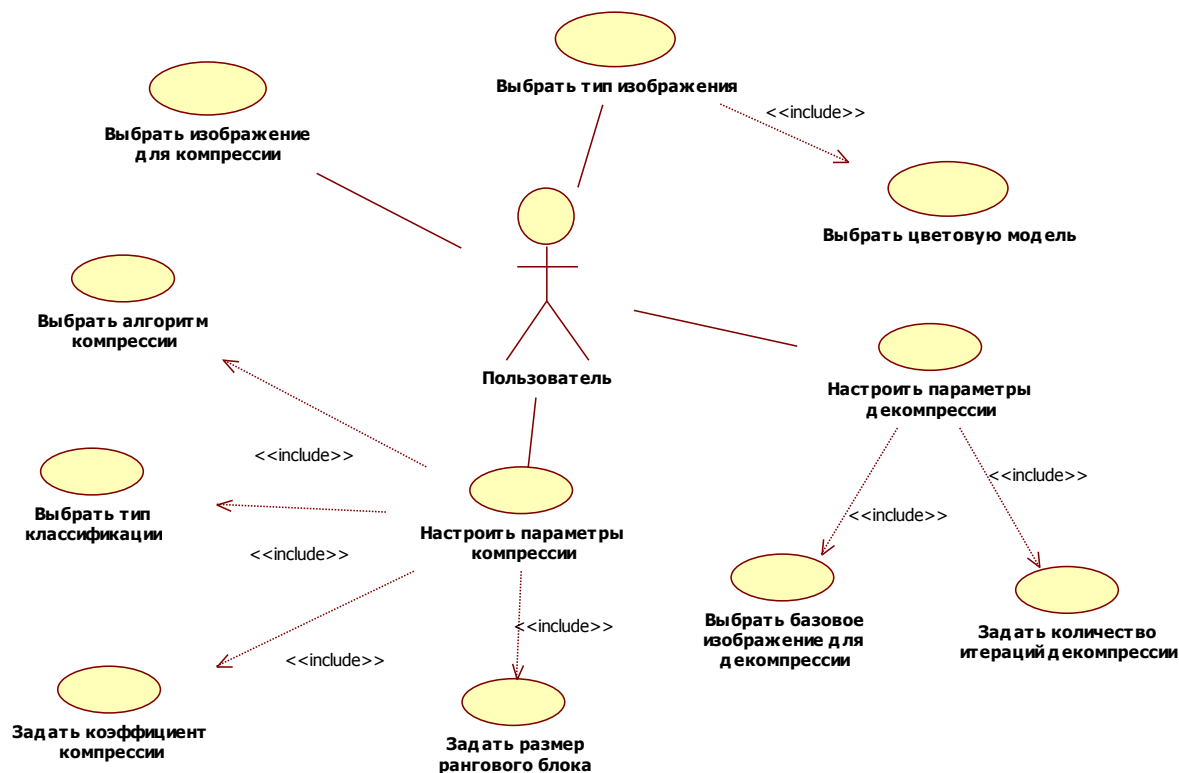


Рисунок 12 – Диаграмма вариантов использования системы

Исходя из приведенной диаграммы, система должна позволять пользователю выбрать сжимаемое изображение, выбрать его тип (цветное или в оттенках серого) и цветовую модель (RGB или YIQ). Пользователь системы должен иметь возможность настраивать параметры компрессии (выбрав алгоритм сжатия, тип классификации, коэффициент компрессии и размер рангового блока) и декомпрессии (выбрав базовое изображение для декомпрессии и количество итераций).

Диаграмма состояний (statechart diagram) - это диаграмма, главное предназначение которой - описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла [9].

По существу диаграмма состояний является ориентированным графом специального вида, который представляет некоторый конечный автомат. Вершинами этого графа являются состояния и некоторые другие типы элементов автомата (псевдосостояния), которые изображаются соответствующими графическими символами. Дуги графа служат для обозначения переходов из состояния в состояние.

Поведение (*behavior*) является спецификацией того, как экземпляр классификатора изменяет значения отдельных характеристик в течение своего времени жизни [10].

Состояние (state) – элемент модели поведения, предназначенный для представления ситуации, в ходе которой поддерживается некоторое условие инварианта.

Переход (transition) является направленным отношением между двумя состояниями, одно из которых является вершиной *источником* (source vertex), а другое – *целевой вершиной* (target vertex).

Диаграммы состояний могут быть вложены друг в друга, образуя вложенные диаграммы более детального представления отдельных элементов модели.

На рисунке 13 приведена диаграмма состояний для разрабатываемой системы. После начального состояния система переходит в состояние «Запуск программы», в котором открывается главная форма приложения. Далее система переходит в состояние «Ожидание действий пользователя».

В зависимости от них система оперирует изображением (состояние «Открытие диалогового окна выбора изображения»), изменяет параметры компрессии (состояние «Изменение параметров компрессии»), или декомпрессии (состояние «Изменение параметров декомпрессии»), производит компрессию и декомпрессию, а также выводит на экран информацию о затраченном времени и качестве декодируемого изображения (состояния «Вывод времени компрессии на экран», «Вывод времени декомпрессии и SSIM на экран»).

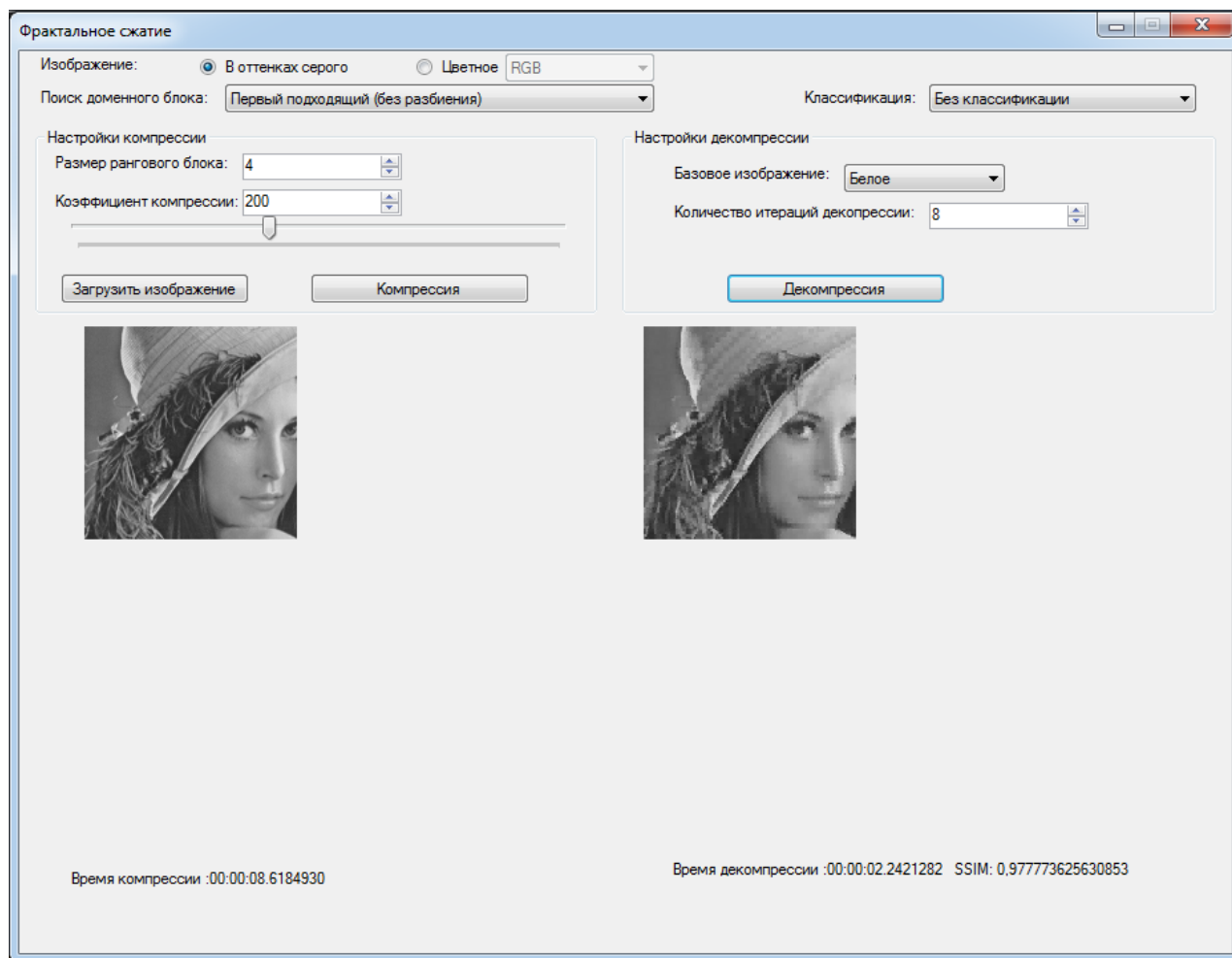


Рисунок 14 – Интерфейс разработанной системы

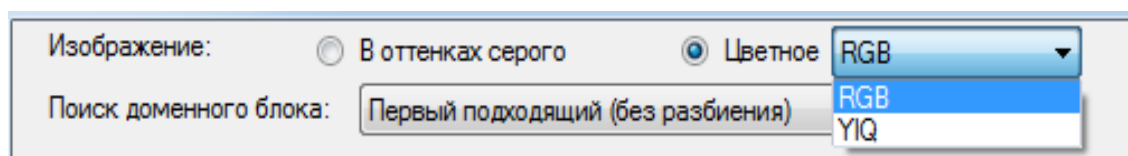


Рисунок 15 – Панель выбора типа изображения и цветовой модели

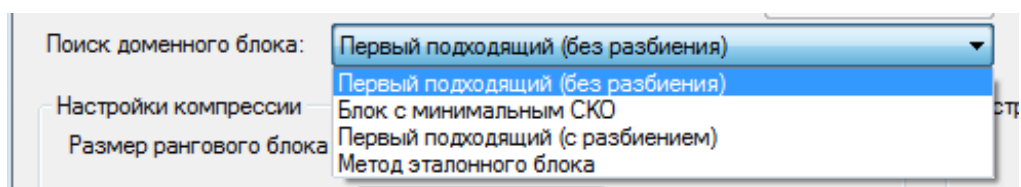


Рисунок 16 – Панель выбора алгоритма поиска доменного блока

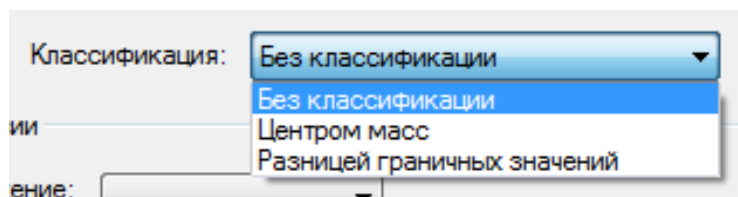


Рисунок 17 – Панель выбора используемой классификации

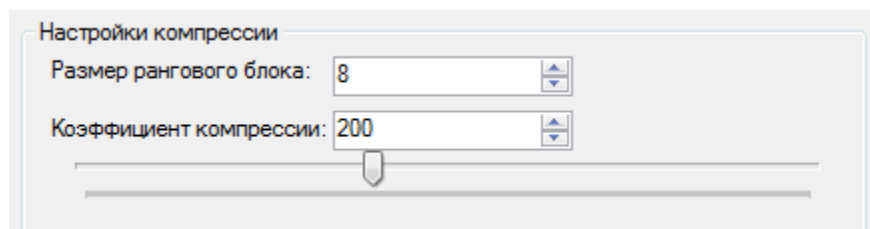


Рисунок 18 – Панель ввода коэффициента компрессии и размера рангового блока

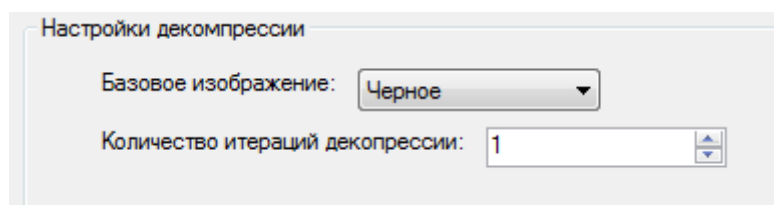


Рисунок 19 – Панель выбора параметров декомпрессии

Под областью настроек параметров на экран выводятся сжимаемое и декодированное изображение, а в нижней части программного окна (см. рисунок 14) – время компрессии, время декомпрессии и SSIM (метрика качества декодированного изображения, подробное описание которой приведено в разделе 4.1).

3.3 Выбор и обоснование комплекса программных средств

3.3.1 Выбор операционной системы

Операционная система – комплекс взаимосвязанных программ, предназначенных для управления ресурсами компьютера и организации взаимодействия с пользователем [11]. Выбор ОС зависел от требований, которым должна в оптимальной степени удовлетворять ОС:

- совместимость с выбранной архитектурой ЛВС;
- поддержка выбранного аппаратного обеспечения;
- эффективное управление ресурсами;
- достаточная для решения основных задач производительность;
- масштабируемость;
- совместимость;
- довольно высокая надёжность и отказоустойчивость;
- защита данных от несанкционированного доступа;

- поддержка многозадачности;
- приоритет удобства работы пользователя над скоростью вычислений.

Данным требованиям в той или иной мере соответствуют весьма популярные представители семейства Windows – Windows 7 и Windows 8.

3.3.2 Выбор языка программирования и средства разработки

C# является языком программирования, который разработан для создания множества приложений, работающих в среде .NET Framework. Язык C# прост и объектно-ориентирован. Благодаря множеству нововведений C# обеспечивает возможность быстрой разработки приложений, но при этом сохраняет выразительность и элегантность, присущую C-подобным языкам [12].

Visual C# - это реализация языка C# корпорацией Майкрософт. Поддержка Visual C# в Visual Studio обеспечивается с помощью полнофункционального редактора кода, компилятора, шаблонов проектов, конструкторов, мастеров кода, мощного и удобного отладчика и многих других средств. Библиотека классов .NET Framework предоставляет доступ ко многим службам операционной системы и к другим полезным, хорошо спроектированным классам, что существенно ускоряет цикл разработки.

Microsoft Visual Studio Community 2015 - это бесплатная полнофункциональная интегрированная среда разработки с мощными, эффективными возможностями для кодирования, инструментами кроссплатформенных разработок мобильных приложений для Windows, iOS и Android, веб- и облачных приложений, а также доступом к тысячам расширений. Этот выпуск Visual Studio доступен для отдельных разработчиков, для разработки проектов с открытым исходным кодом, академических исследований, образования и небольших групп специалистов [13].

4 Исследования

4.1 Исследуемые параметры

Исследуемые параметры можно разделить на 2 типа: параметры компрессии и параметры декомпрессии.

Параметры фрактального сжатия включают в себя *время сжатия*, вычисляемое программной системой, а также *степень сжатия*, которая вычисляется по формуле:

$$СЖ = \frac{\text{исходный размер изображения}}{\text{размер сжатого изображения}}.$$

В качестве размера сжатого изображения берется размер файла с расширением txt, в который сохраняется фрактальный код. Сохранение коэффициентов преобразований для каждого рангового блока происходит следующим образом:

- 1) в единое 64-х битное число сохраняются координаты доменного блока, координаты рангового блока, номер аффинного преобразования и степень разбиения;
- 2) вычисленное в п.1 число, а также значения яркости и контрастности (представленные числами с двумя знаками после запятой) через пробел записываются в файл.

Сохранение нескольких параметров в единое число осуществляется за счет побитового сдвига и выполняется по формуле:

$$\begin{aligned} \text{Результирующее число} = & (Y0 \ll 10) + (X0 \ll 21) + \\ & +(k \ll 25) + (afinn \ll 29) + (Y \ll 40) + (X \ll 51), \end{aligned}$$

где слева от символа " \ll " располагается число, для которого осуществляется сдвиг, а справа – количество битов сдвига,

$X0, Y0$ - координаты верхнего левого угла декодируемого рангового блока,

X, Y - координаты верхнего левого угла доменного блока,

k - степень разбиения,

$afinn$ - порядковый номер аффинного преобразования доменного блока.

В качестве параметров декомпрессии берутся *время декомпрессии* (измеряемое программной системой) и *качество декодируемого изображения*.

Для оценки качества декодируемого изображения используется метрика SSIM [14]. Данная метрика учитывает искажение яркости и контрастности, а также – степень коррелированности между исходным и декодируемым изображением.

Значение SSIM рассчитывается по следующей формуле:

$$SSIM(x, y) = L(x, y)^\alpha C(x, y)^\beta S(x, y)^\gamma, \quad (6)$$

где $L(x, y)$ - яркость, $C(x, y)$ - контрастность, $S(x, y)$ - структура, x – матрица значений пикселей оригинального (эталонного) изображения, y - матрица значений пикселей оцениваемого изображения, α, β, γ – коэффициенты для регулировки важности составляющих компонент.

Сами компоненты вычисляются по следующим формулам:

$$L(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1},$$

$$C(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2},$$

$$S(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3},$$

где C_1, C_2, C_3 - выравнивающие коэффициенты, предотвращающие деление на число, близкое к 0, при высоком качестве оцениваемого изображения.

Коэффициенты рассчитываются по формуле:

$$C_i = (K_i L)^\alpha,$$

где K_i – константа ($K_i \ll 1$), L - максимальное значение канала. Обычно берут $K_1 = 0,01, K_2 = 0,03$. Для изображений, в которых для хранения значения пикселя используется 8 бит $L = 255$.

Для упрощения выражения (6) берут $\alpha = \beta = \gamma = 0$, а $C_3 = \frac{C_2}{2}$, что приводит к следующему преобразованию:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \quad (7)$$

где μ_f – среднее арифметическое значений каналов пикселей изображения f ,
 σ_f - стандартное отклонение значений каналов пикселей изображения f ,
 σ_{xy} - ковариация значений каналов пикселей изображений x и y .

Среднее арифметическое каналов пикселей изображения рассчитывается по формулам:

$$\mu_x = \frac{1}{3N} \sum_{i=1}^N (x_{i_R} + x_{i_G} + x_{i_B}),$$

$$\mu_y = \frac{1}{3N} \sum_{i=1}^N (y_{i_R} + y_{i_G} + y_{i_B}),$$

где $x_{i_R}, x_{i_G}, x_{i_B}, y_{i_R}, y_{i_G}, y_{i_B}$ – значения соответствующих цветовых каналов для пикселей изображений x и y , N - размер матриц изображений x и y ;

$$\sigma_x^2 = \frac{1}{3(N-1)} \sum_{i=1}^N ((x_{i_R} - \mu_x)^2 + (x_{i_G} - \mu_x)^2 + (x_{i_B} - \mu_x)^2),$$

$$\sigma_y^2 = \frac{1}{3(N-1)} \sum_{i=1}^N ((y_{i_R} - \mu_y)^2 + (y_{i_G} - \mu_y)^2 + (y_{i_B} - \mu_y)^2),$$

$$\sigma_{xy} = \frac{1}{3(N-1)} \sum_{i=1}^N ((x_{i_R} - \mu_x)(y_{i_R} - \mu_y) + (x_{i_G} - \mu_x)(y_{i_G} - \mu_y) + (x_{i_B} - \mu_x)(y_{i_B} - \mu_y)).$$

Значения SSIM лежат в диапазоне от -1 до 1, и чем ближе к 1, тем более декодируемое изображение похоже на исходное. Если SSIM равно 1, то два сравниваемых изображения полностью идентичны.

4.2 Сжатие изображений типа «Портрет»

В данном разделе исследования проводятся над набором изображений в оттенках серого, размером 160×160 пикселей [15].

В таблице 2 представлена зависимость параметров декомпрессии от размера рангового блока и коэффициента компрессии для изображений типа «Портрет» при использовании алгоритма A1.

Таблица 2 – Зависимость результатов декомпрессии от размера рангового блока и ε при использовании алгоритма A1 для изображений типа «Портрет»

Размер рангового блока	ε	$t_{\text{комп}}$, сек	$t_{\text{декомп}}$, сек	Степень сжатия	SSIM
4	10	120,67	1,71	4,39	0,9901
	50	51,99	1,65	4,39	0,9872
	100	25,88	1,72	4,37	0,9833
	150	15,54	1,72	4,37	0,9799
	200	7,83	1,67	4,37	0,9778
	300	2,49	1,68	4,37	0,9730
8	2	46,66	1,71	17,51	0,9566
	5	39,76	2,15	17,55	0,9564
	10	38,33	2,04	17,55	0,9560
	50	28,13	1,65	17,55	0,9543
	100	14,96	1,76	17,55	0,9517
	150	13,32	1,72	17,55	0,9491

Из таблицы 2 можно видеть, что с увеличением коэффициента компрессии уменьшается не только время сжатия, но и качество декодируемого изображения. Для дальнейшего исследования для алгоритма A1 размер рангового блока берется равный 4 и коэффициент ε , равный 150.

Таблица 3 – Зависимость результатов декомпрессии от использованного алгоритма сжатия и параметров компрессии для изображений типа «Портрет»

Алгоритм выбора доменного блока	Метод классификации	Размер рангового блока	ε	$t_{\text{комп}}$, сек	$t_{\text{декомп}}$, сек	Степень сжатия	SSIM
Первый подходящий (без разбиения)	-	4	150	9,21	1,89	4,53	0,9808
	Центр масс	4	150	4,06	2,01	4,49	0,9765
	Разница граничных значений	4	150	2,95	2,07	4,53	0,9773
Первый подходящий (с разбиением)	-	16	5	45,08	1,83	8,79	0,9793
	Центр масс	16	5	16,84	1,91	8,43	0,9797
	Разница граничных значений	16	5	15,86	2,45	8,45	0,9805
Доменный блок с минимальным СКО	-	8	-	35,72	2,15	17,67	0,9722
	Центр масс	8	-	15,75	2,13	17,71	0,9677
	Разница граничных значений	8	-	13,43	2,21	17,72	0,9711

Как видно из рисунка 20 и таблицы 3, наиболее эффективным с точки зрения затрачиваемого времени и качества декодируемого изображения, будет использование алгоритма A1 с классификацией РГЗ.

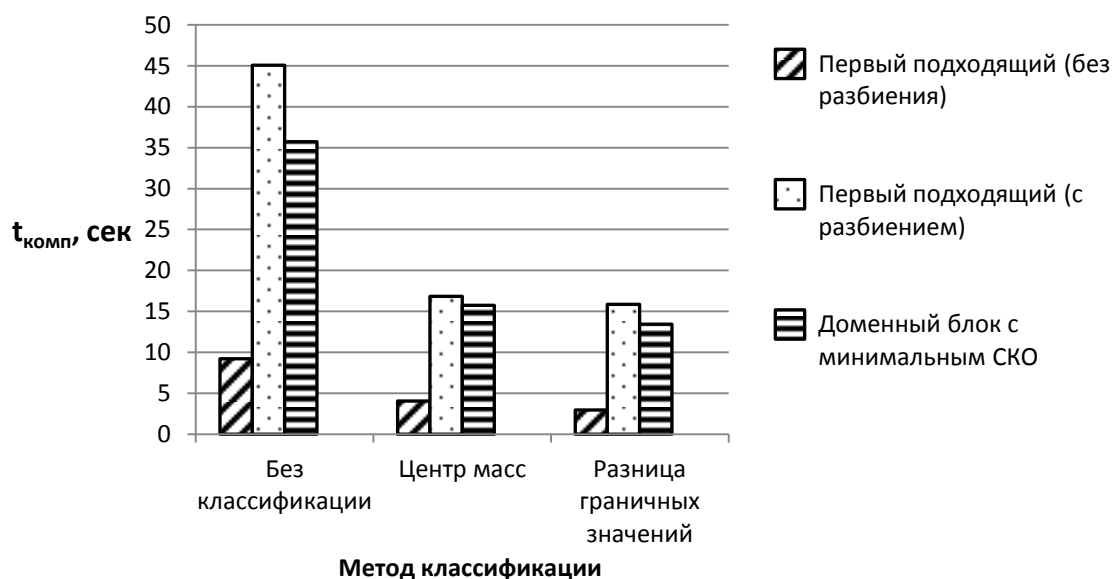


Рисунок 20 – Зависимость времени сжатия изображения от алгоритма выбора доменного блока и типа классификации (для изображений типа «Портрет»)

В таблице 4 приводится исследование зависимости параметров декомпрессии при использовании метода эталонного блока от размера рангового блока и коэффициента ε .

Таблица 4 – Зависимость результатов декомпрессии от размера рангового блока и ε при сжатии методом эталонного блока для изображений типа «Портрет»

Размер рангового блока	ε	$t_{\text{комп}}, \text{сек}$	$t_{\text{декомп}}, \text{сек}$	Степень сжатия	SSIM
4	10	144,35	1,65	4,39	0,9898
	50	84,28	1,71	4,39	0,9860
	100	57,08	1,73	4,42	0,9818
	200	39,64	1,62	4,42	0,9754
	300	34,65	1,67	4,42	0,9719
	400	33,46	1,73	4,42	0,9663
8	5	53,62	1,72	17,51	0,9564
	10	47,25	1,83	17,51	0,9560
	50	38,36	1,68	17,51	0,9528
	100	25,27	1,71	17,51	0,9503
	150	19,73	1,65	17,55	0,9473
	200	18,08	1,68	17,51	0,9454

Для сравнения метода эталонного блока с алгоритмами A1 и A2(при условии приблизительно равного качества декодируемого изображения) размер рангового блока берется равный 4 и коэффициент ε , равный 300.

Таблица 5 – Зависимость времени сжатия от выбранного алгоритма для изображений типа «Портрет»

Алгоритм выбора доменного блока	Метод классификации	Размер рангового блока	ε	$t_{\text{комп, сек}}$	$t_{\text{декомп, сек}}$	Степень сжатия	SSIM
Первый подходящий (без разбиения)	Разница граничных значений	4	150	2,95	2,07	4,53	0,9773
Первый подходящий (с разбиением)	-	16	5	45,08	1,83	8,79	0,9793
Первый подходящий (с разбиением)	Разница граничных значений	16	5	15,86	2,45	8,45	0,9805
Метод эталонного блока	-	4	300	34,99	2,36	4,69	0,9782

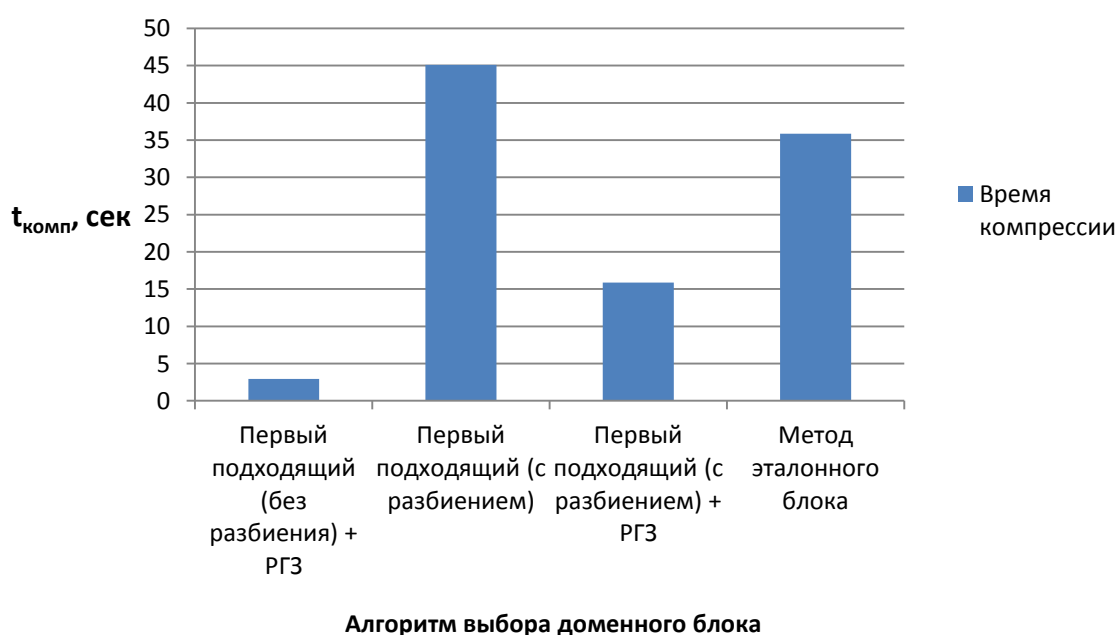


Рисунок 21 – Зависимость времени сжатия изображения от выбранного алгоритма (для изображений типа «Портрет»)

Как видно из рисунка 21 и таблицы 5 использование классификации РГЗ обеспечивают заметный выигрыш по времени, а метода эталонного блока лишь немного уменьшает время по сравнению с алгоритмом А2. Наиболее эффективным по времени выполнения остается использование поиска первого подходящего доменного блока (алгоритм А1) с классификацией разницей граничных значений.

4.3 Сжатие изображений с малым количеством деталей

В данном разделе исследования проводятся над набором изображений в оттенках серого, размером 160×160 пикселей [15].

В таблице 6 представлена зависимость параметров декомпрессии (при использовании алгоритма А1) от размера рангового блока и коэффициента компрессии.

Таблица 6 – Зависимость результатов декомпрессии от размера рангового блока и ϵ при использовании алгоритма А1 для изображений с малым количеством деталей

Размер рангового блока	ϵ	$t_{\text{комп}}, \text{сек}$	$t_{\text{декомп}}, \text{сек}$	Степень сжатия	SSIM
4	10	155,83	1,92	4,39	0,9854
	50	103,41	1,85	4,39	0,9854
	100	60,38	1,76	4,39	0,9851
	200	22,04	1,68	4,39	0,9841
	300	9,64	1,71	4,39	0,9834
	400	3,26	1,68	4,39	0,9824
8	5	22,21	1,69	17,51	0,9786
	10	18,63	1,78	17,55	0,9783
	50	14,48	1,65	17,59	0,9771
	100	12,36	1,99	17,55	0,9750
	150	12,15	1,83	17,51	0,9746
	200	10,84	1,72	17,51	0,9745

Из таблицы 6 можно видеть, что с увеличением коэффициента компрессии уменьшается не только время сжатия, но и качество декодируемого изображения. Уменьшение происходит с меньшим шагом, чем в случае сжатия изображения типа «Портрет» (см. таблицу 2).

Для дальнейшего исследования для алгоритма А1 размер рангового блока берется равный 4 и коэффициент ε , равный 300.

Таблица 7 – Зависимость результатов декомпрессии от алгоритма сжатия и параметров компрессии для изображений с малым количеством деталей

Алгоритм выбора доменного блока	Метод классификации	Размер рангового блока	ε	$t_{\text{комп, сек}}$	$t_{\text{декомп, сек}}$	Степень сжатия	SSIM
Первый подходящий (без разбиения)	-	4	300	2,12	2,50	4,38	0,9893
	Центр масс	4	300	1,04	2,09	4,39	0,9902
	Разница граничных значений	4	300	0,94	2,52	4,38	0,9891
Первый подходящий (с разбиением)	-	16	5	27,50	2,01	21,82	0,9886
	Центр масс	16	5	8,84	2,38	19,95	0,9903
	Разница граничных значений	16	5	8,90	2,17	20,19	0,9926
Доменный блок с минимальным СКО	-	8	-	46,89	2,49	17,52	0,9921
	Центр масс	8	-	14,34	2,56	17,61	0,9887
	Разница граничных значений	8	-	30,45	2,32	17,55	0,9891

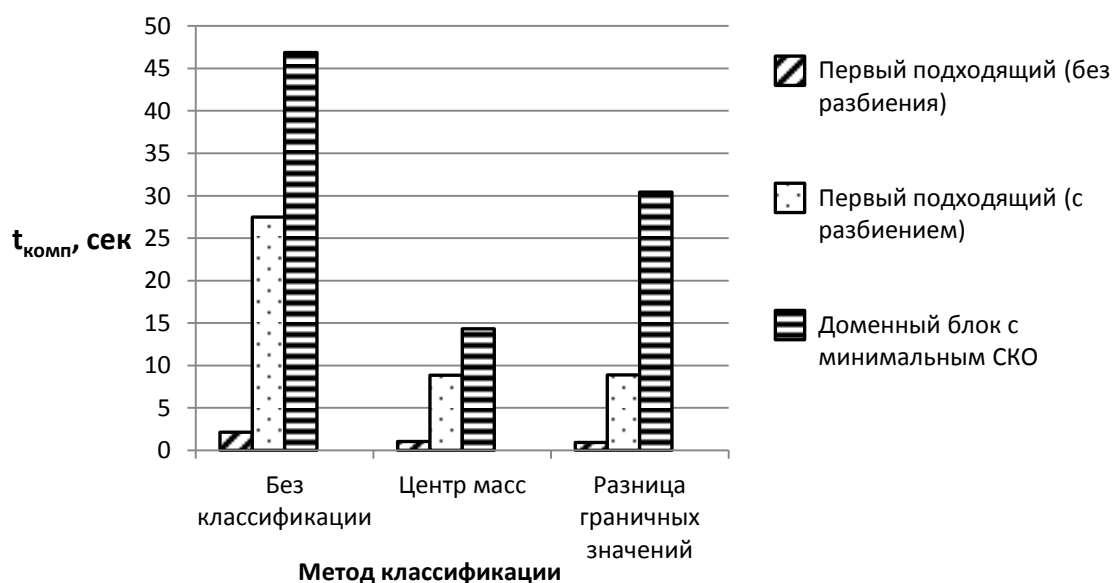


Рисунок 22 – Зависимость времени сжатия изображения от алгоритма и типа классификации (для изображений с малым количеством деталей)

Как видно из рисунка 22 и таблицы 7, наиболее эффективным с точки зрения времени сжатия является алгоритм А1, а поиск доменного блока с наименьшим СКО (алгоритм Б) занимает больше всего времени. Выигрыш по

времени алгоритмов A1 и A2 обуславливается отсутствием большого количества деталей в изображениях, что позволяет быстрее найти подходящий доменный блок.

В таблице 8 приводится исследование зависимости параметров декомпрессии (при использовании метода эталонного блока) от размера рангового блока и коэффициента ε .

Таблица 8 – Зависимость результатов декомпрессии от размера рангового блока и ε для изображений с малым количеством деталей

Размер рангового блока	ε	$t_{\text{комп}}$, сек	$t_{\text{декомп}}$, сек	Степень сжатия	SSIM
4	5	105,51	1,71	4,39	0,9849
	10	89,95	1,64	4,39	0,9854
	50	55,15	1,72	4,39	0,9854
	100	44,65	1,65	4,39	0,9851
	200	20,72	1,67	4,39	0,9841
	300	9,17	1,68	4,39	0,9834
8	5	22,94	1,83	17,51	0,9786
	10	19,59	1,79	17,55	0,9783
	50	15,19	1,67	17,59	0,9771
	100	13,62	1,83	17,55	0,9750
	200	11,01	1,76	17,51	0,9745
	300	8,33	1,75	17,55	0,9742

Таблица 9 – Зависимость времени сжатия от выбранного подхода для изображений с малым количеством деталей

Алгоритм выбора доменного блока	Метод классификации	Размер рангового блока	ε	$t_{\text{комп}}$, сек	$t_{\text{декомп}}$, сек	Степень сжатия	SSIM
Первый подходящий (без разбиения)	Разница граничных значений	4	300	0,94	2,52	4,38	0,9891
Первый подходящий (с разбиением)	-	16	5	27,50	2,01	21,82	0,9886
Первый подходящий (с разбиением)	Центр масс	16	5	8,84	2,38	19,95	0,9903
Метод эталонного блока	-	4	300	35,03	2,29	4,69	0,9918

Для сравнения результатов использования метода эталонного блока с алгоритмами A1 и A2 (при условии приблизительно равного качества декодируемого изображения) размер рангового блока берется равный 4 и коэффициент ε , равный 300.

Как видно из рисунка 23 и таблицы 9 использование классификации РГЗ обеспечивают заметный выигрыш по времени, а метода эталонного блока для данного типа изображений затрачивает больше времени, чем алгоритм A2. Наиболее эффективным по времени выполнения остается использование поиска первого подходящего доменного блока (алгоритм A1) с классификацией разницей граничных значений.

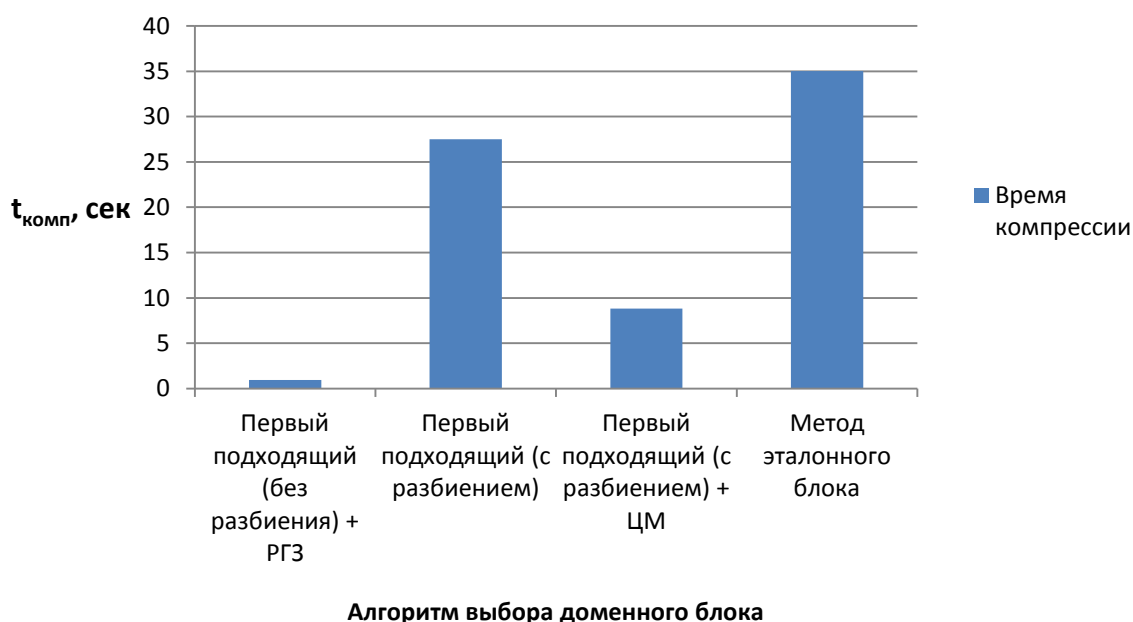


Рисунок 23 – Зависимость времени сжатия изображения от выбранного алгоритма (для изображений с малым количеством деталей)

4.4 Сжатие изображений с большим количеством деталей

В данном разделе исследования проводятся над набором изображений в оттенках серого, размером 160×160 пикселей [15].

В таблице 10 представлена зависимость параметров декомпрессии (при использовании алгоритма A1) от размера рангового блока и коэффициента компрессии.

Таблица 10 – Зависимость результатов декомпрессии от размера рангового блока и ϵ при использовании алгоритма A1 для изображений с большим количеством деталей

Размер рангового блока	ϵ	$t_{\text{комп, сек}}$	$t_{\text{декомп, сек}}$	Степень сжатия	SSIM
4	5	174,75	1,76	4,42	0,9780
	10	168,56	1,68	4,42	0,9779
	50	100,11	1,78	4,42	0,9743
	100	50,57	1,68	4,42	0,9682
	150	27,94	1,68	4,42	0,9624
	200	15,93	1,86	4,42	0,9548
8	5	30,48	2,07	17,55	0,8787
	10	29,95	1,92	17,51	0,8779
	50	28,92	2,26	17,55	0,8757
	100	26,54	1,79	17,51	0,8747
	150	23,19	1,75	17,47	0,8698
	200	21,11	1,79	17,51	0,8664

Из таблицы 10 можно видеть, что с увеличением коэффициента компрессии уменьшается не только время сжатия, но и качество декодируемого изображения. Для дальнейшего исследования для алгоритма A1 размер рангового блока берется равный 4 и коэффициент ϵ , равный 50.

Таблица 11 – Зависимость результатов декомпрессии от использованного алгоритма сжатия и параметров компрессии для изображений с большим количеством деталей

Алгоритм выбора доменного блока	Метод классификации	Размер рангового блока	ϵ	$t_{\text{комп, сек}}$	$t_{\text{декомп, сек}}$	Степень сжатия	SSIM
Первый подходящий (без разбиения)	-	4	50	131,24	2,15	4,39	0,9809
	Центр масс	4	50	69,36	2,41	4,39	0,9794
	Разница граничных значений	4	50	56,09	2,15	4,39	0,9888
Первый подходящий (с разбиением)	-	16	5	109,67	1,76	4,55	0,9751
	Центр масс	16	5	43,27	1,81	4,55	0,9787
	Разница граничных значений	16	5	37,42	2,13	4,54	0,9751
Доменный блок с минимальным СКО	-	4	-	225,11	2,31	4,42	0,9881
	Центр масс	4	-	57,06	2,05	4,39	0,9775
	Разница граничных значений	4	-	31,31	2,48	4,40	0,9723

Как видно из рисунка 24 и таблицы 11, наиболее эффективными с точки зрения времени сжатия является алгоритма А2 или Б, ускоренные классификацией разницей граничных значений.

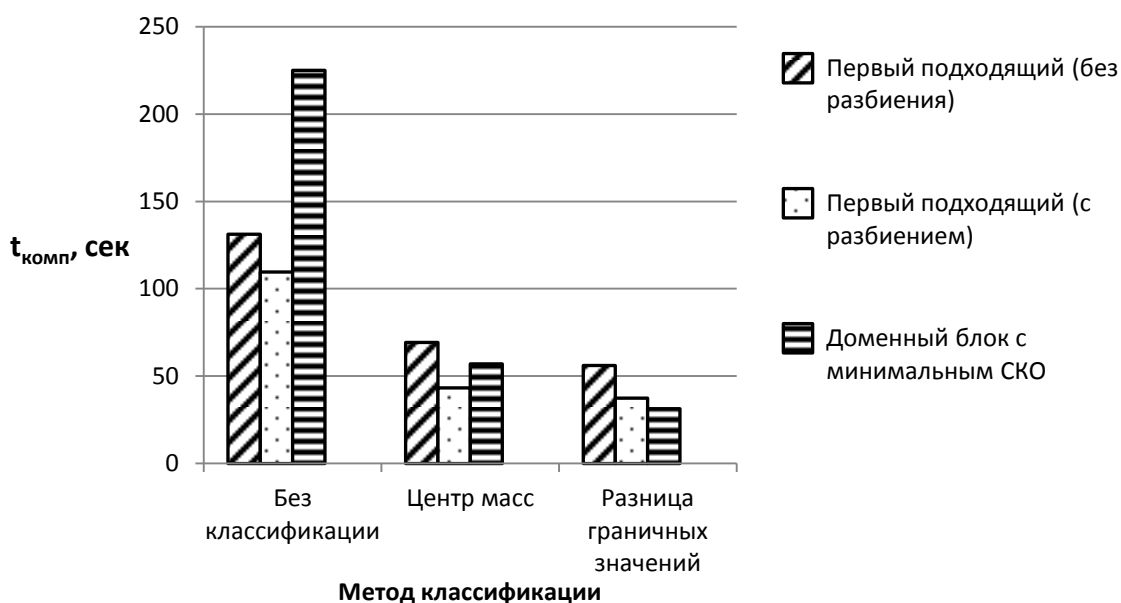


Рисунок 24 – Зависимость скорости сжатия изображения от алгоритма и типа классификации (для изображений с большим количеством деталей)

В таблице 12 приводится исследование зависимости параметров декомпрессии от размера рангового блока и коэффициента ϵ для изображений с большим количеством деталей при использовании метода эталонного блока.

Таблица 12 – Зависимость результатов декомпрессии от размера рангового блока и ϵ для изображений с большим количеством деталей

Размер рангового блока	ϵ	$t_{\text{комп}}, \text{сек}$	$t_{\text{декомп}}, \text{сек}$	Степень сжатия	SSIM
4	5	215,33	1,68	4,39	0,9773
	10	203,59	1,75	4,39	0,9772
	50	129,69	1,67	4,39	0,9725
	100	83,83	1,65	4,37	0,9647
	150	59,98	1,64	4,34	0,9568
	200	48,05	1,67	4,32	0,9485
8	5	38,17	1,73	17,55	0,8786
	10	37,78	1,75	17,55	0,8784
	50	36,67	1,76	17,55	0,8765
	100	33,79	1,64	17,55	0,8756
	150	30,69	1,68	17,55	0,8704
	200	29,11	1,61	17,63	0,8677

Для сравнения метода эталонного блока с алгоритмами А2 и Б размер рангового блока берется равный 4 и коэффициент ε , равный 100.

Таблица 13 – Зависимость времени сжатия от выбранного подхода для изображений с большим количеством деталей

Алгоритм выбора доменного блока	Метод классификации	Размер рангового блока	ε	$t_{\text{комп, сек}}$	$t_{\text{декомп, сек}}$	Степень сжатия	SSIM
Первый подходящий (с разбиением)	-	16	5	109,67	1,76	4,55	0,9751
Первый подходящий (с разбиением)	Разница граничных значений	16	5	37,42	2,13	4,54	0,9751
Доменный блок с минимальным СКО	Разница граничных значений	4	-	31,31	2,48	4,40	0,9723
Метод эталонного блока	-	4	100	103,77	2,35	4,37	0,9729

Как видно из рисунка 25 и таблицы 13, использование метода эталонного блока не обеспечивает выигрыша по времени по сравнению с алгоритмом А2. Наименьшее время сжатия позволяет получить использование поиска доменного блока с минимальным СКО с применением классификации разницей граничных значений (алгоритм Б).

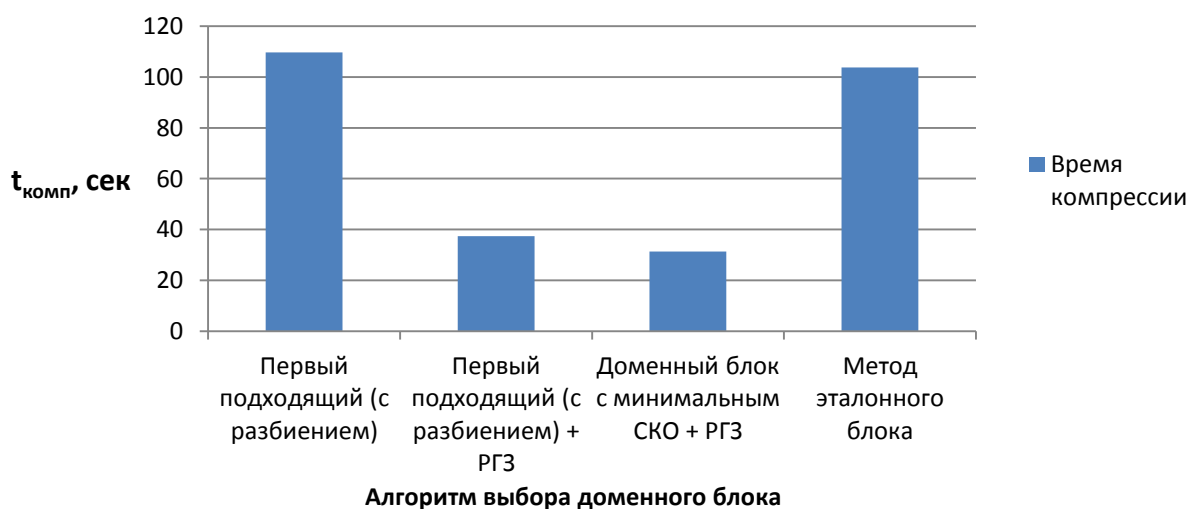


Рисунок 25 – Зависимость времени изображения от выбранного алгоритма (для изображений с большим количеством деталей)

4.5 Сжатие изображений с текстом

В данном разделе исследования проводятся над набором изображений в оттенках серого, размером 160×160 пикселей [15].

В таблице 14 представлена зависимость результатов декомпрессии (времени декомпрессии изображения, и SSIM - метрики качества декодированного изображения) от размера рангового блока и коэффициента компрессии, при условии использовании алгоритма поиска первого подходящего доменного блока без разбиения (алгоритма A1).

Таблица 14 – Зависимость результатов декомпрессии от размера рангового блока и ϵ при использовании алгоритма A1 для изображений с текстом

Размер рангового блока	ϵ	$t_{\text{комп}}, \text{сек}$	$t_{\text{декомп}}, \text{сек}$	Степень сжатия	SSIM
4	2	162,99	1,89	4,47	0,9291
	5	145,17	1,83	4,47	0,9300
	10	134,53	1,79	4,50	0,9264
	50	148,96	1,65	4,52	0,9290
	100	108,84	1,72	4,52	0,9265
	150	98,67	1,76	4,55	0,9244
8	2	37,38	1,76	17,71	0,6387
	5	36,78	1,79	17,67	0,6383
	10	25,71	1,73	17,71	0,6380
	50	25,24	1,67	17,71	0,6343
	100	25,15	1,78	17,71	0,6400
	150	25,08	1,76	17,71	0,6426

Из таблицы 14 можно видеть, что с увеличением коэффициента компрессии уменьшается не только время сжатия, но и качество декодируемого изображения. Для дальнейшего исследования для алгоритма A1 размер рангового блока берется равный 4 и коэффициент ϵ , равный 200. Также, если сравнить значения SSIM со значениями из таблиц 2, 6 и 7, можно сделать вывод, что декодированные изображения с текстом имеют наиболее плохое качество.

Как видно из рисунка 26 и таблицы 15, наиболее эффективным с точки зрения времени сжатия для данного типа изображений является выбор доменного блока с минимальным СКО (алгоритм Б) с применением классификации разницей граничных значений.

Таблица 15 – Зависимость результатов декомпрессии от использованного алгоритма сжатия и параметров компрессии для изображений с текстом

Алгоритм выбора доменного блока	Метод классификации	Размер рангового блока	ε	$t_{\text{комп, сек}}$	$t_{\text{декомп, сек}}$	Степень сжатия	SSIM
Первый подходящий (без разбиения)	-	4	200	148,96	1,93	4,52	0,9289
	Центр масс	4	200	64,17	2,44	4,55	0,9045
	Разница граничных значений	4	200	43,26	2,35	4,55	0,9128
Первый подходящий (с разбиением)	-	16	5	120,94	1,72	5,36	0,9152
	Центр масс	16	5	71,21	1,80	5,38	0,9012
	Разница граничных значений	16	5	52,24	1,91	5,22	0,9161
Доменный блок с минимальным СКО	-	4	-	136,17	2,10	4,44	0,9066
	Центр масс	4	-	96,77	2,34	16,81	0,9019
	Разница граничных значений	4	-	37,19	2,21	4,47	0,9014

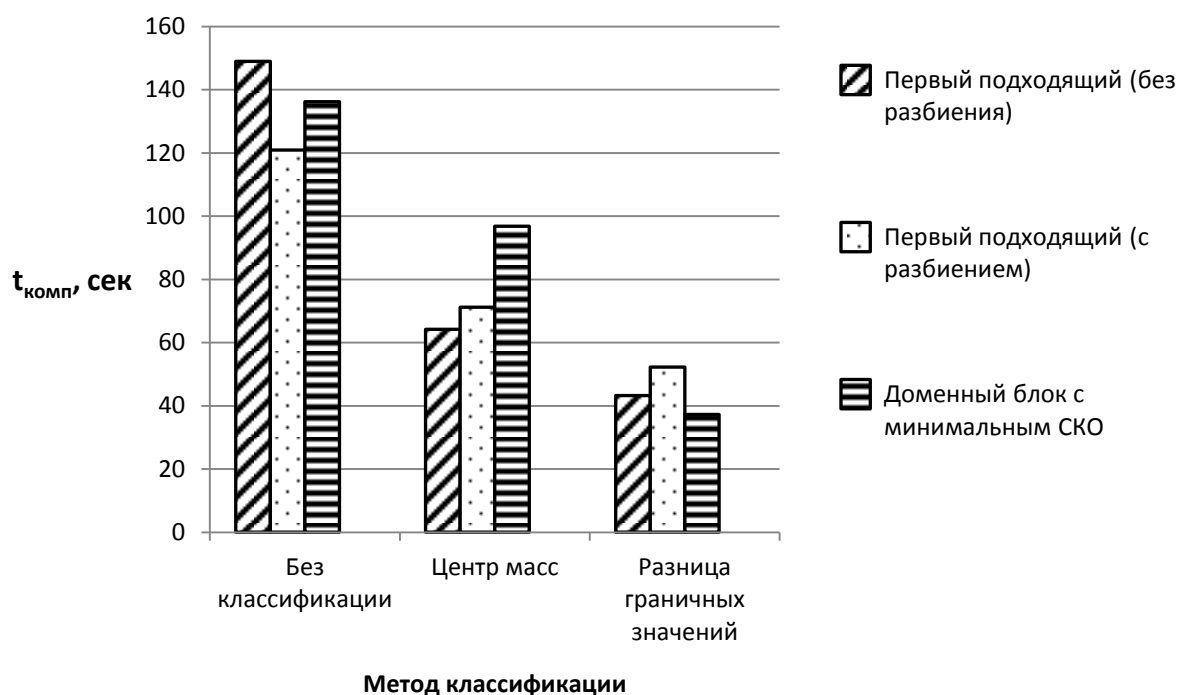


Рисунок 26 – Зависимость времени сжатия изображения от алгоритма и типа классификации (для изображений с текстом)

В таблице 16 приводится исследование зависимости параметров декомпрессии (при использовании метода эталонного блока) от размера рангового блока и коэффициента ε для изображений, содержащих текст.

Таблица 16 – Зависимость результатов декомпрессии от размера рангового блока и ϵ при сжатии методом эталонного блока для изображений с текстом

Размер рангового блока	ϵ	$t_{\text{комп}}, \text{сек}$	$t_{\text{декомп}}, \text{сек}$	Степень сжатия	SSIM
4	10	203,82	2,17	4,47	0,9317
	50	193,87	1,68	4,50	0,9306
	100	130,03	1,82	4,50	0,9265
	200	109,95	1,68	4,50	0,9232
	300	100,26	1,73	4,50	0,9179
	400	90,85	1,64	4,52	0,9144
8	5	49,29	1,97	17,63	0,6381
	10	50,34	1,71	17,63	0,6372
	50	48,71	2,22	17,67	0,6324
	100	41,89	1,68	17,67	0,6357
	150	35,09	1,64	17,71	0,6343
	200	30,82	1,65	17,67	0,6324

Для сравнения метода эталонного блока с алгоритмами A1 и A2(при условии приблизительно равного качества декодируемого изображения) размер рангового блока берется равный 4 и коэффициент ϵ , равный 200.

Таблица 17 – Зависимость времени сжатия от выбранного подхода для изображений, содержащих текст

Алгоритм выбора доменного блока	Метод классификации	Размер рангового блока	ϵ	$t_{\text{комп}}, \text{сек}$	$t_{\text{декомп}}, \text{сек}$	Степень сжатия	SSIM
Первый подходящий (с разбиением)	-	16	5	120,94	1,72	5,36	0,9152
Первый подходящий (с разбиением)	Разница граничных значений	16	5	52,24	1,91	5,22	0,9161
Доменный блок с минимальным СКО	Разница граничных значений	4	-	37,19	2,21	4,47	0,9014
Метод эталонного блока	-	4	200	109,95	2,24	4,49	0,9232

Как видно из рисунка 27 и таблицы 17, использование метода эталонного блока не обеспечивает выигрыша по времени по сравнению с алгоритмом A2.

Наименьшее время сжатия позволяет получить использование поиска доменного блока с применением классификации разниц граничных значений (алгоритм Б).

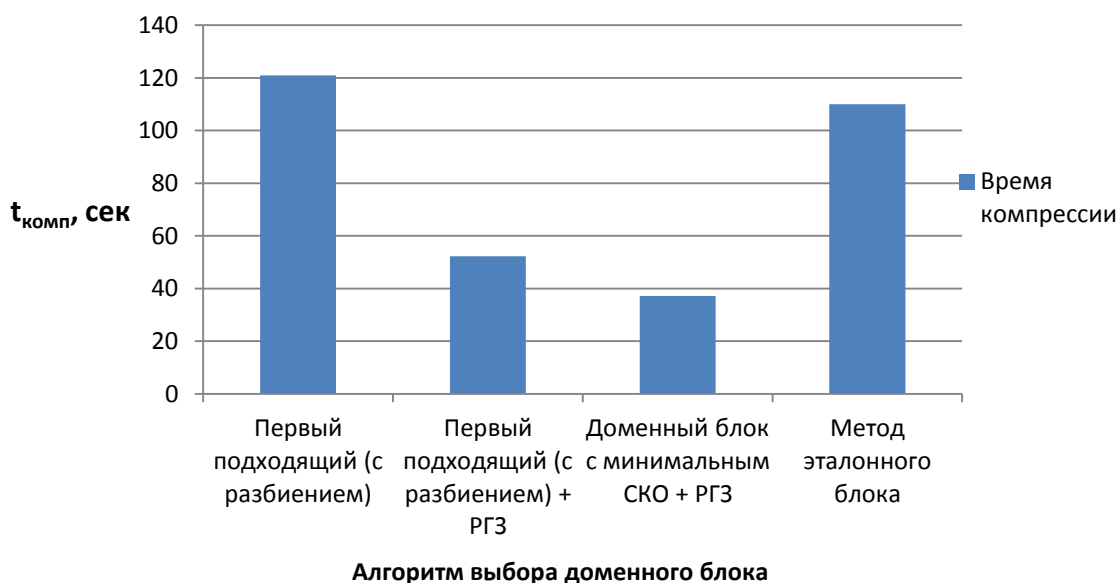


Рисунок 27 – Зависимость времени сжатия изображения от выбранного алгоритма (для изображений с текстом)

4.6 Сводная таблица результатов исследований сжатия изображений четырех типов

Далее приведена сводная таблица результатов исследований над наборами изображений 4 типов (портрет, изображения с маленьким количеством деталей, изображения с большим количеством деталей, текст), представленных в оттенках серого, размером 160×160 пикселей. В таблице 2 А1 – алгоритм поиска первого подходящего доменного блока без разбиения, А2 – алгоритм поиска первого подходящего доменного блока с разбиением, Б – поиск доменного блока с минимальным СКО, К – тип классификации, R – размер рангового блока, Н – столбец, в котором отмечен наиболее эффективный метод для исследуемых наборов изображений и алгоритмов.

Таблица 18 позволяет сравнить параметры компрессии, задаваемые для каждого типа изображений, при использовании каждого из алгоритмов сжатия и методов ускорения, при которых достигается относительно равное качество декодированного изображения.

Таблица 18 – Сводная таблица результатов исследований

Алгоритм поиска доменного блока	К	Тип изображения	Н	R	ε	$t_{\text{комп, сек}}$	$t_{\text{декомп, сек}}$	Степень сжатия	SSIM
A1	-	Портрет		4	150	9,21	1,89	4,53	0,9808
		Мало деталей		4	300	2,12	2,50	4,38	0,9893
		Много деталей		4	50	131,24	2,15	4,39	0,9809
		Текст		4	200	148,96	1,93	4,52	0,9289
	Центр масс	Портрет		4	150	4,06	2,01	4,49	0,9765
		Мало деталей		4	300	1,04	2,09	4,39	0,9902
		Много деталей		4	50	69,36	2,41	4,39	0,9794
		Текст		4	200	64,17	2,44	4,55	0,9045
	Разница граничных значений	Портрет	+	4	150	2,95	2,07	4,53	0,9773
		Мало деталей	+	4	300	0,94	2,52	4,38	0,9891
		Много деталей		4	50	56,09	2,15	4,39	0,9888
		Текст		4	200	43,26	2,35	4,55	0,9128
A2	-	Портрет		16	5	45,08	1,83	8,79	0,9793
		Мало деталей		16	5	27,50	2,01	21,82	0,9886
		Много деталей		16	5	109,67	1,76	4,55	0,9751
		Текст		16	5	120,94	1,72	5,36	0,9152
	Центр масс	Портрет		16	5	16,84	1,91	8,43	0,9797
		Мало деталей		16	5	8,84	2,38	19,95	0,9903
		Много деталей		16	5	43,27	1,81	4,55	0,9787
		Текст		16	5	71,21	1,80	5,38	0,9012
	Разница граничных значений	Портрет		16	5	15,86	2,45	8,45	0,9805
		Мало деталей		16	5	8,90	2,17	20,19	0,9926
		Много деталей		16	5	37,42	2,13	4,54	0,9751
		Текст		16	5	52,24	1,91	5,22	0,9161

Продолжение таблицы 18

Алгоритм поиска доменного блока	К	Тип изображения	Н	R	ε	t _{комп} , сек	t _{декомп} , сек	Степень сжатия	SSIM
Б	-	Портрет		8	-	35,72	2,15	17,67	0,9722
		Мало деталей		8	-	46,89	2,49	17,52	0,9921
		Много деталей		4	-	225,11	2,31	4,42	0,9881
		Текст		4	-	136,17	2,10	4,44	0,9066
	Центр масс	Портрет		8	-	15,75	2,13	17,71	0,9677
		Мало деталей		8	-	14,34	2,56	17,61	0,9887
		Много деталей		4	-	57,06	2,05	4,39	0,9775
		Текст		4	-	96,77	2,34	16,80	0,9019
	Разница граничных значений	Портрет		8	-	13,43	2,21	17,72	0,9711
		Мало деталей		8	-	30,45	2,32	17,55	0,9891
		Много деталей	+	4	-	31,31	2,48	4,40	0,9723
		Текст	+	4	-	37,19	2,21	4,47	0,9014
Метод эталонного блока	-	Портрет		4	300	35,86	2,42	4,81	0,9777
	-	Мало деталей		4	300	35,03	2,29	4,69	0,9918
	-	Много деталей		4	100	103,77	2,35	4,37	0,9729
	-	Текст		4	200	119,01	2,24	4,52	0,9259

4.7 Сжатие изображений разных размеров

В данном разделе исследования проводились над набором изображений в оттенках серого, размером, превышающим 160×160 пикселей и над изображениями не квадратной формы.

В таблице 19 представлена зависимость результатов декомпрессии от выбранного подхода и параметров сжатия для изображения размером 120×200 пикселей. В таблице 20 представлены аналогичные зависимости в случае сжатия изображения размером 160×160 пикселей.

Таблица 19 – Зависимость параметров декомпрессии от использованного алгоритма и методов ускорения для изображения размером 120×200

Алгоритм выбора доменного блока	Метод классификации	Размер рангового блока	ε	$t_{\text{комп, сек}}$	$t_{\text{декомп, сек}}$	Степень сжатия	SSIM
Первый подходящий (без разбиения)	-	4	150	7,71	1,81	4,37	0,9904
	Центр масс	4	150	5,81	1,97	4,39	0,9832
	Разница граничных значений	4	150	2,59	1,72	4,39	0,9906
Первый подходящий (с разбиением)	-	16	5	142,17	1,68	6,51	0,9844
	Центр масс	16	5	25,87	1,61	6,39	0,9687
	Разница граничных значений	16	5	24,69	1,62	6,23	0,9711
Доменный блок с минимальным СКО	-	8	-	31,34	1,62	17,58	0,9805
	Центр масс	8	-	8,67	1,61	17,58	0,9765
	Разница граничных значений	8	-	5,28	1,58	17,58	0,9759
Метод эталонного блока	-	4	300	47,42	1,61	4,54	0,9871

Таблица 20 – Зависимость параметров декомпрессии от использованного алгоритма и методов ускорения для изображения размером 160×160

Алгоритм выбора доменного блока	Метод классификации	Размер рангового блока	ε	$t_{\text{комп, сек}}$	$t_{\text{декомп, сек}}$	Степень сжатия	SSIM
Первый подходящий (без разбиения)	-	4	150	9,21	1,83	4,52	0,9808
	Центр масс	4	150	4,06	1,9	4,49	0,9765
	Разница граничных значений	4	150	2,95	1,89	4,51	0,9773
Первый подходящий (с разбиением)	-	16	5	45,08	1,79	7,99	0,9793
	Центр масс	16	5	16,84	1,78	7,76	0,9797
	Разница граничных значений	16	5	15,86	1,79	7,77	0,9805
Доменный блок с минимальным СКО	-	8	-	35,72	1,78	17,65	0,9722
	Центр масс	8	-	15,75	1,79	17,69	0,9677
	Разница граничных значений	8	-	15,75	1,68	17,69	0,9677
Метод эталонного блока	-	4	300	35,86	1,72	4,68	0,9777

В таблице 21 представлена зависимость параметров компрессии и декомпрессии от выбранного подхода и параметров сжатия для изображения размером 304×304 пикселей.

Таблица 21 – Зависимость параметров декомпрессии от использованного алгоритма и методов ускорения для изображения размером 304×304

Алгоритм выбора доменного блока	Метод классификации	Размер рангового блока	ε	$t_{\text{комп, сек}}$	$t_{\text{декомп, сек}}$	Степень сжатия	SSIM
Первый подходящий (без разбиения)	-	4	150	68,42	7,01	4,38	0,9826
	Центр масс	4	150	33,32	8,56	4,35	0,9894
	Разница граничных значений	4	150	18,02	6,38	4,38	0,9894
Первый подходящий (с разбиением)	-	16	5	506,02	7,64	7,54	0,9801
	Центр масс	16	5	189,61	6,26	7,21	0,9866
	Разница граничных значений	16	5	152,18	6,55	7,24	0,9869
Доменный блок с минимальным СКО	-	8	-	361,05	7,48	17,53	0,9822
	Центр масс	8	-	127,12	8,14	17,53	0,9795
	Разница граничных значений	8	-	109,15	7,71	17,42	0,9794
Метод эталонного блока	-	4	300	399,98	7,53	4,73	0,9873

Далее представлены рисунки, иллюстрирующие зависимость времени компрессии и качества декодируемого изображения для данных их таблиц таблицам 19, 20 и 21.

Как видно из рисунков 28, 29 и 30, в случае изменения пропорций изображения, или его размера, тенденция уменьшения времени выполнения алгоритмов А1, А2 и Б, при использовании классификаций сохраняется. При этом эффективность классификаций зависит от содержимого изображения: нельзя однозначно выявить преобладание классификации центром масс над классификацией разницей граничных значений (или наоборот). Наименьшее время сжатия во всех трех случаях дает выбор первого подходящего доменного

блока без разбиения (алгоритм A1). Метод эталонного блока дает преимущество по времени, относительно примененного без классификаций алгоритма A2 (который можно назвать классической реализацией выбора доменного блока для алгоритма фрактального сжатия).

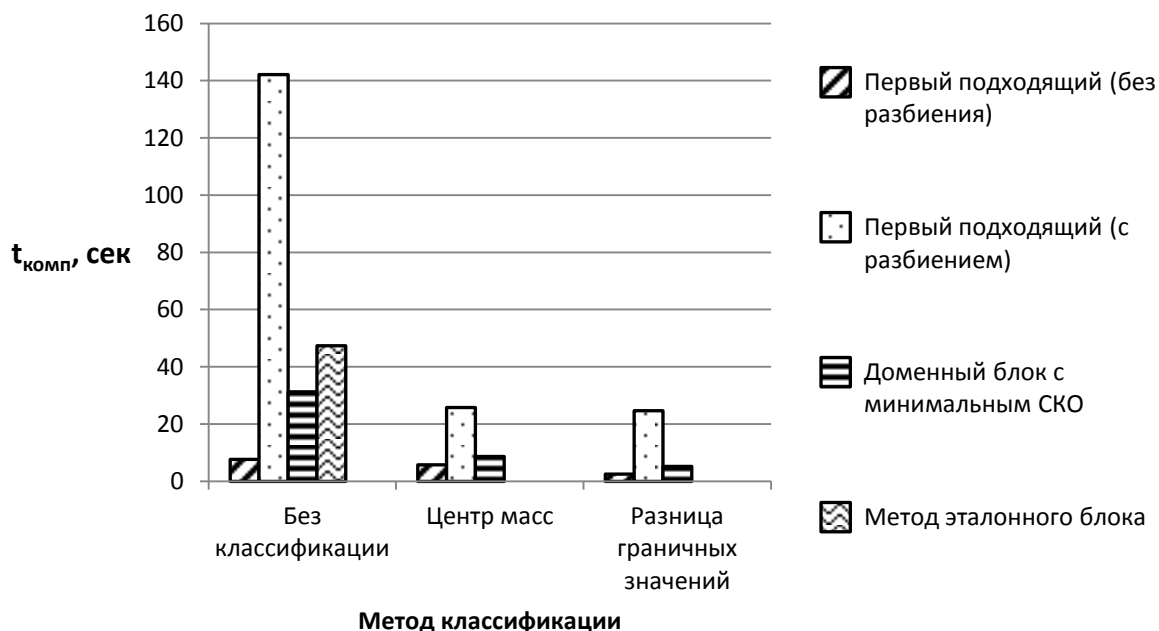


Рисунок 28 – Зависимость времени сжатия от примененного алгоритма и метода ускорения (размер изображения 120×200)

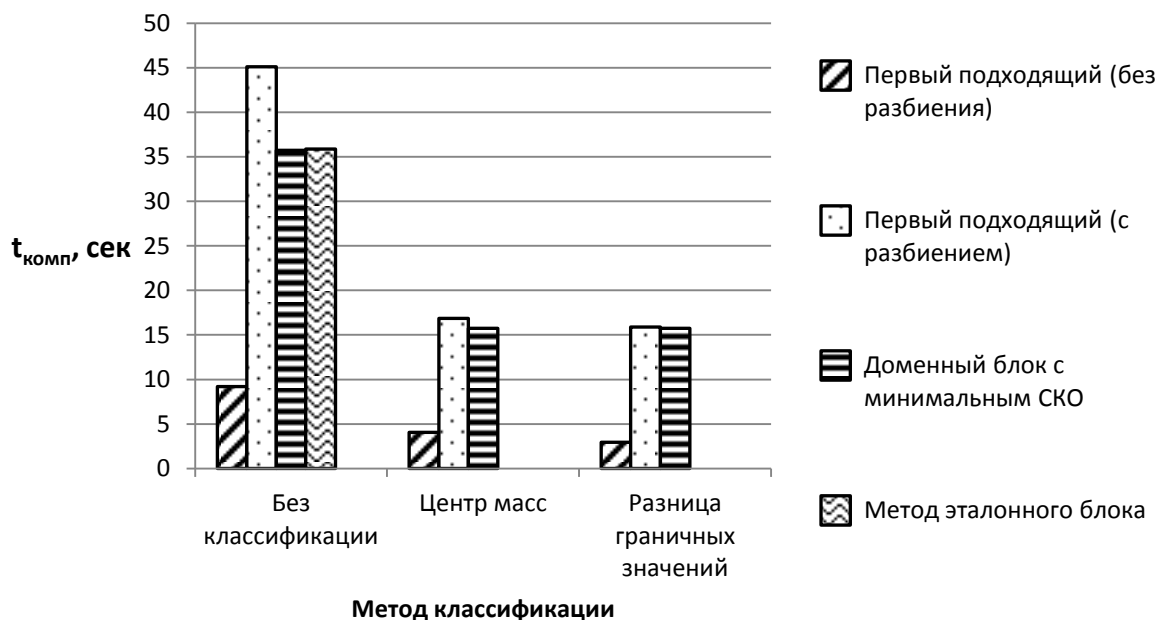


Рисунок 29 – Зависимость времени сжатия от примененного алгоритма и метода ускорения (размер изображения 160×160)

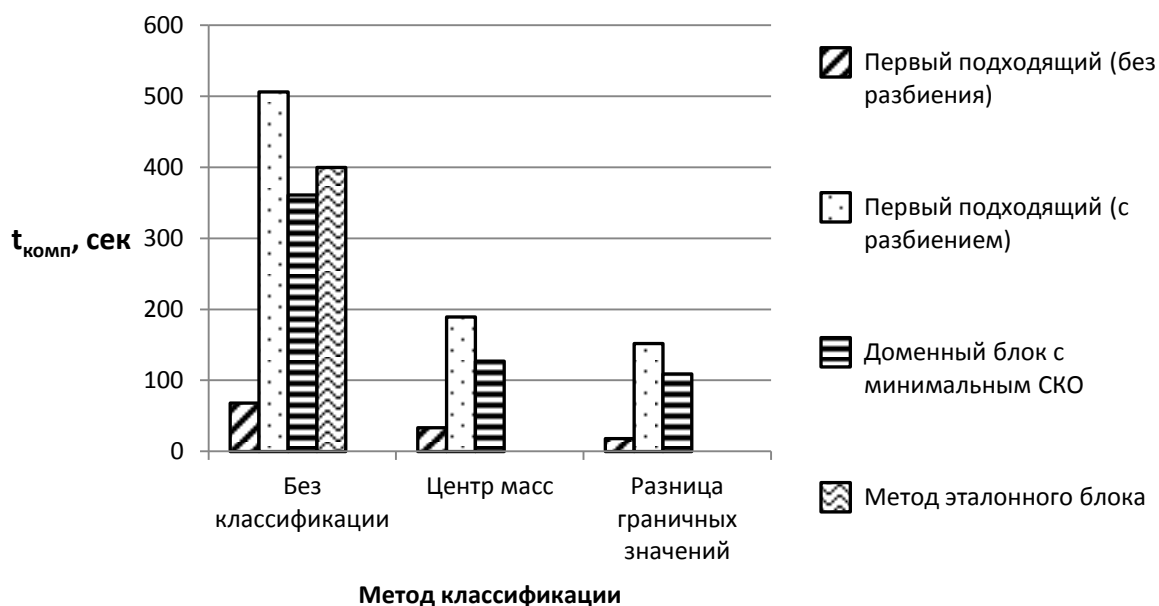


Рисунок 30 – Зависимость времени сжатия от примененного алгоритма и метода ускорения (размер изображения 304×304)

4.8 Сжатие цветных изображений

В данном разделе исследования проводились над набором цветных изображений, размером 160×160 пикселей [15]. Сжатие производилось в цветовых моделях RGB и YIQ.

В таблице 22 представлена зависимость параметров компрессии и декомпрессии алгоритма A1 от размера рангового блока и коэффициента компрессии для изображений в RGB представлении.

Таблица 22 – Зависимость параметров декомпрессии при использовании алгоритма A1 от размера рангового блока и ϵ (цветовая модель RGB)

Размер рангового блока	ϵ	$t_{\text{комп, сек}}$	$t_{\text{декомп, сек}}$	Степень сжатия	SSIM
4	10	211,45	6,69	1,46	0,9953
	50	163,12	8,31	1,46	0,9927
	100	60,78	8,28	1,47	0,9894
	200	31,31	8,18	1,47	0,9849
	300	18,99	7,59	1,47	0,9818
	400	6,29	8,03	1,47	0,9801
8	10	68,86	10,34	5,87	0,9715
	50	61,09	7,64	5,87	0,9708
	100	49,33	6,77	5,82	0,9690
	200	21,93	6,88	5,82	0,9637
	300	13,85	6,58	5,82	0,9585
	400	9,75	6,52	5,87	0,9540

Из таблицы 22 можно видеть, что и для цветного изображения с увеличением коэффициента компрессии уменьшается и время сжатия, и качество декодируемого изображения. Для дальнейшего исследования для алгоритма A1 размер рангового блока берется равный 4 и коэффициент ϵ , равный 300.

Таблица 23 – Зависимость параметров декомпрессии от выбранного алгоритма (цветовая модель RGB)

Алгоритм выбора доменного блока	Размер рангового блока	ϵ	$t_{\text{комп, сек}}$	$t_{\text{декомп, сек}}$	Степень сжатия	SSIM
Первый подходящий (без разбиения)	4	300	18,99	7,78	1,46	0,9798
Первый подходящий (с разбиением)	16	10	154,74	8,51	2,28	0,9725
Доменный блок с минимальным СКО	8	-	55,91	6,61	5,87	0,9721

Как видно из рисунка 31 и таблицы 23, наиболее эффективным с точки зрения затрачиваемого времени, будет выбор первого подходящего доменного блока без разбиения с размером рангового блока, равным 4 и ϵ – 300.

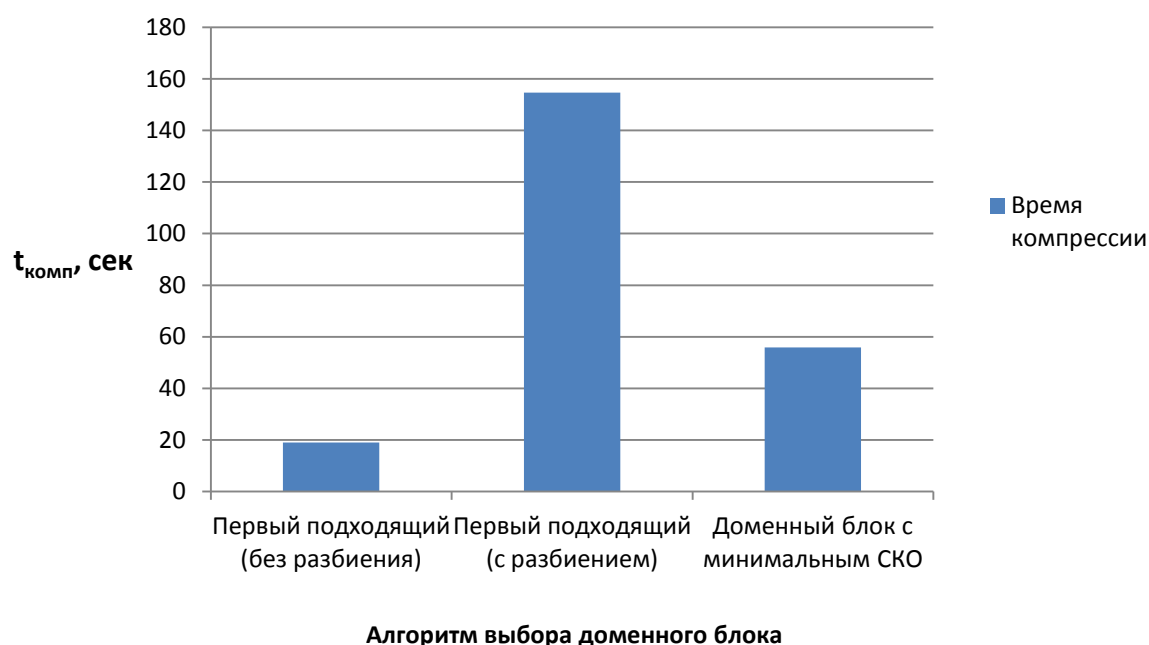


Рисунок 31 – Зависимость времени сжатия изображения от алгоритма и типа классификации (цветовая модель RGB)

В таблице 24 представлена зависимость параметров компрессии и декомпрессии алгоритма A1 от размера рангового блока и коэффициента компрессии для изображений в YIQ представлении.

Таблица 24 – Зависимость параметров декомпрессии при использовании алгоритма A1 от размера рангового блока и ε (цветовая модель YIQ)

Размер рангового блока	ε	$t_{\text{комп}}$, сек	$t_{\text{декомп}}$, сек	Степень сжатия	SSIM
4	10	498,23	11,73	1,46	0,8894
	50	178,73	11,09	1,46	0,9085
	100	108,17	10,98	1,46	0,9143
	200	96,38	12,76	1,46	0,9128
	300	32,51	12,65	1,46	0,9101
	400	16,56	10,97	1,46	0,9078
8	10	209,59	13,02	5,87	0,8789
	50	108,41	10,87	5,87	0,8535
	100	67,89	10,61	5,87	0,8256
	200	52,29	10,64	5,87	0,8260
	300	40,37	10,55	5,87	0,8182
	400	33,07	10,53	5,87	0,8087

Из таблицы 24 можно видеть, что и для цветного изображения с увеличением коэффициента компрессии уменьшается и время сжатия, и качество декодируемого изображения. Для дальнейшего исследования для алгоритма A1 размер рангового блока берется равный 4 и коэффициент ε , равный 200.

Таблица 25 – Зависимость параметров декомпрессии от выбранного алгоритма (цветовая модель YIQ)

Алгоритм выбора доменного блока	Размер рангового блока	ε	$t_{\text{комп}}$, сек	$t_{\text{декомп}}$, сек	Степень сжатия	SSIM
Первый подходящий (без разбиения)	4	200	96,38	12,76	1,46	0,9128
Первый подходящий (с разбиением)	16	10	220,31	10,42	4,61	0,9031
Доменный блок с минимальным СКО	8	-	200,37		5,87	0,9132

Как видно из рисунка 32 и таблицы 25, наиболее эффективным с точки зрения затрачиваемого времени, будет выбор первого подходящего доменного

блока без разбиения с размером рангового блока, равным 4 и эpsilon – 200, а выбор доменного блока с наименьшим СКО не дает большого выигрыша по времени.

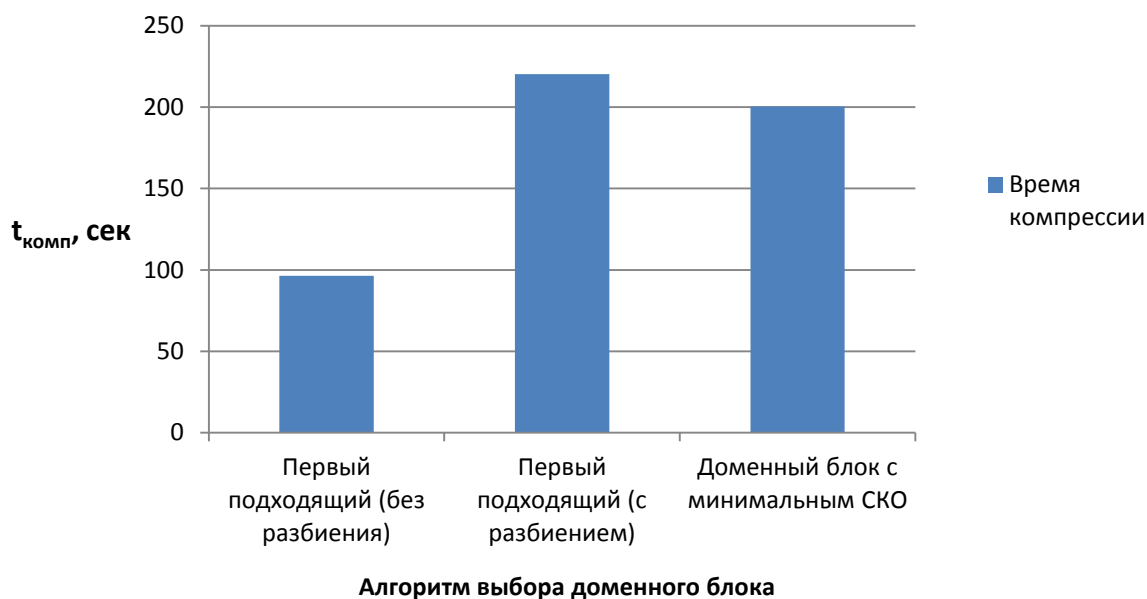


Рисунок 32 – Зависимость времени сжатия изображения от алгоритма и типа классификации (цветовая модель YIQ)

В таблице 26 приведены результаты исследования зависимости времени сжатия и качества декодируемого изображения от типа изображения (цветное или в оттенках серого) и размера рангового блока. Доменный блок выбирается с минимальным СКО.

Таблица 26 – Зависимость параметров декомпрессии от размера рангового блока и типа изображения

Тип изображения	Размер рангового блока	$t_{\text{комп, сек}}$	$t_{\text{декомп, сек}}$	Степень сжатия	SSIM
В оттенках серого	4	136,96	1,68	4,39	0,9958
	8	31,26	1,75	17,55	0,9721
	16	6,81	1,62	70,19	0,9030
RGB	4	337,02	5,28	1,46	0,9957
	8	51,53	7,75	5,87	0,9716
	16	10,58	6,52	23,47	0,9049
YIQ	4	536,67	10,23	1,46	0,9844
	8	170,18	9,89	5,87	0,9688
	16	37,88	9,69	23,47	0,9008

Данные из таблицы 26 указывают на то, что время сжатия цветного изображения ожидаемо больше времени сжатия изображения в оттенках серого, однако время обработки возрастает не в 3 раза, как этого можно было бы ожидать, а меньше. Это достигается за счет того, что для всех трех цветовых компонент рангового блока запускается единый цикл перебора доменных блоков, а не отдельный цикл для каждой компоненты. Для каждого типа изображения можно видеть, что чем больше размер рангового блока, тем меньше время компрессии и хуже качество декодируемого изображения. Также можно видеть, что для цветного изображения увеличивается время декомпрессии.

В таблице 27 представлена зависимость параметров компрессии и декомпрессии от используемого алгоритма и выбранного типа изображения.

Таблица 27 – Зависимость параметров декомпрессии от используемого алгоритма и типа изображения

Алгоритм выбора доменного блока	Тип изображения	Размер рангового блока	ε	$t_{\text{комп, сек}}$	$t_{\text{декомп, сек}}$	Степень сжатия	SSIM
Первый подходящий (без разбиения)	В оттенках серого	4	150	9,21	1,89	4,53	0,9808
	RGB	4	300	18,99	7,78	1,46	0,9798
	YIQ	4	200	96,38	12,76	1,46	0,9128
Первый подходящий (с разбиением)	В оттенках серого	16	5	45,08	1,83	8,79	0,9793
	RGB	16	10	154,74	8,51	2,28	0,9725
	YIQ	16	10	220,31	10,42	4,61	0,9031
Доменный блок с минимальным СКО	В оттенках серого	8	-	35,72	2,15	17,67	0,9722
	RGB	8	-	55,91	6,61	5,87	0,9721
	YIQ	8	-	200,37	11,23	5,87	0,9132

Как видно из рисунка 33 и таблицы 27, и для изображения в оттенках серого и для цветных изображений обеих моделей, при условии одинакового содержимого изображения, наилучший результат по времени предоставляет использование алгоритма A1, а наихудший – A2.

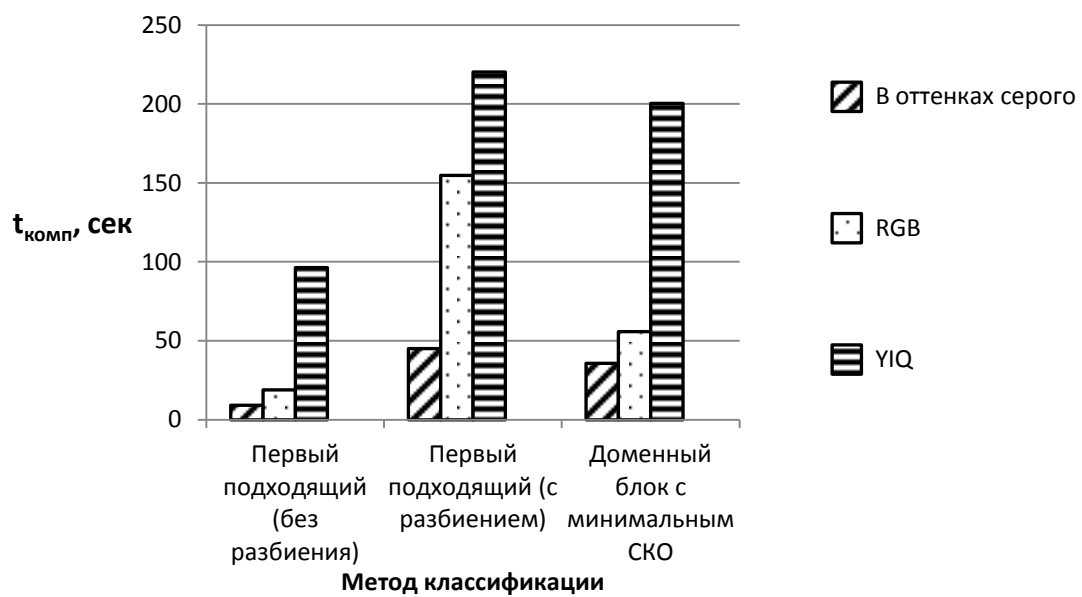


Рисунок 33 – Зависимость времени сжатия изображения от алгоритма и типа изображения

ЗАКЛЮЧЕНИЕ

В соответствии с заданием выпускной квалификационной работы разработана автоматизированная система фрактального сжатия изображения, позволяющая выбрать сжимаемое изображение, вариант реализации алгоритма и метод ускорения и предоставляющая информацию о времени декомпрессии и качестве декодированного изображения.

Проведены исследования зависимости времени фрактального сжатия и декомпрессии от параметров компрессии, содержимого изображения и его типа. Для исследованных наборов изображений определены наиболее эффективные (с точки зрения времени сжатия и качества декодируемого изображения) параметры компрессии, используемый вариант реализации и метод ускорения. Из проведенных исследований можно сделать вывод, что для исследованных реализаций фрактального сжатия применение метода предварительной классификации блоков позволяет сократить время сжатия, а параметры компрессии (размер рангового блока, коэффициент ε) зависят от содержимого изображения.

Основные положения и результаты выпускной квалификационной работы магистра представлялись, докладывались и обсуждались на Международной научно-технической конференции «Перспективные информационные технологии – 2017» (Самара, 2017), доклад был опубликован в сборнике трудов конференции [16]. По результатам исследований опубликованы статьи в сборниках трудов по материалам Международной заочной научно-практической конференции «Научное сообщество студентов XXI века» (Москва, 2018) [17] и «Технические и математические науки. Студенческий научный форум» (Новосибирск, 2018) [18]. Также результаты опубликовывались в сборнике трудов Международной научно-технической конференции «Перспективные информационные технологии – 2018» (Самара, 2018) [19] и в трудах Международного симпозиума «Надежность и качество» (Пенза, 2018) [20].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Данисюк, А.А. Фрактальное сжатие изображений [Текст]/А.А. Данисюк, А.А. Полупанов// Перспективные информационные технологии и интеллектуальные системы: сб. науч. Трудов / Таганрог. гос. радиотех. ун-т. – 2006. – Вып. 4. – С. 24 – 29.
- 2 Кудрина, М.А. Компьютерная графика [Текст]: учеб. / М.А. Кудрина, К.Е. Климентьев. – Самара: Изд-во Самар. гос. аэрокосм. ун-та, 2013. – 138 с.
- 3 Уэлстид С. Фракталы и вейвлеты для сжатия изображений в действии [Текст]/ С. Уэлстид. – М.: Триумф, 2003. – 320 с.
- 4 Аффинное преобразование [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Аффинное_преобразование (дата обращения: 15.04.2018 г.).
- 5 Попов, Е. А. Классический алгоритм фрактального сжатия изображений [Текст] / Е. А. Попов, А. В. Холодков // Вестник Алтайской государственной педагогической академии. Сер.: Естественные и точные науки. - 2011. - Вып. 7. - С. 42-46 : табл. - Библиогр.: с. 46
- 6 YIQ [Электронный ресурс]. – URL: <https://ru.wikipedia.org/wiki/YIQ> (дата обращения: 15.04.2018 г.).
- 7 Ансон, Л. Фрактальное сжатие изображения [Текст] // Л. Ансон, М. Барнсли. - Мир ПК, 1992, № 4, с. 52 – 58.
- 8 Леоненков, А. В. Нотация и семантика языка UML [Электронный ресурс]/ А.В. Леоненков. – Интернет-университет информационных технологий. <http://www.intuit.ru/department/pl/umlbasics> (дата обращения: 22.04.2018 г.).
- 9 Зеленко, Л.С. Методические указания к лабораторному практикуму по дисциплине «Технологии программирования» [Электронный ресурс]/Л.С.Зеленко. – СГАУ, 2014. – 65 с.
- 10 Гома, Х. UML. Проектирование систем реального времени, параллельных и распределенных приложений: Пер. с англ. [Текст] – М.: ДМК Пресс, 2011. – 704 с.

- 11 Программное обеспечение компьютеров. [Электронный ресурс]. – URL: <http://book.kbsu.ru/theory/chapter6/> (дата обращения: 28.04.2018 г.).
- 12 C# [Электронный ресурс]. – URL: <https://msdn.microsoft.com/ru-ru/library/kx37x362.aspx> (дата обращения: 28.04.2018 г.).
- 13 Внешнее описание программного средства [Электронный ресурс]. – URL: <http://studend.ru/gotovye-raboty/lektsiya-po-teme-vneshnee-opisanie-programmnogo-srdestva-l012.html> (дата обращения: 30.04.2018 г.).
- 14 Wang Z., Bovik A.C., Sheikh H.R., Simoncelli E.P. Image Quality Assessment: From Error Visibility to Structural Similarity. – IEEE TRANSACTIONS ON IMAGE PROCESSING, 2004. – 14 с.
- 15 Fractal image repository [Электронный ресурс]. – URL: <https://github.com/ViktoriaWofC/Fractl/tree/master/documents/fractImage> (дата обращения: 18.04.2018 г.).
- 16 Сахибназарова, В.Б. Исследование зависимости скорости фрактального сжатия изображения от параметров сжатия [Текст]/В.Б. Сахибназарова, М.А. Кудрина//Перспективные информационные технологии (ПИТ 2017): труды Международной научно-практической конференции / под ред. С.А. Прохорова. – Самара: Издательство Самарского научного центра РАН, 2017 – С. 161-164
- 17 Сахибназарова, В.Б. Применение предварительной классификации данных для повышения скорости фрактального сжатия изображений [Текст]/В.Б. Сахибназарова // Научное сообщество студентов XXI столетия: сб. науч. тр. по мат-лам Междунар. заоч. науч-практ. конф. апрель 2018 г. – Москва: Изд-во «МЦНО», 2018. – С. 75 – 79
- 18 Сахибназарова, В.Б. Использование эталонного метода для увеличения скорости фрактального сжатия изображений [Текст]/В.Б. Сахибназарова //Технические и математические науки. Студенческий научный форум: сб. науч. тр. по мат-лам Междунар. заоч. науч-практ. конф. апрель 2018 г. – Новосибирск: Изд-во АНС «СибАК», 2018. – С. 120 – 124

- 19 Сахибназарова, В.Б. Исследование алгоритмов фрактального сжатия изображений [Текст]/В.Б. Сахибназарова, М.А. Кудрина//Перспективные информационные технологии (ПИТ 2018): труды Международной научно-практической конференции / под ред. С.А. Прохорова. – Самара: Издательство Самарского научного центра РАН, 2018 – С. 185-188
- 20 Сахибназарова, В.Б. Исследование вариантов реализации и методов ускорения фрактального сжатия изображения [Текст]/В.Б. Сахибназарова, М.А. Кудрина// Труды Международного симпозиума НАДЕЖНОСТЬ И КАЧЕСТВО / под ред. Юркова Н.К. – Пенза: Издательство Пензенского государственного университета, 2018. – Т.1. – С. 333-337.

ПРИЛОЖЕНИЕ А

Руководство пользователя

А.1 Назначение системы

Программная система предназначена для выполнения фрактальной компрессии и декомпрессии изображений. Она позволяет пользователю определять тип сжимаемого изображения, настраивать параметры сжатия и декомпрессии. Также система выводит на экран сжимаемое изображение, декодированное изображение, время сжатия и декомпрессии и оценку качества декодированного изображения.

А.2 Требования к аппаратным и программным средствам

Для обеспечения условия работы системы, необходимо использовать следующие технические средства:

- рабочую станцию (компьютер).

Минимальные требования к техническим характеристикам и функционированию рабочей станции:

- процессор – Intel Pentium не менее 1,5 ГГц;
- объем оперативной памяти – 3 Гб;
- свободное место на диске – 9000 Мб;
- операционная система: Windows 7, Windows 8.

А.3 Установка программы

Для установки системы скопируйте с установочного диска папку «FractalCompression» на персональный компьютер в произвольную директорию. Папка «FractalCompression» содержит файл Fract.exe, папку «baseImage» (содержащую базовые изображения для декомпрессии) и папку «resultFiles» (в которой после сжатия будет помещаться файл с фрактальным кодом). Программа запускается двойным щелчком мыши по файлу Fract.exe.

А.4 Описание интерфейса пользователя

Окно программы открывается непосредственно при запуске и предоставляет основные функции по управлению системой. Интерфейс программы представлен на рисунке А.1.

Как видно из рисунка, в верхней части окна программы располагаются панели настроек параметров компрессии и декомпрессии.

Первой представлена группа радио-кнопок и выпадающий список, позволяющий выбрать тип изображения и цветовую модель (рисунок А.2).

Далее располагаются выпадающие списки, позволяющие выбрать алгоритм поиска доменного блока и используемую классификацию (рисунок А.3 и рисунок А.4).

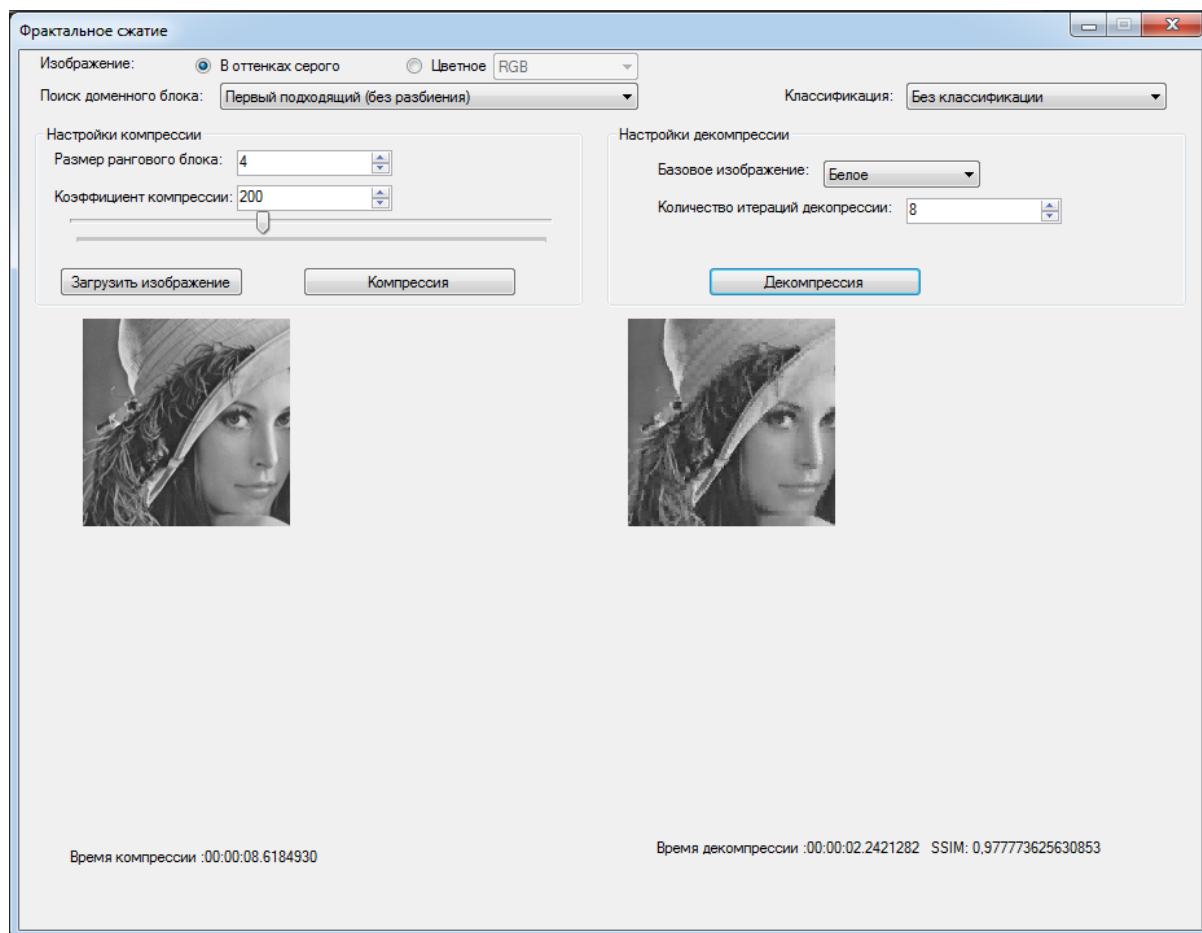


Рисунок А.1 – Интерфейс разработанной системы

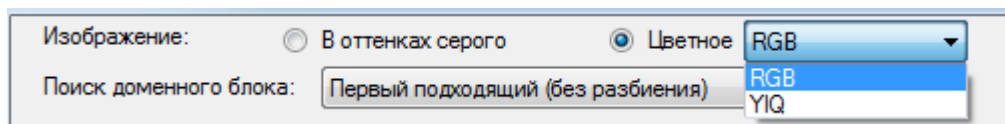


Рисунок А.2 – Панель выбора типа изображения и цветовой модели

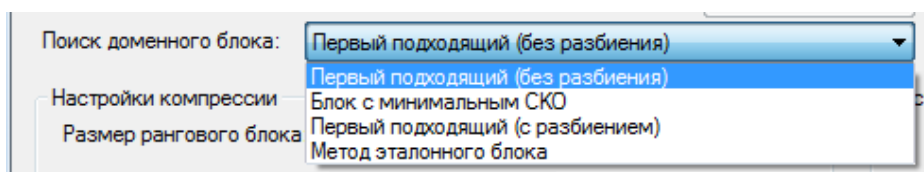


Рисунок А.3 – Панель выбора алгоритма поиска доменного блока

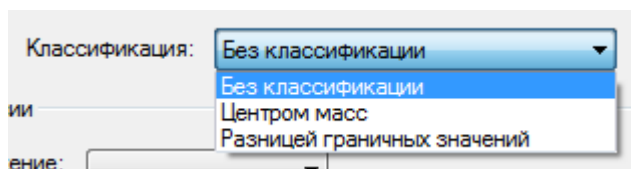


Рисунок А.4 – Панель выбора используемой классификации

Ниже размещены панели ввода коэффициента компрессии, размера рангового блока (рисунок А.5) и параметров декомпрессии (рисунок А.6).

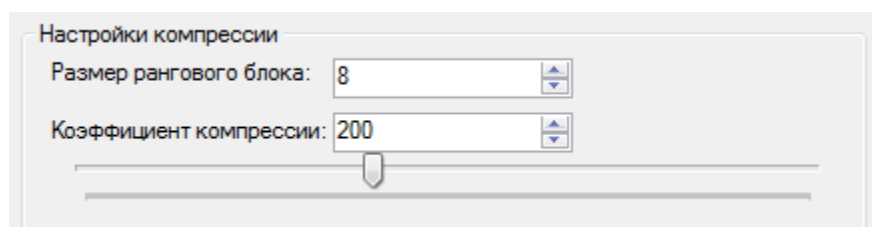


Рисунок А.5 – Панель ввода коэффициента компрессии и размера рангового блока

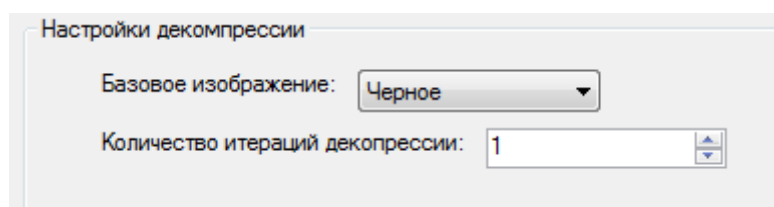


Рисунок А.6 – Панель выбора параметров декомпрессии

За ними располагаются кнопка вызова окна выбора сжимаемого изображения, кнопки запуска процесса компрессии и декомпрессии (рисунок А.1).

Под областью настроек параметров на экран выводятся сжимаемое и декодированное изображение, а в нижней части программного окна (см. рисунок А.1) – время компрессии, время декомпрессии и SSIM (метрика качества декодированного изображения).

ПРИЛОЖЕНИЕ Б

Листинг программы

//Класс компрессии

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Fract
{
    public class Compress : Compression
    {
        private List<Rang> rangList = new List<Rang>();
        private int r;//размер рангового блока
        private int[,] pix;
        private int width;//ширина картинки
        private int height;//высота картинки
        private int n;//количество блоков по высоте
        private int m;//количество блоков по ширине
        private double epsilon;//коэффициент компрессии
        private List<Rang> rangListR = new List<Rang>();
        private List<Rang> rangListG = new List<Rang>();
        private List<Rang> rangListB = new List<Rang>();
        private List<Rang> rangListY = new List<Rang>();
        private List<Rang> rangListI = new List<Rang>();
        private List<Rang> rangListQ = new List<Rang>();

        public Compress(int[,] pix, int r, double epsilon)    {
            this.pix = pix;
            this.r = r;
            this.width = pix.GetLength(1);//pix[0].Length;
            this.height = pix.GetLength(0); //pix.Length;
            this.m = width / r;
            this.n = height / r;
            this.epsilon = epsilon;
        }

        public List<Rang> getRangList()    {
            return rangList;
        }

        public List<Rang> getRangListComponent(string component)    {
            if(component.Equals("R"))        return rangListR;
            else if (component.Equals("G"))    return rangListG;
            else if (component.Equals("B"))    return rangListB;
            else if (component.Equals("Y"))    return rangListY;
            else if (component.Equals("I"))    return rangListI;
            else if (component.Equals("Q"))    return rangListQ;
            else return rangList;
        }
    }
}
```

```

public int getR()    {
    return r;
}

public void compressImage(string searchDomen, string imageColor)    {
    if (imageColor.Equals("gray")) {
        if (searchDomen.Equals("first<eps"))        compressImageFirst();
        else if (searchDomen.Equals("min"))        compressImageMin();
        else if (searchDomen.Equals("min and <eps"))    compressImageDivide();
        else if (searchDomen.Equals("test"))        compressImageTest();
        else if (searchDomen.Equals("etalons"))        compressImageEtalons();
    } else {
        if (searchDomen.Equals("first<eps"))        compressImageFirstColor(imageColor);
        else if (searchDomen.Equals("min"))        compressImageMinColor(imageColor);
        else if (searchDomen.Equals("min and <eps"))    compressImageDivideColor(imageColor);
    }
}

```

//Поиск первого подходящего доменного блока без разбиения (алгоритм A1)

```

public void compressImageFirst()    {
    bool b = false;
    int[,] rang = new int[r, r]; // ранговый блок
    //перебор ранговых блоков
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)    {
            //выделяем ранговый блок
            for (int ii = 0; ii < r; ii++)
                for (int jj = 0; jj < r; jj++)
                    rang[ii, jj] = pix[r * i + ii, r * j + jj];
            //пербор доменных блоков
            getDomenBlocFirst(rang, 1, j * r, i * r);
        }
}

```

//Метод эталонного блока

```

public void compressImageTest()    {
    bool b = false;
    int R = r; // k; //размер рангового блока
    int D = R * 2; //размер доменного блока
    int N = height / R; //количество блоков по высоте
    int M = width / R; //количество блоков по ширине
    int[,] rang = new int[r, r]; // ранговый блок
    int[,] domen = new int[R, R]; //уменьшенный доменный блок
    int[,] domenBig = new int[D, D]; //доменный блок
    int id = 0;
    int jd = 0;
    int[,] testRang = new int[r, r]; // etalon
    testRang = createEtalon(5);

    List<List<double>>> domenSKOList = new List<List<double>>>();

    //вычисляем все СКО для доменных блоков
    while ((id < N - 1) && (b == false))    {
        jd = 0;
        while ((jd < M - 1) && (b == false))    {

```

```

int sum = 0;
//выделяем доменный блок
for (int i = 0; i < D; i++)
    for (int j = 0; j < D; j++) {
        domenBig[i, j] = pix[R * id + i, R * jd + j];
        //domenBig[i, j] = pix[id + i, jd + j];
    }

//уменьшаем его усреднением
domen = reduceBlock(domenBig);

domenSKOList.Add(getAllSKO(testRang, domen));
jd++;
}
id++;
}

double[] so;
double s, o, sko;
Color colorTest, colorRang;
List<double> rangSKOList = new List<double>();
//перебор ранговых блоков
for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++) {
        //выделяем ранговый блок
        for (int ii = 0; ii < r; ii++)
            for (int jj = 0; jj < r; jj++)
                rang[ii, jj] = pix[r * i + ii, r * j + jj];

        sko = 0;
        so = getSO(testRang, rang);
        s = so[0];
        o = so[1];

        for (int iz = 0; iz < R; iz++)
            for (int jz = 0; jz < R; jz++) {
                colorRang = Color.FromArgb(rang[iz, jz]);
                colorTest = Color.FromArgb(testRang[iz, jz]);
                double per = (s * colorRang.R + o) - colorTest.R;
                sko = sko + (per) * (per);
            }
        rangSKOList.Add(sko);
        //пербор доменных блоков
        getDomenBlocTest(rang, 1, j * r, i * r, sko, domenSKOList);
    }
double tuu = rangSKOList[1];
}

```

//Выбор подходящего доменного блока для алгоритма A1

```

public void getDomenBlocFirst(int[,] rang, int k, int x0, int y0) {
    //x - начальная координата блока
    //y - начальная координата блока
    //пербор доменных блоков
    Rang ran = null;
    int R = r / k; //размер рангового блока
    int D = R * 2; //размер доменного блока

```

```

int N = height / R; //количество блоков по высоте
int M = width / R; //количество блоков по ширине
int[,] domen = new int[R, R]; //уменьшенный доменный блок
int[,] domenAfin = new int[R, R]; //доменный блок подвергнутый аффинному преобразованию
int[,] domenBig = new int[D, D]; //доменный блок
double minSKO = 10000000;
Rang minRang = new Rang(0, 0, 0, 1, x0, y0, 1, 1, epsilon);
int minX = 0, minY = 0, minAfin = 0;
bool b = false;
int id = 0;
int jd = 0;

while ((id < N - 1) && (b == false)) {
    jd = 0;
    while ((jd < M - 1) && (b == false)) {
        int sum = 0;
        //выделяем доменный блок
        for (int i = 0; i < D; i++)
            for (int j = 0; j < D; j++)
                domenBig[i, j] = pix[R * id + i, R * jd + j];
        //уменьшаем его усреднением
        domen = reduceBlock(domenBig);

        double[] so;
        double s, o, sko;
        List<double> skoMass = new List<double>();
        Color colorDomen, colorRang;

        //вычисляем все СКО
        for (int h = 0; h < 8; h++) {
            sko = 0;
            domenAfin = setAfinnInt(domen, h);
            so = getSO(rang, domenAfin);
            s = so[0];
            o = so[1];
            for (int i = 0; i < R; i++)
                for (int j = 0; j < R; j++) {
                    colorDomen = Color.FromArgb(domenAfin[i, j]);
                    colorRang = Color.FromArgb(rang[i, j]);
                    double per = (s * colorDomen.R + o) - colorRang.R;
                    sko = sko + (per) * (per);
                }
            skoMass.Add(sko); //skoMass[h] = sko;
        }

        //ищем минимальное СКО
        double min = skoMass.Min();

        //сравниваем с коэффициентом компрессии
        if (min < epsilon) {
            b = true;
            int afin = skoMass.IndexOf(min);
            domenAfin = setAfinnInt(domen, afin);
            so = getSO(rang, domenAfin);
            s = so[0];
            o = so[1];
            ran = new Rang(jd * R, id * R, afin, k, x0, y0, s, o, min);
        }
    }
}

```

```

    }
    else {
        if (min < minSKO) {
            int afin = skoMass.IndexOf(min);
            domenAfin = setAfinnInt(domen, afin);
            so = getSO(rang, domenAfin);
            s = so[0];
            o = so[1];
            minSKO = min;
            minRang = new Rang(jd * R, id * R, afin, k, x0, y0, s, o, minSKO);
        }
    }
    jd++;
}
id++;
}
//если для рангового блока не нашли доменного
if (ran == null) rangList.Add(minRang);
else rangList.Add(ran);
}

```

//Выбор подходящего доменного блока для метода эталонного блока

```

public void getDomenBlocTest(int[,] rang, int k, int x0, int y0, double rangSKO, List<List<double>>
domenSKOList)

```

```

{
    //x - начальная координата блока
    //y - начальная координата блока
    //пербор доменных блоков
    Rang ran = null;
    int R = r / k; //размер рангового блока
    int D = R * 2; //размер доменного блока
    int N = height / R; //количество блоков по высоте
    int M = width / R; //количество блоков по ширине
    int[,] domen = new int[R, R]; //уменьшенный доменный блок
    int[,] domenAfin = new int[R, R];
    int[,] domenBig = new int[D, D]; //доменный блок
    double minSKO = 100000000;
    Rang minRang = new Rang(0, 0, 0, 1, x0, y0, 1, 1, epsilon);
    bool b = false;
    int id = 0;
    int jd = 0;
    int number = 0;
    double resSKO = 100000000;

    while ((id < N - 1) && (b == false)) {
        jd = 0;
        while ((jd < M - 1) && (b == false)) {
            double[] so;
            double s, o, sko;
            List<double> skoMass = new List<double>();
            //ищем минимальное СКО
            for (int h = 0; h < 8; h++) {
                sko = Math.Abs(rangSKO - domenSKOList[number][h]); //min|r-d|
                skoMass.Add(sko);
            }
            double min = skoMass.Min();

```

```

        if (min < minSKO) {
            int afin = skoMass.IndexOf(min);
            //выделяем доменный блок
            for (int i = 0; i < D; i++)
                for (int j = 0; j < D; j++)
                    domenBig[i, j] = pix[R * id + i, R * jd + j];
            //уменьшаем его усреднением
            domen = reduceBlock(domenBig);

            domenAfin = setAfinnInt(domen, afin);
            so = getSO(rang, domenAfin);
            s = so[0];
            o = so[1];
            Color colorDomen, colorRang;
            sko = 0;
            for (int i = 0; i < R; i++)
                for (int j = 0; j < R; j++) {
                    colorDomen = Color.FromArgb(domenAfin[i, j]);
                    colorRang = Color.FromArgb(rang[i, j]);
                    double per = (s * colorDomen.R + o) - colorRang.R;
                    sko = sko + (per) * (per);
                }
            resSKO = sko;
            minRang = new Rang(jd * R, id * R, afin, k, x0, y0, s, o, sko);
        }
        number++;
        jd++;
    }
    id++;
}
if (resSKO < epsilon)    rangList.Add(minRang);
else    getDomenBlocFirst(rang, k, x0, y0);
}

```

```

public List<double> getAllSKO(int[,] blockTest, int[,] block)    {
    int size = block.GetLength(0);
    int[,] blockAfin = new int[size, size];
    double[] so;
    double s, o, sko;
    List<double> skoMass = new List<double>();
    Color colorAfin, colorTest, col;
    col = Color.White;
    int c = col.ToArgb();

    //вычисляем все СКО
    for (int h = 0; h < 8; h++)    {
        sko = 0;
        blockAfin = setAfinnInt(block, h);
        so = getSO(blockTest, blockAfin);
        s = so[0];
        o = so[1];
        for (int i = 0; i < size; i++)
            for (int j = 0; j < size; j++)    {
                colorAfin = Color.FromArgb(blockAfin[i, j]);
                colorTest = Color.FromArgb(blockTest[i, j]);
                double per = (s * colorAfin.R + o) - colorTest.R;
                sko = sko + (per) * (per);
            }
    }
}

```

```

    }
    skoMass.Add(sko); //skoMass[h] = sko;
}
return skoMass;
}

```

//Преобразование яркости

```

public int[,] changeBright(int[,] pix, double s, double o)    {
    int n = pix.GetLength(0);
    double x;
    int[,] p = new int[n, n];
    Color color;

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)    {
            color = Color.FromArgb(pix[i, j]); //new Color(pix[i, j]);
            x = s * color.R + o;
            if (x > 255)                x = 255;
            if (x < 0)                  x = 0;
            color = Color.FromArgb((int)x, (int)x, (int)x);
            p[i, j] = color.ToArgb();
        }
    return p;
}

```

//Применение аффинного преобразования

```

public int[,] setAfinnInt(int[,] pix, int k)    {
    int n = pix.GetLength(0);
    int x, y;
    int[,] p = new int[n, n];

    if (k < 4)    {
        if (k == 1)    { //поворот на 90
            for (int i = 0; i < n; i++)
                for (int j = 0; j < n; j++)    {
                    x = n - 1 - i;
                    y = j;
                    p[y, x] = pix[i, j];
                }
        } else if (k == 2)    { //поворот на 180
            int h;
            for (int i = 0; i < n; i++)
                for (int j = 0; j < n; j++)    {
                    x = n - 1 - i;
                    y = j;
                    h = x;
                    x = n - 1 - y;
                    y = h;
                    p[y, x] = pix[i, j];
                }
        } else if (k == 3)    { //поворот на 270
            for (int i = 0; i < n; i++)
                for (int j = 0; j < n; j++)    {
                    x = i;
                    y = n - 1 - j;
                    p[y, x] = pix[i, j];
                }
        }
    }
}

```



```

    }
}
} else {
if (k == 4) { //отражение по вертикали
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            x = n - 1 - j;
            y = i;
            p[y,x] = pix[i,j];
        }
} else if (k == 5) { //отражение по горизонтали
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            x = j;
            y = n - 1 - i;
            p[y,x] = pix[i,j];
        }
}
else if (k == 6) {
    int[,] p2 = new int[n, n];
    //поворот на 90(k=1)
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
        {
            x = n - 1 - i;
            y = j;
            p2[y, x] = pix[i, j];
        }
    //отражение по вертикали (k=4)
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
        {
            x = n - 1 - j;
            y = i;
            p[y, x] = p2[i, j];
        }
} else if (k == 7) {
    int[,] p2 = new int[n, n];
    //поворот на 270 (k=3)
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            x = i;
            y = n - 1 - j;
            p2[y, x] = pix[i, j];
        }
    //отражение по вертикали (k=4)
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            x = n - 1 - j;
            y = i;
            p[y, x] = p2[i, j];
        }
}
}
if (k == 0)
    return pix;
else return p;

```

```
}
```

//Уменьшение блока в 2 раза

```
public int[,] reduceBlock(int[,] blockBig)    {
    int n = blockBig.GetLength(0);
    int[,] block = new int[n / 2, n / 2];
    Color color;// = new Color(domen[i][j]);
    int d = 0, sum;
    for (int i = 0; i < n ; i = i + 2)
        for (int j = 0; j < n; j = j + 2)      {
            sum = 0;
            sum += Color.FromArgb(blockBig[i, j]).R;
            sum += Color.FromArgb(blockBig[i + 1 , j]).R;
            sum += Color.FromArgb(blockBig[i, j + 1]).R;
            sum += Color.FromArgb(blockBig[i + 1, j + 1]).R;
            d = (int)(sum / 4);
            color = Color.FromArgb(d, d, d);
            block[i/2, j/2] = color.ToArgb();
        }
    return block;
}
```

//Получение значений оптимальных яркости и контрастности

```
public double[] getSO(int[,] rang, int[,] domen)  {
    int N = rang.GetLength(0);
    double s = 0, o = 0, D = 0, R = 0, a = 0, b = 0;
    Color colorDomen, colorRang;
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)              {
            colorDomen = Color.FromArgb(domen[i, j]);
            colorRang = Color.FromArgb(rang[i, j]);
            R = R + colorRang.R;
            D = D + colorDomen.R;
        }
    R = R / (N * N);
    D = D / (N * N);
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)              {
            colorDomen = Color.FromArgb(domen[i, j]);
            colorRang = Color.FromArgb(rang[i, j]);
            a = a + (colorDomen.R - D) * (colorRang.R - R);
            b = b + (colorDomen.R - D) * (colorDomen.R - D);
        }
    s = a / b;
    if (b == 0 && a == 0)
        s = 0;
    o = R - s * D;
    return new double[] {s,o};
}
}
```

//Класс декомпрессии

```
using System;
using System.Collections.Generic;
using System.Drawing;
```

```

using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Fract
{
    public class Decompress
    {
        private List<Rang> rangList = new List<Rang>();
        private int r; //размер рангового блока
        private Bitmap bi; //BufferedImage bi;
        private int width; //ширина картинки
        private int height; //высота картинки
        private List<Rang> rangListR = new List<Rang>();
        private List<Rang> rangListG = new List<Rang>();
        private List<Rang> rangListB = new List<Rang>();
        private List<Rang> rangListY = new List<Rang>();
        private List<Rang> rangListI = new List<Rang>();
        private List<Rang> rangListQ = new List<Rang>();

        public Decompress(List<Rang> rangList, Bitmap bi, int r) //Decompress(List<Rang> rangList,
BufferedImage bi, int r) {
            this.rangList = rangList;
            this.r = r;
            this.bi = bi;
            this.width = bi.Width;
            this.height = bi.Height;
        }

        public Decompress(List<Rang> rangListR, List<Rang> rangListG, List<Rang> rangListB, List<Rang>
rangListY, List<Rang> rangListI, List<Rang> rangListQ, Bitmap bi, int r) //Decompress(List<Rang>
rangList, BufferedImage bi, int r) {
            this.rangListR = rangListR;
            this.rangListG = rangListG;
            this.rangListB = rangListB;
            this.rangListY = rangListY;
            this.rangListI = rangListI;
            this.rangListQ = rangListQ;
            this.r = r;
            this.bi = bi;
            this.width = bi.Width;
            this.height = bi.Height;
        }

        public Bitmap decompressImage(int k, string imageColor) {
            if (imageColor.Equals("gray")) return decompressImage(k);
            else if (imageColor.Equals("rgb")) return decompressImageColorRGB(k);
            else if (imageColor.Equals("yiq")) return decompressImageColorYIQ(k);
            else return decompressImage(k);
        }
    }
}

//Алгоритм декомпрессии
public Bitmap decompressImage(int k) {
    int[, ] domen;
    int[, ] domenBig;
    int R, D;

```

```

Color color;
int tk = 0;
for (int g = 0; g < k; g++)      {
    int m = bi.Width;
    int n = bi.Height;
    int[,] pixels = new int[n, m];
    //получаем массив интовых чисел из изображения
    for (int i = 0; i < n; i++)//строки
        for (int j = 0; j < m; j++)//столбцы
            pixels[i, j] = bi.GetPixel(j, i).ToArgb();
    tk = 0;
    foreach (Rang rang in rangList)      {
        R = r / rang.getK();
        D = R * 2;
        domen = new int[R,R];
        domenBig = new int[D,D];
        //выделяем доменный блок
        for (int i = 0; i < D; i++)
            for (int j = 0; j < D; j++)
                domenBig[i, j] = bi.GetPixel(rang.getX() + j, rang.getY() + i).ToArgb();
        int d = 0, sum = 0;
        //и уменьшаем его усреднением
        domen = reduceBlock(domenBig);
        //аффинное преобразование
        domen = setAfinnInt(domen, rang.getAfinn());
        //преобразование яркости
        domen = changeBright(domen, rang.getS(), rang.getO());
        for (int i = 0; i < R; i++)
            for (int j = 0; j < R; j++)      {
                color = Color.FromArgb(domen[i, j]);
                bi.SetPixel(rang.getX0() + j, rang.getY0() + i,color);
            }
        tk++;
    }
    printDecompression(g);
}
return bi;
}

```

//Алгоритм декомпрессии в случае цветного изображения

```

public Bitmap decompressImageColorRGB(int k)      {
    int[,] domen_Red, domenBig_Red, domen_Green, domenBig_Green, domen_Blue, domenBig_Blue;
    int R, D, tk = 0;
    Color color;
    for (int g = 0; g < k; g++)      {
        int m = bi.Width;
        int n = bi.Height;
        int[,] pixels = new int[n, m];
        //получаем массив интовых чисел из изображения
        for (int i = 0; i < n; i++)//строки
            for (int j = 0; j < m; j++)//столбцы
                pixels[i, j] = bi.GetPixel(j, i).ToArgb();//bi.getRGB(j, i);
        tk = 0;
        for (int l = 0; l < rangListR.Count; l++)      {
            R = r / rangListR[l].getK();
            D = R * 2;

```

```

    domen_Red = new int[R, R];
    domenBig_Red = new int[D, D];
    int d = 0, sum = 0;
    //выделяем доменный блок
    for (int i = 0; i < D; i++)
        for (int j = 0; j < D; j++)
            domenBig_Red[i, j] = bi.GetPixel(rangListR[l].getX() + j, rangListR[l].getY() +
i).ToArgb();
    //и уменьшаем его усреднением
    domen_Red = reduceBlockColor(domenBig_Red);
    //аффинное преобразование
    domen_Red = setAfinnInt(domen_Red, rangListR[l].getAfinn());
    //преобразование яркости - получаем компоненту одного цвета
    domen_Red = changeBrightColorRGB(domen_Red, rangListR[l].getS(), rangListR[l].getO(),
"R");
    for (int i = 0; i < R; i++)
        for (int j = 0; j < R; j++)
            {
                color = Color.FromArgb(domen_Red[i, j],
                    bi.GetPixel(rangListR[l].getX0() + j, rangListR[l].getY0() + i).G,
                    bi.GetPixel(rangListR[l].getX0() + j, rangListR[l].getY0() + i).B);
                bi.SetPixel(rangListR[l].getX0() + j, rangListR[l].getY0() + i, color);
            }
        tk++;
    }
    for (int l = 0; l < rangListG.Count; l++)
        {
            R = r / rangListG[l].getK();
            D = R * 2;
            domen_Green = new int[R, R];
            domenBig_Green = new int[D, D];
            int d = 0, sum = 0;
            //выделяем доменный блок
            for (int i = 0; i < D; i++)
                for (int j = 0; j < D; j++)
                    domenBig_Green[i, j] = bi.GetPixel(rangListG[l].getX() + j, rangListG[l].getY() +
i).ToArgb();
            //и уменьшаем его усреднением
            domen_Green = reduceBlockColor(domenBig_Green);
            //аффинное преобразование
            domen_Green = setAfinnInt(domen_Green, rangListG[l].getAfinn());
            //преобразование яркости - получаем компоненту одного цвета
            domen_Green = changeBrightColorRGB(domen_Green, rangListG[l].getS(),
rangListG[l].getO(), "G");
            for (int i = 0; i < R; i++)
                for (int j = 0; j < R; j++)
                    {
                        color = Color.FromArgb(bi.GetPixel(rangListG[l].getX0() + j, rangListG[l].getY0() + i).R,
                            domen_Green[i, j],
                            bi.GetPixel(rangListG[l].getX0() + j, rangListG[l].getY0() + i).B);
                        bi.SetPixel(rangListG[l].getX0() + j, rangListG[l].getY0() + i, color);
                    }
                tk++;
            }
        }
    for (int l = 0; l < rangListB.Count; l++)
        {
            R = r / rangListB[l].getK();
            D = R * 2;
            domen_Blue = new int[R, R];
            domenBig_Blue = new int[D, D];
            int d = 0, sum = 0;

```

```

        //выделяем доменный блок
        for (int i = 0; i < D; i++)
            for (int j = 0; j < D; j++)
                domenBig_Blue[i, j] = bi.GetPixel(rangListB[l].getX() + j, rangListB[l].getY() +
i).ToArgb();
        //и уменьшаем его усреднением
        domen_Blue = reduceBlockColor(domenBig_Blue);
        //аффинное преобразование
        domen_Blue = setAfinnInt(domen_Blue, rangListB[l].getAfinn());
        //преобразование яркости - получаем компоненту одного цвета
        domen_Blue = changeBrightColorRGB(domen_Blue, rangListB[l].getS(), rangListB[l].getO(),
"B");
        for (int i = 0; i < R; i++)
            for (int j = 0; j < R; j++)
                {
                    color = Color.FromArgb(bi.GetPixel(rangListB[l].getX0() + j, rangListB[l].getY0() + i).R,
                    bi.GetPixel(rangListB[l].getX0() + j, rangListB[l].getY0() + i).G,
                    domen_Blue[i, j]);
                    bi.SetPixel(rangListB[l].getX0() + j, rangListB[l].getY0() + i, color);
                }
            tk++;
        }
        printDecompression(g);
    }
    return bi;
}

```

//Изменение яркости для блока цветного изображения

```

public int[,] changeBrightColorRGB(int[,] pix, double s, double o, string imageColor)    {
    int n = pix.GetLength(0);
    double x;
    int[,] p = new int[n, n];
    Color color;
    bool red = false;
    bool green = false;
    bool blue = false;

    if (imageColor.Equals("R"))                red = true;
    else if (imageColor.Equals("G"))            green = true;
    else if (imageColor.Equals("B"))            blue = true;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            {
                color = Color.FromArgb(pix[i, j]);
                if(red) x = s * color.R + o;
                else if(green) x = s * color.G + o;
                else if (blue) x = s * color.B + o;
                else x = s * color.R + o;
                if (x > 255)
                    x = 255;
                if (x < 0)
                    x = 0;
                p[i, j] = (int)x;
            }
        return p;
    }
}

```