Федеральное государственное бюджетное образовательное учреждение высшего образования
Московский авиационный институт
(национальный исследовательский университет) (МАИ)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Тема: Сжатие графических изображений.

Направление: 230100 – Информатика и вычислительная техника.

Работа допущена к защите:

зав. Кафедрой 304

Брехов Олег Михайлович _____ «_____ « 2017 г.

Выполнил:

Студент гр. 3О-409Б

Моцик Дмитрий Юрьевич _____ «_____ « 2017 г.

Научный руководитель:

Доцент, к.т.н.

Чугаев Борис Николаевич _____ «_____ « 2017 г.

Москва – 2017

Содержание:

Введение

Постановка задачи.

Проанализировать основные алгоритмы сжатия графических растровых изображений, область применения, модификации, принцип и технологию сжатия. Также проанализировать форматы хранения цифровых изображений, технологию хранения данных, область применения и ориентированность. Реализовать и применить алгоритм сжатия с чтением и хранением данных, с возможностью оптимизации(увеличение степени сжатия и уменьшение объема изображения) в соответствии с выбранным форматом и технологией и структурой формата.

Представление изображения на экране.

Во всех дисплеях графическое цифровое изображение представляет собой двумерный массив из дискретных элементов, которые называются «пикселями» (англ. pixel, сокращение от **pic**ture**'s el**ement – элемент изображений). Пиксель представляет собой неделимый прямоугольный или круглой формы элемент, характеризующийся определенным цветом. На экране, объекты получаются путем установки цветов группы пикселей. Пиксельное представление отличается зубчатыми краями, «лесенками» вместо гладкой линии. Это очень плохо сказывается на изображении, при плохом разрешении экрана. Рисунок 1 демонстрирует, как представляется изображение через пиксели и что как оно портиться при масштабировании или плохом разрешении экрана.

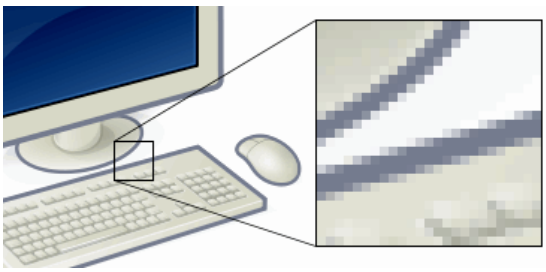

Рис.1. Пример изображения из пикселей.

Современные экраны позволяют отображать огромное число пикселей, что очень положительно сказывается на изображении. Сейчас обычное разрешение мониторов 1280×720 пикселей – называется «HD», а 1920×1080 – «Full HD». А разрешение современных телевизоров намного больше. Количество пикселей, которое может отображать система, ограничивается объемом видеопамяти и технологическими особенностями мониторов и видео-

адаптерами. В видеопамяти каждому пикселю соответствует ячейка, размер который от 1 до 32 бит.

Форматы файлов графических изображений можно разделить на два типа: растровый и векторный. Растровые изображения состоят из матрицы пикселей у которой определен размер, фиксированные высота и ширина. Каждый пиксель имеет свой цвет. При масштабировании на изображении появляются заметные «ступени» из пикселей, что заметно портит изображение. Если в устройстве отображения пиксели расположены достаточно плотно, человеческому глазу трудно заметить структуру массива точек. Растровые изображения могут быть созданы с помощью разнообразных устройств ввода, таких как цифровые камеры, сканеры, и т.д. Они также могут быть синтезированы из произвольных данных без изображения, таких как математические функции или трехмерные геометрические модели. Последнее является основной областью компьютерной графики. Растровое изображение часто употребляют, как синоним «цифровому» изображению или «точечный графический формат».

Векторные изображения представляются с помощью последовательности команд рисования. Применение векторных форматов не ограничивается устройствами выводов. Компьютеры, мониторы и принтеры оснащены ПО, которое преобразует векторный формат в пикселы. В отличие от растровых изображений, векторные при масштабировании не теряют качества изображения. Но у векторной графики два основных недостатка:

1) Они не подходят для фотографий или живописи. Например, для отображения простого портрета потребовалось бы огромное количество команд рисования.
2) Отображение сложных изображений требует больших временных затрат. В большинстве систем отображения, каждый векторный объект должен преобразовываться в пиксельное изображение.

На рисунке 2 представлено изображение в растровом (слева) и векторном(справа) представлении.

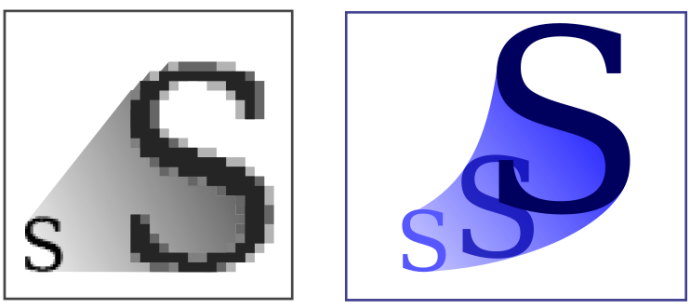

Рис.2. Растровое и векторное представление изображения.

Наиболее главное преимущество растровых изображений стоит в их качестве. Другим недостатком растровых изображений является то, что они зависимы от размера и непригодны для обширного редактирования. При работе с векторными проще добавлять и модифицировать часть данных. Растровые изображения проблематично даже масштабировать. Но самый главный недостаток, это объем данных, который нужен для представления растровых изображений. Размер изображения в байтах(без учета избыточных) составляет:

$$\frac{\text{ширина}*\text{высота}*\text{число битов на пиксель} + 7}{8}.$$

Тогда для изображения 1920×1080 с 24 битами на пиксель получится 62208001 байтов в памяти. Для уменьшения объема данных, они хранятся в сжатом виде, одним их алгоритмов сжатия.

Хоть векторная графика обладает аппаратной независимостью, легкостью редактирования и использование памяти, временные затраты на представление даже простых снимков и представление их в пиксельное отображение очень велики. Между тем, у векторной графики своя область применения, у растровой - другая. Поэтому трудно отдать предпочтение одному из этих форматов однозначно. Современные программы, для работы с графическими изображениями используют растровую и векторную графику в тандеме[1].

Численное представление цвета.

Каждому пикселю соответствует свой цвет и численное значение, представляющее этот цвет. Система представления цветов называется *моделью представления цвета*. Большинство мониторов, для отображения цвета пикселя используют три люминофора – красный, зеленый, синий. Изменяя яркость каждого из них можно добиться различных оттенков. Минимальная яркость трех компонентов – черный цвет, максимальная – белый цвет. Общепринятая модель в ПО считается RGB(с англ. **R**ed – красный, **G**reen – зеленый, **B**lue – синий). Диапазон цветов в цветовой модели называют цветовым пространством, где каждый представляется в виде 3 действительных целых чисел. Первое число обозначает красную компоненту, второе - зеленую, 3 – синюю. Диапазон значений для компонентов называется *частой дискретизации*, которая равна количеству битов, используемых для представления цветового компонента. Чаще всего она равна 8 битам. Также встречаются значения частоты, равные 1,2,4,12 и 16 бит. Числовые значения компонентов

могут находится от 0 до $2^{\text{частота\_дискретизации}} - 1$. Так система может распознавать 256 различных оттенков каждого основного цвета.

Полутоновая шкала – еще одна модель представления цвета. Цвета в этой модели могут отображаться только в оттенках серого, без палитры. Оттенки серого могут представлены одним из компонентов в диапазоне от 0(черный) до $2^{\text{частота\_дискретизации}} - 1$(белый).

Также используется трехкомпонентная *YCbCr* модель представления цвета, которая почти всегда используется в формате JPEG. Компонент Y (яркость) представляет яркость изображения, Cb(синева) – задает синеву изображения и Cr(краснота) – задает красноту изображения. Соотношение между цветовыми моделями YCbCr и RGB описываются системами уравнений 1 и 2[1]:

$$\begin{cases} Y = 0.299R + 0.587G + 0.114B \\ Cb = -0.1687R - 0.3313G + 0.5B + 2^{\frac{\text{точность дискретизации}}{2}} \\ Cr = 0.5R - 0.4187G - 0.0813B + 2^{\frac{\text{точность дискретизации}}{2}} \end{cases} \quad (1)$$

$$\begin{cases} \mathrm{R} = \mathrm{Y} + 1.402\mathrm{Cr} \\ \mathrm{G} = \mathrm{Y} - 0.34414\left(\mathrm{Cb} - 2^{\frac{\text{точность дискретизации}}{2}}\right) - \\ \qquad 0.71414(\mathrm{Cr} - 2^{\frac{\text{точность дискретизации}}{2}}) \\ \mathrm{B} = \mathrm{Y} + 1.722(\mathrm{Cb} - 2^{\frac{\text{точность дискретизации}}{2}}) \end{cases} \quad (2)$$

Классы изображений.

Для того, чтобы корректно оценивать алгоритмы, область применения каждого алгоритма, изображения можно разделить на 4 основных класса. Под понятием класса, будет сопоставима совокупность определенных изображений, применение алгоритмов сжатия к которым даст наилучшую степень сжатия, одинаково ко всем изображениям данного класса. Например, алгоритм А для одного класса изображений даст наилучшую степень сжатия, для другого класса среднюю, а для 3 наихудшую. Наиболее распространенные классы изображений:

1. **Изображения с небольшим количеством цветов(4 - 16) и большими областями, заполненные одним цветом**. Плавные переходы цветов отсутствуют. Пример: Деловая графика – диаграммы, графики и т.д.
2. **Изображения с плавным переходом цветов, построенные на ЭВМ.** Пример: графики презентаций, результаты из различных САПР.

3. **Фотореалистичные изображения.** Пример: отсканированные фотографии, фотографии
4. **Фотореалистичные изображения с наложением деловой графики.** Пример: Реклама.

Также существуют отдельные узкие классы, которые могут содержать в себе совсем специфичные изображения, например рентгеновские снимки, геоинформационные снимки. Как итог, нельзя применять алгоритм сжатия, не обозначив класс изображения, на который направлен алгоритм [2].

Глава 1. Основные алгоритмы сжатия изображений.

Требования к алгоритмам компрессии.

Для того, чтобы сравнивать алгоритмы компрессии, существует список требований к алгоритмам сжатия. В соответствии с ним, выбирается оптимальный алгоритм под формат и задачу. Алгоритмы компрессии должны обладать:

- **Высокая степень сжатия.** Но не для всех изображений нужна и высокая степень сжатия. Некоторые методы дают высокое соотношение качества при высоких степенях, но проигрывают для низких.
- **Высокое качество изображения.** Методы сжатия должны поддерживать изображения в высоком качестве. Это противоречит предыдущему, т.к. высокая степень сжатия влияет на качество изображения.
- **Высокая скорость компрессии.** Один из критериев, по которому стоит выбирать алгоритм, но стоит понимать, что при маленьком времени компрессии алгоритм может давать плохую степень сжатия, т.к. не успевает «анализировать» изображение.
- **Высокая скорость декомпрессии.** Аналогично с предыдущим пунктом.
- **Масштабирование изображений.** Этот критерий подразумевает легкость изменения размеров изображения. С помощью некоторых алгоритмов можно легко масштабировать изображение, другие же могут стать причиной появления пиксельных «лесенок»
- **Возможность показать огрубленное изображение (низкого разрешения),** использовав только начало файла. Данная возможность актуальна для изображений в сетях и различных сетевых приложениях, где перекачивание изображений может занять достаточно большое время и желательно, получив начало файла, корректно показать его превью. Но это может сказаться в худшую сторону на степени компрессии.
- **Устойчивость к ошибкам.** Это требование нужно, когда нет возможности повторно попросить данные с сервера, а информация должна отображаться корректно. Избыточность данных при их передаче, может плохо сказаться на степени архивации.
- **Учет специфики изображения.** Более высокая степень сжатия для класса изображений, которые статистически чаще будут применяться в нашем приложении. В предыдущих подразделах это требование уже обсуждалось.

- **Редактируемость.** Многие алгоритмы с потерей информации могут существенно испортить изображение за несколько итераций редактирования.
- **Небольшая стоимость аппаратной реализации.**
- **Эффективность программной реализации.** Данные требования к алгоритму реально предъявляют не только производители игровых приставок, но и производители многих информационных систем.

Критерии сравнения алгоритмов сжатия.

Итак, степень и скорость компрессии будет зависеть от того, к какому классу изображения будет сопоставлен алгоритм, также скорость компрессии будет зависеть от платформы, на которой реализован алгоритм. Поэтому нельзя утверждать об оптимальном алгоритме на все классы изображений. Также стоит учитывать технологию алгоритма, например, может он быть распараллелен или нет.

«Таким образом, невозможно составить универсальное сравнительное описание известных алгоритмов. Это можно сделать только для типовых классов приложений при условии использования типовых алгоритмов на типовых платформах. Однако такие данные необычайно быстро устаревают

Так, например, еще в 1994 г., интерес к показу огрубленного изображения, используя только начало файла (требование 6), был чисто абстрактным. Реально эта возможность практически нигде не требовалась, и класс приложений, использующих данную технологию, был крайне невелик. Со взрывным распространением Интернета, который характеризуется передачей изображений по сравнительно медленным каналам связи, использование Interlaced GIF (алгоритм LZW) и Progressive JPEG (вариант алгоритма JPEG), реализующих эту возможность, резко возросло. То, что новый алгоритм (например, wavelet) поддерживает такую возможность, существеннейший плюс для него сегодня.» - [1].

Далее, со всеми оговорками которые были выше, можно составить несколько критериев по которым будут сравниваться алгоритмы.

1. **Худшая, средняя и лучшая степень сжатия.** Это доля, на которую будет возрастать изображение, если исходные данные будут наихудшими, средняя – для класса изображений, на который направлен алгоритм, лучшая для идеального изображения.
2. **Класс изображений,** на который ориентирован алгоритм. Здесь указывается, почему для других классов алгоритм показывает худший результат.

3. **Симметричность.** Отношение полученных данных при компрессии к данным декомпрессии. Характеризует ресурсоемкость процессов кодирования и декодирования. Наиболее важной считается симметричность по времени.
4. **С потерями качества алгоритм или нет?** Если нет, за счет чего изменяется степень сжатия.
5. **Характерные особенности алгоритма и изображений, к которым его следует применять.** Здесь стоит указывать специфичные свойства алгоритма сжатия, его особенности.

Каждый из алгоритмов сжатия таких, как RLE, LZW, Huffman имеют свои модификации. В зависимости от них, будут меняться характеристики алгоритмов. Даже незначительное изменение, такое как размер окна, число длин могу играть существенную роль на степень сжатия и время компрессии, да и в общем, эффективность алгоритма в целом. Параметры алгоритмов устанавливаются исходя из требований к экономии памяти, времени архивации. Поэтому, для конкретной реализации под входные данные алгоритмы одного семейства могут отличаться, но в целом, картина не изменится.

Алгоритмы сжатия без потерь.

Алгоритм RLE.

RLE(с англ. **R**un - **l**ength **e**ncoding) – кодирование длин серий или кодирование повторов. Данный метод сжатия основан на замене повторяющихся серий символов, которые находятся в исходном изображении, на один символ и число его повторов. Серией символов называется последовательность, состоящая из нескольких одинаковых символов. При кодировании, серия символов заменяется строкой, содержащей сам повторяющийся символ и число его повторов - <число повторений, значение >. Этот алгоритм очень прост в реализации и один из самых старых. В идеале, картинку «черный квадрат» сожмет до двух байт. В одном будет количество черных пикселей, а в другом черный цвет. А строка вида «ПППППППРРРРИВВВВВЕЕЕЕТТТ» будет закодирована, как «7П3Р1И5В4Е3Т». Есть два варианта реализации этого алгоритма.

Чтобы как-то представить результат в понятном компьютеру виде. Для этого, в потоке данных должны как-то отделять одиночные байты от кодируемых цепочек. Поскольку весь диапазон значений байта используется данными, то просто так выделить какие-либо диапазоны значений удастся.

Есть как минимум два выхода из этой ситуации:

1. В качестве индикатора сжатой цепочки выделить одно значение байта, а в случае коллизии с реальными данными экранировать их. Например, если использовать в «служебных» целях значение 255, то при встрече этого значения во входных данных мы вынуждены будем писать «255, 255» и после индикатора использовать максимум 254.

2. Структурировать закодированные данные, указывая количество не только для повторяемых, но и последующих далее одиночных элементов. Тогда мы будем заранее знать, где какие данные.

Первый способ не эффективен по сравнению со вторым, поэтому, пожалуй, прибегнем ко второму.

Существует два варианта алгоритма RLE для второго подхода. Рассмотрим первый:

В этом варианте при кодировании серия признаком счетчика будут служить две единицы в старших битах закодированного файла. Оставшиеся 6 бит занимает счетчик, который принимает значение от 1 до 64. Дальше следует значение, которое стоит повторить. Пример представлен на рисунке 3.

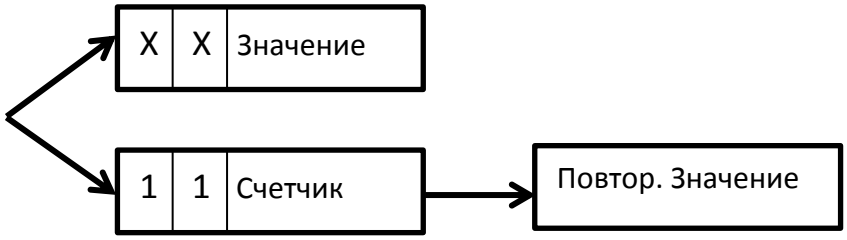

Рисунок.3. Пример представления счетчика в байте в одной из реализации RLE.

Данный алгоритм рассчитан на деловую графику - это изображения, в которых есть большие области залитые одним цветом или большое количество областей с повторяющимся цветом. Данный алгоритм плохо применим к цветным фотографиям, где значение всех пикселе больше двоичного 11000000. Второй вариант алгоритма:

Второй вариант алгоритма имеет большую максимальную степень сжатия и меньше увеличивает в размерах исходный файл. Признаком счетчика, в отличие от первого варианта, является одна единица в старшем разряде байта(рисунок 4). Тем самый, длина счетчика увеличивается до 7 бит, и длина увеличивается до 128. Средние показатели степени компрессии данного алгоритма находятся на уровне показателей первого варианта.

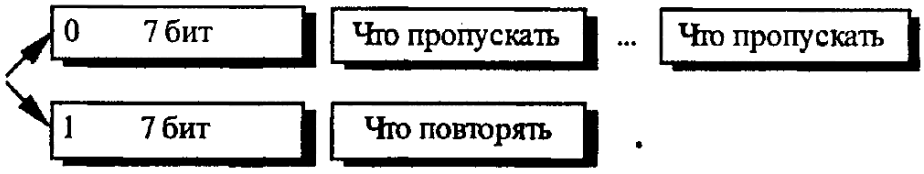

Рисунок.4. Пример представления счетчика в байте во второй реализации RLE.

Пример:

Последовательность {1, 1, 1, 1, 1, 1, 4, 4, 4, 5, 5, 6, 7, 8, 100, 100, 100, 100, 100, 153, 153, 153, 153, 250, 250, 250, 250}. Представление данной последовательности в hex формате будет выглядеть:
01 01 01 01 01 01 04 04 04 05 05 06 07 08 64 64 64 64 64 99 99 99 99 FA FA FA FA.

Выделим в некоторых байтах один бит под тип последовательности: 0 — одиночные элементы, 1 — одинаковые. В оставшихся 7 битах мы будут хранить длины последовательностей, т.е. максимальная длина кодируемой последовательности — 127 байт.

В выходной поток будут помещаться сначала длина последовательности, а далее символ, который должен повторяться. Не может быть последовательностей с нулевой длиной, поэтому можно увеличить максимальную длину до 128 байт, отнимая от длины единицу при кодировании и прибавляя при декодировании. Таким образом, можно кодировать длины от 1 до 128 вместо длин от 0 до 127.

Не бывает последовательностей одинаковых элементов единичной длины. Поэтому, от значения длины таких последовательностей при кодировании отнимается ещё й, увеличив тем самым их максимальную длину до 129 (максимальная длина цепочки одиночных элементов по-прежнему равна 128). Т.е. цепочки одинаковых элементов у нас могут иметь длину от 2 до 129.

Закодируем наши данные в понятном для компьютера виде. В байты данные будут записываться как [T|L], где T — тип последовательности, а L — длина. Сразу учитывается, что длины мы записываем в изменённом виде: при T=0 прибавляем к L единицу, при T=1 — двойку:

[1|4], 1, **[1|1]**, 4**,[1|0]**, 5, **[0|2],**6, 7,8**, [1|3]**, 100, **[1|2]**, 153, **[1|2]**, 250

Декодируем результат:

[1|4]. T=1, значит следующий байт будет повторяться L+2 (4+2) раз: 1, 1, 1, 1, 1, 1.

[1|1]. T=1, значит следующий байт будет повторяться L+2 (1+2) раз:4, 4, 4.

[1|0]. T=1, повторяем следующий байт 0+2 раз: 5,5.

[0|2]. T=0, читаем следующих L +1(2+1) байта: 6,7,8.

[1|3]. T=1, значит следующий байт будет повторяться L+2 (3+2) раз:100, 100, 100, 100, 100.

[1|2]. T=1, значит следующий байт будет повторяться L+2 (2+2) раз:153, 153, 153, 153.

[1|2]. T=1, значит следующий байт будет повторяться L+2 (2+2) раз:250, 250, 250, 250.

Теперь упаковываем каждое значение в байт:

[1|4] = 10000100 = 0x84

[1|1] = 10000001 = 0x81

[1|0] = 10000000 = 0x80

[0|2] = 00000010 = 0x02

[1|3] = 10000011 = 0x83

[1|2] = 10000010 = 0x82

[1|2] = 10000010 = 0x82

Получаем: 84 01 81 04 80 05 02 06 07 08 83 64 82 99 82 FA. Вместо 27 байтов, получилось 16.

Эффективность алгоритма зависит не только от алгоритма, но и от способа его реализации. Поэтому, для разных данных можно разрабатывать разные вариации кодирования и представления закодированных данных. Например, при кодировании изображений можно сделать цепочки переменной длины: выделить один бит под индикацию длинной цепочки, и если он выставлен в единицу, то хранить длину и в следующем байте тоже. Так мы жертвуем длиной коротких цепочек (65 элементов вместо 129), но зато даём возможность всего тремя байтами закодировать цепочки длиною до 16385 элементов (214 + 2)!

Также можно добиться эффективности путем игнорирования цепочек из 2 одинаковых байт. Например: «ППОО» будет закодирована как [1,2]П[1,2]О. Разницы в архивации нет.

Характеристики алгоритма RLE	
Степени сжатия (лучшая, средняя, худшая):	Первый вариант: 32, 2, 0.5. Второй вариант: 64, 3, 128/129
Класс изображений:	Ориентирован алгоритм на изображения с небольшим количеством цветов: деловую и научную графику.
Симметричность:	~1
Характерные особенности:	Он не требует дополнительной памяти при работе, и быстро выполняется. Алгоритм применяется в форматах PCX, TIFF, BMP. Интересная особенность группового кодирования в PCX заключается в том, что степень архивации для некоторых изображений может быть существенно повышена всего лишь за счет изменения порядка цветов в палитре изображения.

Таблица 1. Характеристики алгоритма RLE.

Алгоритмы семейства LZ*.

Это алгоритмы сжатия без потерь, которые были основаны на опубликованных статьях математиков Абрахама Лемпела(Abraham **L**empel) и Якоба Зива(Jacob **Z**iv) в течение 1970хх и получили свое название из-за фамилий создателей. На базе этих статей были разработаны алгоритмы LZ77, LZ78, LZW и многие другие. Все алгоритмы семейства LZ относятся к словарным методам. В

LZ77:

Это один из методов, который опубликовали в 77г. Является более сложным обобщением алгоритма RLE. Он состоит из двух принципов: скользящего окна(словаря) и повторяющихся последовательностей.

Принцип скользящего окна заключается в том, что существует некий словарь, который известен кодировщику или он заполняется динамически(информация на каждом шаге по битно заносится в окно определенного размера). Его можно представить, как буфер, который способен запоминать входящую информацию и предоставлять к ней доступ. Затем ранее встреченная информация (информация которая присутствует в словаре) кодируется и замещается ссылками на ее первое вхождение. Поэтому данный метод называют алгоритмом сжатия с использованием скользящего окна. Таким образом, вместо повторяющихся последовательностей их количества, как в методе RLE, LZ77 будет создавать ссылки повторяющихся данных на их значения в скользящем окне. Если словарь определен заранее, он называется статическим, если нет, то динамическим. Если словарь статический, он имеет заданный объем, обычно не больше 32 килобайт.

Повторяющая информация, далее совпадение, будет кодироваться так:

1. Смещение или дистанция в окне
2. Длина совпадений
3. Повторяющееся значение.

Это своего рода команда копирования значения из окна(словаря) начиная с позиции первого аргумента, количества символов указанных во втором аргументе и затем новое значение, хранящееся в 3 аргументе. Этот алгоритм эффективен на больших объемах информации, когда количество повторяющихся символов больше 3-х (в некоторых спецификациях запрещается кодировать меньше 3-х повторяющихся символов чтобы не допустить увеличения объема файла). Так же, если в команде копирования находится длина 10, а аргументов в словаре меньше, то начиная с последнего символа, будут копироваться все совпадающие символы, которые стоят перед ним.

Рассмотрим пример алгоритма LZ77 на строке «рапапапапампампам». Без использования буфера. На первом шаге весь текст помещается словарь. Первый шаг. Берем символ «р». Он встречается первый раз, поэтому его код 00Р. Берем символ «а», аналогично его код ООА, символ «п» будет иметь код 00П. Теперь берем символ «а», он уже встречался, запоминаем его позицию. Берем символ «п», последовательность «ап» тоже встречалась, берем следующий символ «а». Последовательности «апа» еще не было, поэтому кодируем символ «а», как 12А. И так далее. Последний символ не ищется в словаре.

Текущий словарь	**Позиция**	**Смещение**	**Символ**	**Код**
рапапапапампампам	0	0	р	<0,0,Р>
р**а**папапапампампам	0	0	а	<0,0,А>
ра**п**апапапампампам	0	0	п	<0,0,П>
р*ап*ап**а**папампампам	1	2	а	<1,2,А>
ра*пап*апап**а**мпампам	2	4	м	<2,4,М>
рапапапа*пам*пампа**м**	8	3	п	<8,3,П>
р*а*папапапампампа**м**	1	1	м	<1,1,М>

Таблица 2. Пример алгоритма LZ77.

В данном примере алгоритм LZ77 увеличивает размер строки, т.к. алгоритм мало эффективен на небольших объемах данных. Одно из достоинств алгоритма в том, что не нужно хранить таблицу с данными. Достаточно только длины словаря. Для больших объемов данных используют буфер, он должен быть меньше, чем словарь. Данные помещаются в буфер, затем ищется совпадение в словаре. Затем данные кодируются и помещаются в выходную последовательность. Словарь сдвигают на N бит смещения и помещаются данные в конец. Затем сдвигают буфер. Недостатки алгоритма:

1. Невозможность кодировать подстрок, которые находятся на каком то расстоянии друг от друга. Для устранения этого недостатка, компрессор должен повторно использовать любой другой алгоритм сжатия без потерь, для кодирования. Чаще всего используют алгоритм Хаффмана или RLE.
2. Длина подстроки, которую надо закодировать ограничена буфером. Этот недостаток встречается в больших объемах данных, решения недостатка такое же, как и первого.
3. Малая эффективность при кодировании маленького объема данных.

Характеристики алгоритма LZ77:

Характеристики алгоритма LZ77	
Степени сжатия (лучшая, средняя, худшая):	Первый вариант: 4, 1, 0. Определяются объемами данных. Чем их больше, тем большее и степень
Класс изображений:	Ориентирован алгоритм на изображения с большими группами повторяющихся цветов: Д 2. Изображения с плавным переходом цветов, построенные на ЭВМ
Симметричность:	~1.
Характерные особенности:	обычно медленно сжимают избыточные данные. Высокоая скорость декомпрессии. Из-за высокой скорости разжатия идеально подходят для создания дистрибутивов программного обеспечения.

Таблица 3. Характеристики алгоритма LZ77.

LZ78:

Данный алгоритм был опубликован в 1978 году, благодаря чему получил свое название.

В отличии от LZ77 не использует принцип скользящего окна и в словарь помещаются не все строки, которые совпадают(словарь определяется динамически). Помещаются только те строки, которые имеют самую высокую вероятность, что встретятся в будущем. Из – за этой особенности в словаре не бывает одинаковых фраз, это отличает от LZ77.

Алгоритм считывает символы сообщения до тех пор, пока накапливаемая подстрока входит целиком в одну из фраз словаря. Как только эта строка перестанет соответствовать хотя бы одной фразе словаря, алгоритм генерирует код, состоящий из индекса строки в словаре, которая до последнего введенного символа содержала входную строку, и символа, нарушившего совпадение. Затем в словарь добавляется введенная подстрока. Если словарь уже заполнен, то из него предварительно удаляют менее всех используемую в сравнениях фразу. Если в конце алгоритма мы не находим символ, нарушивший совпадения, то тогда мы выдаем код в виде (индекс строки в словаре без последнего символа, последний символ).

Кодировщик порождает только последовательность кодов фраз. Каждый код состоит из следующих двух величин:

1. Индекс родительской фразы или префикса
2. Символ

В начале алгоритма, словарь пустой. Формально нет никаких наложений на объем словаря, но в программной реализации его длина фиксирована. И по достижении этой отметки, он обнуляется. Также не требуется и буфер.

Рассмотрим пример на строке «тромборомбомбом»:

Индекс	**Словарь**	**Считываемая строка**	**Позиция**	**Символ**	**Код**
0		т	0	т	<0,Т>
1	**т**	р	0	р	<0,Р>
2	т,**р**	о	0	о	<0,О>
3	т,р,**о**	м	0	м	<0,М>
4	т,р,о,**м**	б	0	б	<0,Б>
5	т,р,о,м,**б**	о	3	р	<3,Р>
6	т,р,о,м,б ,**ор**	м	3	м	<3,М>
7	т,р,о,м,б ,ор, **ом**	о	5	о	<5,О>
8	т,р,о,м,б ,ор, ом, **бо**	б	3	б	<3,Б>
9	т,р,о,м,б ,ор, ом, бо, **мб**	м	3	м	<3,М>

Таблица 4. Пример алгоритма LZ77.

Как и алгоритм LZ77, LZ78 малоэффективен на малых объемах данных. И может привести к увеличению данных. НА последнем шаге, значение кода совпало из – за того, что это конец последовательности. Если бы за ним шел символ, то он был бы с индексом 7. Как и метод LZ77, 78 обладает высокой скоростью декомпрессии, так еще и опережает своего потомка по этой скорости, и на больших объемах данных позволяет сильнее сжать данные, т.к. в выходной последовательности код состоит из 2-х значений, а не из трех. Но проигрывает по времени в сжатии. Поэтому LZ77 распространен больше. Недостатки такие же, как и у LZ77. Но в 78 не используется буфер, и нет ограничения на длину подстроки словаря, поэтому он выигрывает в объемном ресурсе.

Характеристики алгоритма LZ78:

Характеристики алгоритма LZ78	
Степени сжатия (лучшая, средняя, худшая):	Первый вариант: 3, 2, 0. Определяются объемами данных. Чем их больше, тем большее и степень
Класс изображений:	Ориентирован алгоритм на изображения с большими группами повторяющихся цветов и однородными данными: Деловая графика.
Симметричность:	~1.
Характерные особенности:	из-за относительно небольшой степени сжатия и невысокой скорости декодирования уступают по распространенности алгоритмам семейства LZ77.

Таблица 3. Характеристики алгоритма LZ78.

LZW:

Свое название алгоритм получил по первым буквам фамилий его разработчиков и еще одного математика с фамилией Вельч(Welch). Последним и был опубликован. Сжатие в нем происходит уже за счет одинаковых цепочек байтов. Этот метод скорее ответвление от алгоритма LZ78 и его улучшенная реализация.

Он был разработан с целью быстрой реализации, но не оптимален, т.к. не проверяет никак входные данные. Данный метод при кодировании составляет динамически таблицу преобразования строк, так что определенным последовательностям символов ставятся в соответствие группы бит фиксированной длины. Более подробно процесс сжатия выглядит вот так: последовательно считывается входной поток, символов за символом, и на каждом шаге осуществляется проверка, есть ли в созданной таблице строк текущая строка. Если такая строка существует, то считывается следующий символ, а если строка не существует, то в выходной поток заносится символ предыдущей найденной строки, строка заносится в таблицу, а поиск начинается снова. Новые строки формируют таблицу последовательно, т. е. можно считать индекс строки ее кодом.

Алгоритму декодирования на входе требуется только закодированный текст, поскольку он может воссоздать соответствующую таблицу преобразования непосредственно по закодированному тексту. Алгоритм генерирует однозначно декодируемый код за счет того, что каждый раз, когда генерируется новый код, новая строка добавляется в таблицу строк. На каждом этапе LZW есть проверка, является ли строка уже известной, и, если так, выводит существующий код без генерации нового. Таким образом, каждая строка будет храниться в единственном экземпляре и иметь свой уникальный номер.

Следовательно, при дешифровании при получении нового кода генерируется новая строка, а при получении уже известного, строка извлекается из словаря.

В данном алгоритме для изображений, выгодно сжимать цепочки размером 2 байта. Пример работы этого алгоритма будет рассмотрен ниже. Особенностью LZW(сравнение относительно других алгоритмов сжатия, т.к. данное свойство характерно для всех методов семейства LZW) считается то, что для декомпрессии не надо сохранять таблицу строк в файл для распаковки. Алгоритм построен так, что таблица кодов восстанавливаема из потока кодов. Иллюстрация алгоритма в рисунок 4, где $C_n \ldots C_{n+9}$ – входная последовательность символов.

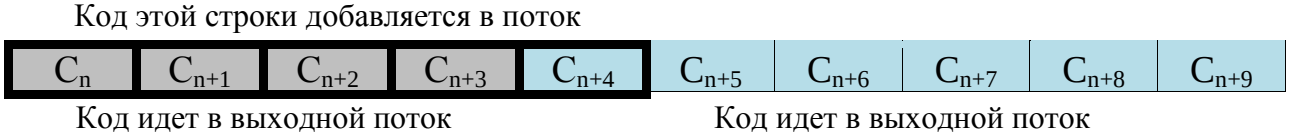

Рис.4

Рассмотрим пример на строке «YESORNOORYESORNO»:

В качестве примера для простоты используются заглавные буквы английского алфавита. В алгоритме LZW используется маркер начала и маркер конца последовательности, например * (он может быть различный в разных программах и системах, использующие данный алгоритм, чаще всего *, \0, #).

Итак, на первом этапе инициализируем входной словарь. Так как мы выбрали только заглавные буквы и *, то будем иметь 27 символов: 26 букв от A – Z и *. Для кодирования каждого символа нужна группа бит. Для данного примера достаточно $2^4 < 27 < 2^5$ – 5 бит. В дальнейшем, группа бит будет расширяться т.к. 5 бит хватит только до 32 символов. Первый этап закончен, исходный словарь будет выглядеть так:

*****	00000	**H**	01000	**P**	10000	**X**	11000
A	00001	**I**	01001	**Q**	10001	**Y**	11001
B	00010	**J**	01010	**R**	10010	**Z**	11010
C	00011	**K**	01011	**S**	10011		
D	00100	**L**	01100	**T**	10100		
E	00101	**M**	01101	**U**	10101		
F	00110	**N**	01110	**V**	10110		
G	00111	**O**	01111	**W**	10111		

Таблица 4. Начальный словарь

Используя данный словарь, перейдем к кодированию. Стоит отметить, для изображения, первоначальный словари будет зависеть от частоты дискретизации. Например, если изображение имеет стандартную частоту 8 бит на пиксель, то группа бит будет состоять из 8 бит и в начальном словаре бу-

дут проинициализированы значения от 0 до 256.Итак, закодированная последовательность представлена ниже:

Входной символ	**Код на выходе**	**Новая запись в словарь**	**Примечание.**
Y	11001	-	Символ есть в словаре? Да. На выход подаем его код. Если первый символ переходим к шагу два.
E	00101	27(11011) - **YE**	Символ есть в словаре? Да. На выход подаем его код. Последовательность Cn, Cn-1 добавляем в словарь. Повторяем данный шаг, пока не встретим символ конца.
S	10011	28(11100) - **ES**	
O	01111	29(11101) - **SO**	
R	10010	30(11110) - **OR**	
N	01110	31(11111) - **RN**	Далее увеличиваем нашу группу на 1 бит, т.к. 5 бит уже не хватает
O	001111	32(100000) - **NO**	
OR	011110	33(100001) - **OO**	
YE	011011	34(100010) - **ORY**	
SO	011101	35(100011) - **YES**	
RN	011111	36(100100) **SOR**	
O	001111	37(100101) - **RNO**	
*	000000	38(100110) **NO***	Конец кодирования.

Таблица 5.Пример кодирования строки алгоритмом LZW.

Как итог, первоначальный текст весил – 5 бит * 16 символов = 80 бит, после кодирования – 67 бит. Если бы последовательность была больше, то алгоритм показал бы более высокую степень. Т.к. количество повторяющихся последовательностей было бы еще больше и размер группы рос. Как и все алгоритмы семейства LZ*, LZW тоже ориентирован на большие объемы данных.

Когда алгоритм LZW появился, он произвел колоссальное впечатление на публику и алгоритмы сжатия того времени. Он давал лучшую степень сжатия, чем его предшественники. Он начал широко использоваться в различном ПО. Характеристики алгоритма представлены в таблице 6.

Характеристики алгоритма LZW	
Степени сжатия (лучшая, средняя, худшая):	~1000 достигается на больших изображениях по объему, 4, 5/7.
Класс изображений:	Ориентирован на изображения, которые построены на ЭВМ или различных САПР. Желательно 8 битные, сжимает за счет одинаковых подцепочек в потоке.
Симметричность:	Симметричен по времени, если реализован оптимальный поиск строки в таблице.
Характерные особенности:	Редко встречаются ситуации, когда алгоритм увеличивает изображение, в отличие от своего предшественника.

Таблица 6. Характеристики алгоритма LZW

Алгоритм Хаффмана.

Это наиболее старый, но эффективный метод кодирования, который был разработан в 60-хх годах, но используется до сих пор. Этот метод основан на кодировании информации с помощью кодов переменной длины. Был разработан аспирантом Массачусетского технологического института Дэвидом Хаффманом. Обычно он не используется в чистом виде для сжатия данных, а используется как дополнение к основному алгоритму. Метод, использующийся в алгоритме Хаффмана заключается в том, что символам, ставятся в соответствие битовые коды переменной длины так, что наиболее часто встречающимся символам ставятся коды короткой длины, реже встречающимся – наоборот. Все коды обладают свойством префиксности(ни один код не является префиксом другого). Это позволяет их однозначно декодировать.

Первым шагом при построении кодов переменной длины является построения таблицы частот. В ней находятся символы и в соответствие им ставится частота их встречаемости в файле. Затем строится «дерево кодирования Хаффмана». Символы входного алфавита образуют список свободных узлов и каждый элемент имеет свой вес, который равен количеству вхождений этого символа в файл или его вероятность встречаемости.

Затем, выбираются два узла с наименьшими значениями частот и создается новый узел(«родитель»), вес которого равен сумме частот (весов) этих двух узлов. Эти два узла удаляются, а родитель включается в список в соответствие со своим весом. Список обычно упорядочен по убыванию(крайний левый элемент с самым большим весом, крайний правый – с самым маленьким весом)

Одному ребру, выходящему из родителя, ставится единичный бит, другому ребру – нулевой. Каждое из ребер указывает на предыдущий узел. Затем ищется следующий родитель. Это все продолжается до тех пор, пока в списке

не останется один свободный элемент. Этот элемент будет называться «корнем дерева».

Теперь, чтобы получить код каждого символа, нужно пройтись от корня по всем ребрам до конечных узлов. Конечный узел – символ. Его код будет складываться из весов ребер, по которым алгоритм должен пройтись, чтобы добраться до конечного узла, записанный в обратно порядке. При декодировании каждый символ будет декодирован однозначно, т.к. алгоритм Хаффмана обладает свойством префиксности.

Алгоритм Хаффмана в классической интерпретации имеет несколько недостатков. Самый очевидный, это хранение таблицы частот. Чтобы алгоритм смог раскодировать и восстановить сжатые данные, он должен хранить таблицу частот. Соответственно увеличится вес файла. Следующий недостаток, это время затраченное на построение дерева и на повторный его проход. На больших и сложных изображениях это может негативно сказаться.

Рассмотрим пример алгоритма Хаффмана на строке: «Карл у Клары украл кораллы, а Клара у Карла украла кларнет».

На первом шаге, строится таблица частот:

а	_	к	л	р	у	н	е	т	К	,
12	10	4	9	8	4	1	1	1	4	1

Таблица 7.

Таблица упорядочивается по убыванию:

а	_	л	р	к	у	К	н	е	т	,
12	10	9	8	4	4	4	1	1	1	1

Таблица 8.

Строим дерево:

На каждом шаге объединяем 2 последних символа в один узел и сортируем таблицу:

1 шаг:

а	_	л	р	к	у	К	,т	н	е
12	10	9	8	4	4	4	2	1	1

Таблица 9.

2 шаг:

а	_	л	р	к	у	К	,т	не
12	10	9	8	4	4	4	2	2

Таблица 10.

3 шаг:

а	_	л	р	к	у	К	,тне

12	10	9	8	4	4	4	4

Таблица 11.

4 шаг:

а	_	л	р	К,тне	к	у
12	10	9	8	8	4	4

Таблица 12.

5 шаг:

а	_	л	р	К,тне	ку
12	10	9	8	8	8

Таблица 13.

6 шаг:

К,тне ку	а	_	л	р
16	12	10	9	8

Таблица 14.

7 шаг:

лр	К,тне ку	а	_
17	16	12	10

Таблица 15.

8 шаг:

_а	лр	К,тне ку
22	17	16

Таблица 16.

9 шаг:

лр К,тне ку	_а
33	22

Таблица 17.

10 шаг:

лр К,тне ку_а
55

Таблица 18.

На рисунке 5 представлено построенное дерево кодов Хаффмана. После алгоритм повторно проходится по дереву и составляет коды Хаффмана переменной длины. А в таблице 19 представлены итоговые значения, относительно входящих.

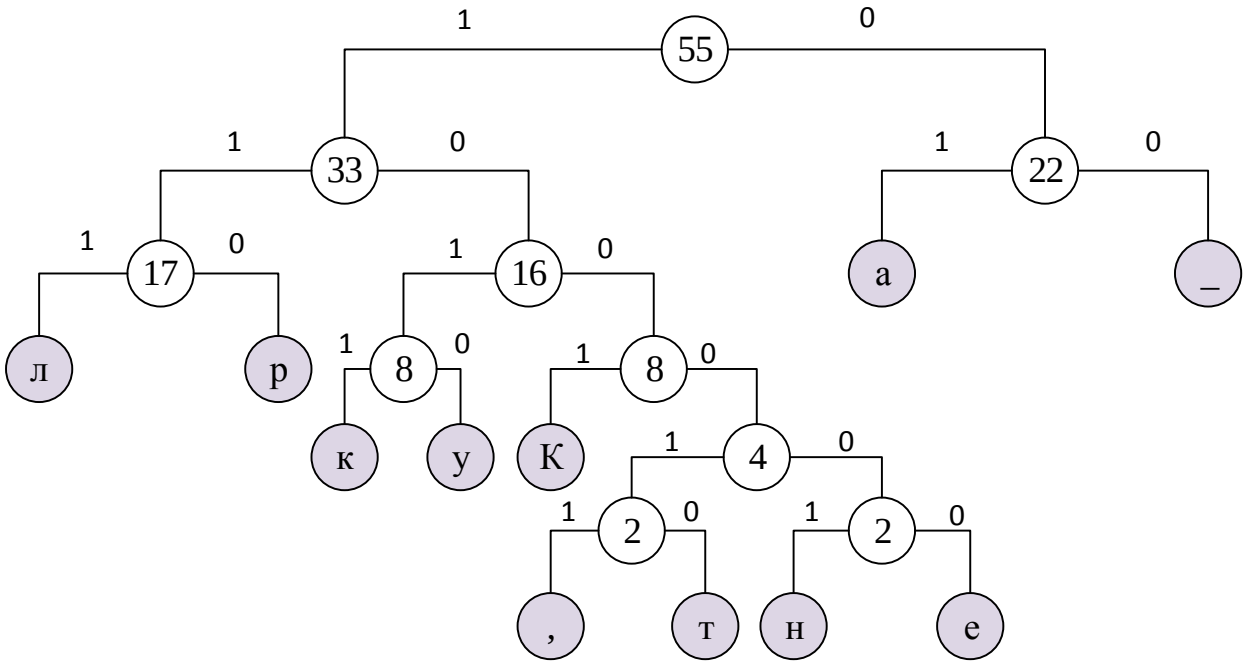

Рис.5 Дерево кодов Хаффмана.

Значение	**Код Хафф-мана**	**Частота**	**Длина кода**	**Использовано бит (длина кода * частота)**
а	01	12	2	24
_	00	10	2	20
л	111	9	3	27
р	110	8	3	24
к	1011	4	4	6
у	1010	4	4	6
К	1001	4	4	16
н	100001	1	6	6
е	100000	1	6	6
т	100010	1	6	6
,	100011	1	6	6
				Всего: 147

Таблица 19.

Как итог, текущая строка имела вес 440 бит (55 символов * 5 бит). В результате объем данных уменьшился ~70%. Но еще нужно место для хранения таблицы частот, для восстановления данных. Пример алгоритма Хаффмана продемонстрирован на простой строке символов. Если рассматривать метод Хаффмана для сжатия графического изображения, в соответствие с классом изображения, который лучше подходит для данного алгоритма, таблица частот не сильно изменит степень сжатия. В различных форматах хранения, таких как JPEG, PNG, где применяется данный алгоритм, описание

хранения таблиц Хаффмана указано в спецификации, соответственно формату.

Также существует две разновидности алгоритма: Статический(фиксированный) алгоритм Хаффмана и динамический. Они отличаются друг от друга тем, что статический уже хранит данные о переменных кодах, а динамический нет. У каждого метода свои плюсы и минусы: либо отводить часть данных под хранение таблиц, или жертвовать временем компрессии и декомпрессии.

Характеристики алгоритма Хаффмана	
Степени сжатия (лучшая, средняя, худшая):	200, 2, увеличивает в 5 раз
Класс изображений:	Двуцветные изображения, в которых преобладают большие пространства, заполненные одним цветом.
Симметричность:	~1
Характерные особенности:	Данный алгоритм прост в реализации и может легко быть реализован аппаратно

Таблица 20.Характеристки алгоритма Хаффмана.

Обобщение.

Существуют еще алгоритмы сжатия без потери информации, такие как JBIG, Lossless JPEG, PPM, Алгоритм Шеннона - Фано и несколько мало использующихся алгоритмов семейства LZ. Некоторые из них имеют либо узкую специализацию, либо запатентованы или вытеснены более хорошим алгоритмом. Рассматривать их подробно не имеет смысла, т.к. все равно алгоритмы, представленные выше, широко используются в различных утилитах, файлах по сей день.

Итак, алгоритм RLE очень хорошо подходит под простые изображения, содержащие большие длины серий, деловая графика, иконки рисунки. Но он очень плох для фотографий. RLE применяется в таких форматах хранения графической информации, как TIFF(в модификации PackBits), BMP, TGA, PCX и ILBM. Также формат применятся в формате хранения данных с потерями JPEG, с последующей обработкой методом Хаффмана. В этом случае RLE очень эффективен.

Алгоритмы семейства LZ* применяются гораздо чаще, чем RLE. Эти алгоритмы используются не только для сжатия изображений. Они применяются в утилитах Unix систем, таких как compress, gzip и в различных архива-

торах. Алгоритм LZW в настоящее время применяется в таких форматах, как TIFF, GIF, PDF, PostScript. На данный алгоритм и его вариации был выдан патент в США и других странах. В США были выданы два патента: «*U.S. Patent 4 814 746*», принадлежащий IBM, и патент Велча «*U.S. Patent 4 558 302*» (выдан 20 июня 1983 года), принадлежащий Sperry Corporation, позднее перешедший к Unisys Corporation. К настоящему времени сроки всех патентов истекли. Эти патенты сыграли негативную роль на формате GIF и методе LZW.

Когда разрабатывался формат хранения графических растровых изображений GIF, разработчики не знали о существовании патента. Когда компании Unisys стало известно об этом формате, она начала взыскивать лицензионные счисления с коммерческих программ, имеющих возможность по созданию GIF-файлов. Это было в разгар популярности формата GIF. Поэтому производителям ПО, работающим с GIF пришлось платить или отказываться от формата GIF. Это привело к разработке нового формата PNG, который вытеснил GIF. Срок действия патента истек в 2004 году во всех странах, к этому времени формат png стал очень популярен, как и по сей день. Он широко используется в сети интернет WWW.

Метод LZ78 стал менее популярен, чем LZW. Также использовался в формате GIF. Но его тоже ждала патентная участь, как и LZW. На LZ78 был выдан американский патент «*U.S. Patent 4 464 65*».

Широкую популярность получил алгоритм LZ77. Он применяется в форматах PNG, как модификация deflate, TIFF,GIF, в различных архиваторах: 7zip,gzip, pkzip. Также применяется в сетевых протоколах и утилитах ssh, PuTTY, http. Широкую популярность он получил благодаря применении алгоритма Хаффмана. LZ77+Huffman - получился метод Deflate. Сейчас он используется в формате PNG. Существуют две стандартные библиотеки (zlib/gzip), написанные на языке C, которые входят в пакет предустановленных программ Unix систем. Данный алгоритм будет рассмотрен мной позже в спецификации PNG.

Метод Хаффмана применяется в таких форматах, как PNG(deflate), Jpeg(JPEG), TIFF(CCITT Group 3). Можно заметить, что алгоритм Хаффмана в чистом виде мало где используется. Он обеспечивает наилучшее сжатие, после применения LZ77(deflate) или RLE(JPEG), т.к. позволяет лучше сжать повторяющие цепочки уже сжатый данных.

Рассмотренные алгоритмы без потерь универсальны и применяются почти во всех типах изображений. Далее будут рассмотрены методы сжатия информации с потерями. Методы сжатия без потерь для сжатия информации используют цепочки определенных цветов и частоты появления цветов. Ал-

горитмы сжатия с потерями позволяют добиться более высокой степени сжатия за счет когерентности. Это свойство заключается в особенности человеческого глаза. Человеческому глазу не заметны малые изменения цветов на небольшом участке изображения и алгоритмы сжатия с потерями этим пользуются.

Алгоритмы сжатия с потерями.

Алгоритм сжатия JPEG.

Данный алгоритм разработан специально для уменьшения размера фотографических изображений. Основан на принципе когерентности. Области 8x8 пикселей с плавным изменением яркости и цвета подвергаются дискретно – косинусоидальному преобразованию (ДКП). Данные сжатые методом JPEG хранятся в одноименном формате. Официальной спецификацией является документ – «ISO/IEC 10918-1». В нем описывается не только формат хранения, но и метод JPEG.Есть алгоритм сжатия без потерь, называется Lossless JPEG, но он проигрывает другим методам без потерь.

На первом этапе сжатия, данные преобразуются из цветовой палитры RGB в YCbCr. Это называется *дискретизацией* изображения. Подробнее это было описано в моей работе на странице 6. Затем, каждая из компонент разбивают на матрицы 8x8 пикселей, которые называются *единицами данных*. Каждый блок подвергается ДКП. Часто перед этим шагом, каналы Cb и Cr, которые отвечают за цветность, прореживают(матрице пикселей ставится усредненное значение)

ДКП – это одно из ортогональных преобразований. Тесно связано с преобразованием Фурье. ДКП преобразует входные пиксели в набор коэффициентов косинусных функций с возрастающими частотами. Каждая единица данных становится суммой косинусов. Преобразование ДКП происходит по следующим формулам:

$$\mathrm{T[i]} = \mathrm{c(i)} \sum_{n=0}^{N-1} V[n] * \cos \frac{(2n+1) * i * \pi}{2N} \qquad (3)$$

$$c(0) = \sqrt{\frac{1}{N}}, c(k) = \sqrt{\frac{2}{N}}, k\ != 0 \qquad (4)$$

$$\mathrm{V[i]} = \sum_{n=0}^{N-1} c(n) * T[n] * \cos \frac{(2i+1) * n * \pi}{2N} \qquad (5)$$

В уравнении (1) представлено одномерное преобразование ДКП массива Vиз чисел N в массив T из N чисел. Обратный процесс, с помощью кото-

рого можно восстановить исходные данные с высокой точностью, представлен уравнением (5) и называется *обратным ДКП*. В получившейся матрице коэффициентов левый верхний коэффициент называется DC, который является усредненным значением остальных 63-х коэффициентов, которые называются AC. Далее, идет следующий этап сжатия - *квантование*.

Квантование включает в себя нахождение и удаление коэффициентов, вклад в изображение которых минимален. За счет этого происходит потеря части информации. Получение значение вычисляется по формуле (6):

$$\text{Квантованное значение} = \text{Округление}\left(\frac{\text{Коэффициент ДКП}}{\text{Значение кванта}}\right) \qquad (6)$$

$$\text{Коэффициент} = \text{Квантованное значение} * \text{Значение кванта} \qquad (7)$$

Уравнением (7) представлен обратный процесс. Значения кванта не определяются из спецификации формата JPEG. Это массив из 64 элементов и называется *таблицей квантования*. Таблицы квантование зависят от конкретной программы. У каждого производителя фотоаппаратов программно заложены свои таблицы. Затем данные кодируются методом Хаффмана.

Перед тем, как данные будут закодированы методом Хаффмана, с целью максимальной группировки квантов, значения из матрицы выбираются в зигзагообразном порядке.

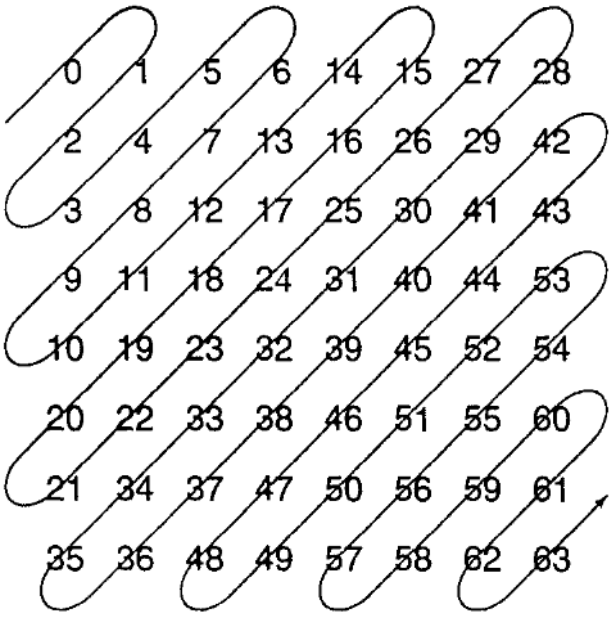

Рис.6. Загзагообразный способ выборки коэффициентов.

После полученный вектор из 64 элементов сжимается с помощью алгоритма RLE, который был рассмотрен выше. Затем каждая пара кодируется согласно методу Хаффмана. Процесс восстановления данных полностью

симметричен, все шаги делаются в обратном порядке. Процесс сжатия алгоритмом JPEG представлен блок схемой на рисунке 7.

Рис.7. Поэтапное представление алгоритма сжатия JPEG.

К главным достоинствам алгоритма можно отнести:

- Симметричность
- Можно регулировать степень сжатия
- Может применять частота дискретизации 24 бита на точку.

К недостаткам можно отнести:

- Появление линий границ областей и «лесенок» на границах резких переходов цветов(называется эффект Гибса)
- При высоком уровне сжатия на изображении заметный квадраты 8x8 пикселей одной цветности. Связано с потерей низких частот при квантовании.

Все же алгоритм эффективно применяется в различных фотоаппаратах и мультимедия. Также, широко используется в сети интернет и конкурирует с PNG. Характеристики алгоритма представлены в таблице 30.

Характеристики алгоритма JPEG	
Степень сжатия	200-2
Класс изображений	Полноцветные до 24 бит, фотографии.
Симметричность	1
Характерные особенности	При высокой компрессии заметно появление блоков 8x8 и появление эффекта Гиббса.

Таблица 30. Характеристики алгоритма JPEG.

Фрактальный алгоритм.

Как и алгоритм JPEG, данный метод ориентирован на фотографии. В основе метода лежат аффинные преобразования. Основной принцип алгоритма - обнаружение самоподобных областей на изображении. Функции, которые основаны на аффинных преобразованиях, называются *итерируемыми* (IFS). Основоположниками идеи алгоритма являются Майкл Барнсли и Алан Слоун. Они запатентовали свою идею в 1991 году. А сам алгоритм представил и реализовал А.Жакен.

Изображение разбивается на несколько подизображений, которые зовутся *ранговыми* и определяется стек перекрывающих изображений. Перекрывающее изображение называется *доменным*. Для каждого рангового блока алгоритм находит соответствующий ему доменный блок и аффинное пре-

образование, которое переводит этот доменный блок в ранговый. Изображение получается путем отображения ранговых блоков, доменных блоков и преобразований.

В данном алгоритме существует понятие *неподвижной точки*. Исходное изображение является неподвижной точкой некоего сжимающего изображения. Алгоритм, каким то образом запоминает отображение изображения на эту точку. Для восстановления изображения достаточно применить это отображение к любому начальному. Из этого следует, что сжимающие изображения обладают свойством: Для каждого изображения существует неподвижная точка и только одна. Самым распространенным примером является треугольник Серпинского (Рис.8). Для построения данного треугольника требуется всего 3 аффинных преобразования. Пример построения представлен на рисунке 9.

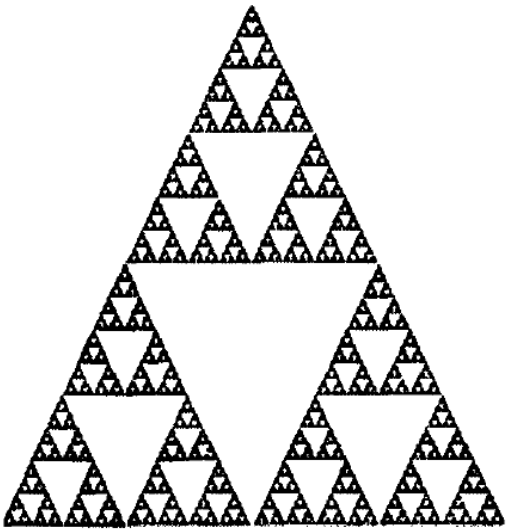

Рис.8. Треугольник Серпинского.

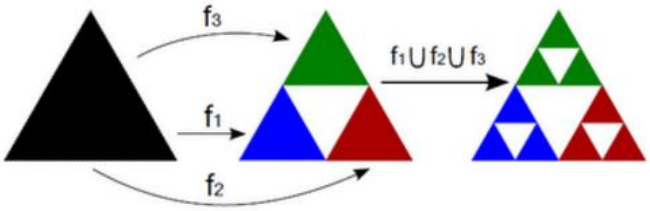

Рис.9. Построение треугольника Серпинского.

Исходный треугольник 3 раза множится, уменьшается и переносится. И так далее. Частота применения действий играет важную роль на качество изображения. В графике неподвижной точкой считается не обычная точка в пространстве, а множество точек, которое образует изображение. В данном случае это треугольник. Для изображений – эта точка называется *аттрактор*.

Самой сложной задачей алгоритма считается нахождение аттрактора. Метод оперирует, такими понятиями, как расстояние между точками, кото-

рое показывает, насколько области близки между собой. Отображение делает эти две области более похожими. Алгоритм сжатия состоит из следующих шагов, которые применяются рекурсивно:

1. Исходное изображение делится на квадратные области, которые не перекрываются. Эти области называются ранговыми.
2. Строится стек возможных перекрывающих блоков(доменных) в 4 раза больше ранговых.
3. Каждый ранговый блок по очереди сравнивают с доменным и ищется такое преобразование, которое превращает доменный блок похожим на текущий ранговый блок.
4. Пара «преобразование – доменный блок», которая приблизилась к идеалу, приравнивается ранговому блоку.

В закодированном изображении хранятся коэффициенты преобразования и координаты доменного блока. Содержимое блока не хранится. Пример декодирования представлен на рисунке 10.

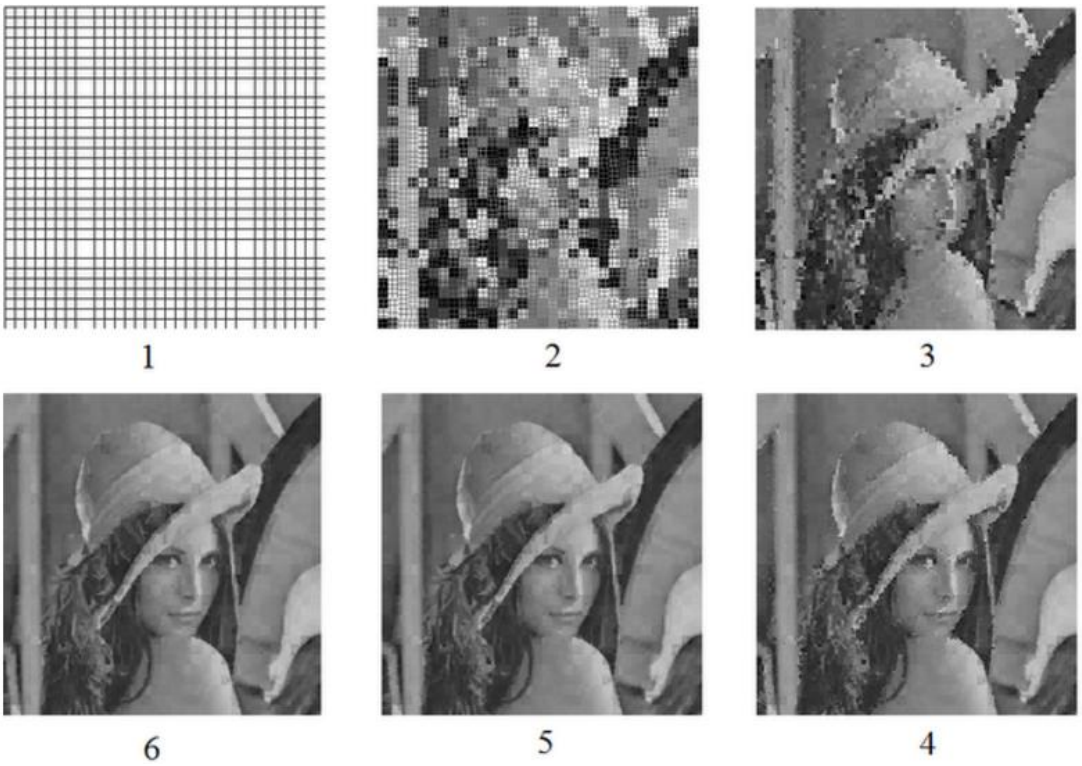

Рис.10. Пример восстановления изображения фрактальным алгоритмом.

По сравнению с отысканием оптимального изображения, декодирование происходит очень быстро. Изображение делится на ранговые области и, постепенно, заменяет их на результат применения преобразований. Каждое преобразование соответствует своей доменной области. Качество изображений зависит от количества применения итераций. Фрактальный алгоритм похож принципом на алгоритмы сжатия семейства LZ*. К основным недостаткам относятся:

- Скорость обнаружения неподвижной точки. Для хорошо детализованного изображения требуется достаточно много времени для отыскания.
- Невозможность применять области, отличные от квадратов.
- Алгоритм не сможет корректно распознавать области, которые повернуты.

Данный алгоритм хорошо применим на слабо детализованных изображениях, за счет простоты хранения данных и степени компрессии. Степень компрессии может быть очень большой, если качество изображения не важно. Характеристики фрактального алгоритма представлены в таблице 31.

Характеристики фрактального алгоритма.	
Степень сжатия	2000 -2
Класс изображений	Высококонтрастные фотографии
Симметричность	100-100000
Характерные особенности	Может масштабировать изображения в несколько раз без сильной потери качества изображения. При увеличении степени изображения, появляются искажения, как и у алгоритма JPEG

Таблица 31. Характеристики фрактального алгоритма сжатия.

Волновой алгоритм.

Данный метод ориентирован на черно белые изображения. Алгоритм идеален для рентгеновских снимков. Метод алгоритма очень прост. Как и в случае с методом JPEG, он основан на принципе когерентности. Значение пары пикселей усредняется. Также для обеспечения максимального сжатия берут по 4 пикселя(квадрат 2x2). Затем цепочки усредненных значений рекурсивно усредняются. Затем в исходный файл сохраняется число – разница между значениями соседних блоков. Затем эти данные сжимаются методом Хаффмана. Алгоритм прост в реализации. В отличие от предыдущих двух методов сжатия с потерями, он не оперирует блоками определенных размеров. Пример волнового алгоритма:

Пусть есть цепочка значений пикселей {100, 110, 234, 120, 110,255, 140, 100, 240, 105, 110, 240}. Получим две цепочки:

1. A1 = {105, 177, 182.5, 120, 172.5, 175}
2. A2 = {5, 57, 72.5, 20, 67.5, 65}

Затем рекурсивно рассматриваем цепочку A1:

1. $A1_1$ = {141, 151.25, 173.75}
2. $A2_1$ = {36, 31.25, 1.22}

Цепочка $A1_1$ может дальше рекурсивно сжиматься, пока не останется 2 значения. Финальная цепочка A2 кодируется методом Хаффмана. Число итераций задается программой или пользователем. Характеристики волнового алгоритма представлены в таблице 32.

Характеристики волнового алгоритма.	
Степень сжатия	2-200
Класс изображений	Фотографии с плавным переходом цветов.
Симметричность	~1.5
Характерные особенности	С увеличением степени сжатия, на изображении не заметны блоки пикселей

Таблица 32. Характеристики волнового алгоритма сжатия.

Обобщение.

Алгоритмы сжатия с потерями чаще всего применяются к фотографиям. В зависимости от детализации изображения, проявляются эффекты от потери информации. В полиграфии и в чертежах данные алгоритмы использовать не рекомендуется, т.к. мелкие детали будут отображаться не корректно, вместо прямых линий может появиться эффект «лесенки». Также не рекомендуются в качестве промежуточного формата хранения.

JPEG применяется во всех фотоаппаратах и хорошо работает с палитрой 24 бита. Поддерживается в QuickTIme, PostScript, TIFF 6.0 и во всех видео и мультимедия приложениях. Также применяется в сети интернет, для загрузки первоначального изображения в плохом качестве. В системах Unix имеется стандартная библиотека libjpeg для работы с jpeg. Поддерживает прогрессивное сжатие.

Фрактальные алгоритмы широко применяются в медицине. Очень хорошо они подходят для создание рентгеновских снимков. Очень часто фракталы применяются в геологии и геофизике. Не секрет что побережья островов и континентов имеют некоторую фрактальную размерность. Также фрактальные алгоритмы применяются в некоторых играх для создания однотипного ландшафта. Все же применение данного алгоритма очень узко.

Волновые алгоритмы применяются как замена ДКП в методе JPEG. Такая модификация называется JPEG2000. В результате изображение получается более мягким и четким по сравнению с JPEG.Также применяются в медицине, для снимков. Применяются в цифровом кинематографе.

Заключение. Обоснование и выбор алгоритма.

Выше были рассмотрены все ключевые алгоритмы, которые применяются в большинстве современных компрессоров. Все они делятся на две большие группы методы сжатия без потерь или с потерями. Также, изображения, на которые собственно и направлены алгоритмы, были поделены на 4 основных класса. Рассмотрение подробно всевозможных алгоритмов сжатия и специфичных классов изображения уходит за рамки данной работы. Все различные модификации алгоритмов берут свои истоки от рассмотренных выше. Методы словарного сжатия, такие как RLE, LZ используют для сжатия принцип одинаковых цепочек цветов. Если для хранения закодированной последовательности требуется меньше объема, чем для записи самого изображения, то наблюдается эффект сжатия. Если нет, то алгоритмы могут увеличить изображение.

Эффект сжатия в методе Хаффмана наблюдается за счет частоты появление определенного цвета на изображении. Удобен для последующего сжатия цепочек, после словарных методов. Есть две основные вариации алгоритма Хаффмана – статический и динамический.

В первом случае программа пробегается по всему файлу и подсчитывает частоту встречаемости символов. Затем строится дерево Хаффмана. Из дерева извлекаются переменные коды и осуществляется еще один проход. При использовании статического Хаффмана требуется два прохода по файлу.

Динамический алгоритм требует всего один проход. Частота встречаемости уже инициализируется в таблице и хранится в файле. С каждым встречаемым символом частота его появления увеличивается динамически. Выиграв в количестве проходов теряется качество сжатия.

Михаил Янковой в статье «Динамическое сжатие методом Хаффмана» в статье с сайта *compression.ru* утверждает - «Статическое кодирование Хаффмана довольно широко применяется в современных алгоритмах сжатия, но не в чистом виде, а как одна из ступеней сжатия в более сложных алгоритмах. Динамический же вариант алгоритма на практике, в последнее время, уступил более гибким и скоростным алгоритмам и применяется в основном только в экспериментальных целях».

Методы сжатия с потерей информации основаны на принципе когерентности. Эффект сжатия происходит за счет объединения областей с плавным переходом цветов. Метод JPEG обширно используется в фотографии, так как наиболее удачно подходит для сжатия таких изображений. Фрактальные и волновые алгоритмы имеют узкое назначение, но применяются к фотографиям. Волновой метод может служит как замена ДКП в формате JPEG, что привело за собой к другой модификации JPEG 2000.

Обобщение всех рассмотренных алгоритмов приведено в таблицах ниже. За основу взят труд Дмитрия Ватолина, Максима Смирнова, Вадима Юкина, Евгения Шелвина, Дмитрия Шкарина, Александра Ратушняка, Сергея Оснача и их детище проект compression.ru. Данный проект является сборником различных материалов, книг, статей, авторами большинства которых являются создатели. Проект существует с 2001 года и поддерживается до сих пор. Описанные методы, протестированы более чем на 100 различных архиваторах и можно смело полагаться на их статистику.

Алгоритм	**Особенности, благодаря которым происходит сжатие**
RLE	Подряд идущие одинаковые цвета
LZ*	Одинаковые подцепочки
Хаффмана	Разная частота появления цветов
JPEG	Отсутствие резких границ
Фрактальный	Подобие между элементами изображения
Волновой	Плавные переходы цветов и отсутствие резких границ

Таблица 33. Алгоритмы сжатия и их особенности.

Алгоритм	**Коэффициент сжатия**	**Симметричность**	**Ориентированность**	**Потери**
RLE	32,2,0.5	1	3,4-битные	Нет
LZ*	1000,4,5/7	1.2-3	8 битные	Нет
Хаффмана	8, 1.5, 1	1-1.5	8 битные	Нет
JPEG	2 - 200	~1	24 битные	Да
Фрактальный	2 - 2000	1000-10000	24 битные	Да
Волновой	2 - 200	1.5	24 битные	Да

Таблица 34. Характеристики алгоритмов.

Если рассматривать растровый класс изображений, как деловая графика, чертежи, изображения, построенные в различных САПР и сжатие без потерь, мною будет выбран алгоритм из семейства LZ* с последующим кодированием Хаффмана, для обеспечения, как я отмечал ранее, максимального сжатия. Они показывают хороший коэффициент, ориентированы на 8 битные изображения и просты в реализации.

Если рассматривать растровый класс изображений, как фотография и сжатие с потерями, мною будет выбран алгоритм JPEG, так как он является лидером в этой области. Фрактальные и волновые эффективны на узком классе изображений.

Стоит отметить, что выбранные алгоритмы нужно согласовать с форматом хранения данных. Т.к. формат изображения тоже направлен на определенный класс изображений, нельзя использовать любой алгоритм в качестве метода сжатия в определенном формате хранения. Как и методам сжатия, ни одному формату нельзя отдать полного предпочтения. У каждого свои области применения. Например, я не смогу использовать метод LZ77 без потерь в формате хранения JPEG. Форматы будут рассмотрены в следующей главе.

Глава 2. Форматы хранения графических изображений.

Разница формата хранения и алгоритма.

Цветные растровые изображения требуют более мегабайта для своего хранения, поэтому в каждом формате предусмотрен свой алгоритм сжатия. Единственным совпадением алгоритма сжатия и формата является JPEG. Также, в одном формате хранения, может быть реализовано несколько алгоритмов сжатия. Стоит отметить, что если файл занимает много места, это не означает, что выбран плохой формат хранения. Это может означать, что плохо выбран алгоритм сжатия. Например, формат TIFF 6.0 поддерживает такие методы сжатия, как RLE, LZW, LZ77, JPEG, Хаффмана. Соответственно, каждый из алгоритмов даст свою степень сжатия. В этом главный недостаток формата – неоднозначность.

Для неоднозначных форматов по выбору алгоритма, стоит указывать, какой метод сжатия используется. На итоговый результат также влияют различные параметры алгоритма, использующиеся при компрессии. Так, разные компрессоры используют свои параметры. Из-за этого разные программы могут показывать разное качество и степень сжатия для одинакового алгоритма и графического изображения. Как правило, режимы управления параметрами рассчитаны на опытных пользователей. Среди этих параметров могут быть: максимальная длина кодов Хаффмана, матрицы квантования, окно в LZ77, размер словаря, порядок обхода байт, количество итераций и т.д. Ниже рассмотрены наиболее известные и популярные по применяемости и пользованию в приложениях форматы хранения графических изображений в ОС Windows, Linux и MacOS.

Формат BMP.

Разработчик Microsoft, является встроенным в Windows. Он поддерживает изображения с 1-24, 32 битами на пиксель. Формат ориентирован на растровую графику. Расширение *.bmp, .dib* или *.rle*. Предусматривает в себя встроенные файлы JPEG и PNG (это сделано не для компактности хранения, а для удобства использования в различных целях). Изображения в данном формате хранятся, как правило, без использования алгоритмов сжатия. Также, редко встречаются изображения с частотой дискретизации 16 и 24 бит. Формат обеспечивает сжатия без потерь. Если разрабатываемое программное обеспечение на платформе Windows, это наиболее лучший формат для отладки и его можно рассматривать без различных специальных средств.

Используемые алгоритмы – это RLE4 и RLE8. RLE8 идентичен RLE из первой главы моей работы. RLE4 отличается только тем, что байт, следующий за счетчиком, хранит в себе два значения. В старших 4-х битах хранится значение первого и нечетных пикселей, в младших 4-х битах хранится значение четных пикселей. Применяется, если частота дискретизации равная 4.

Официальную информацию по формату BMP можно найти в MSDN или справке Microsoft Windows SDK. В файле WinGDI.h от компании Microsoft есть все объявления на языке C++, которые касаются данного формата. Структуры BMP включаются из заголовочного файла windows.h. Структура представлена на рисунке 10.

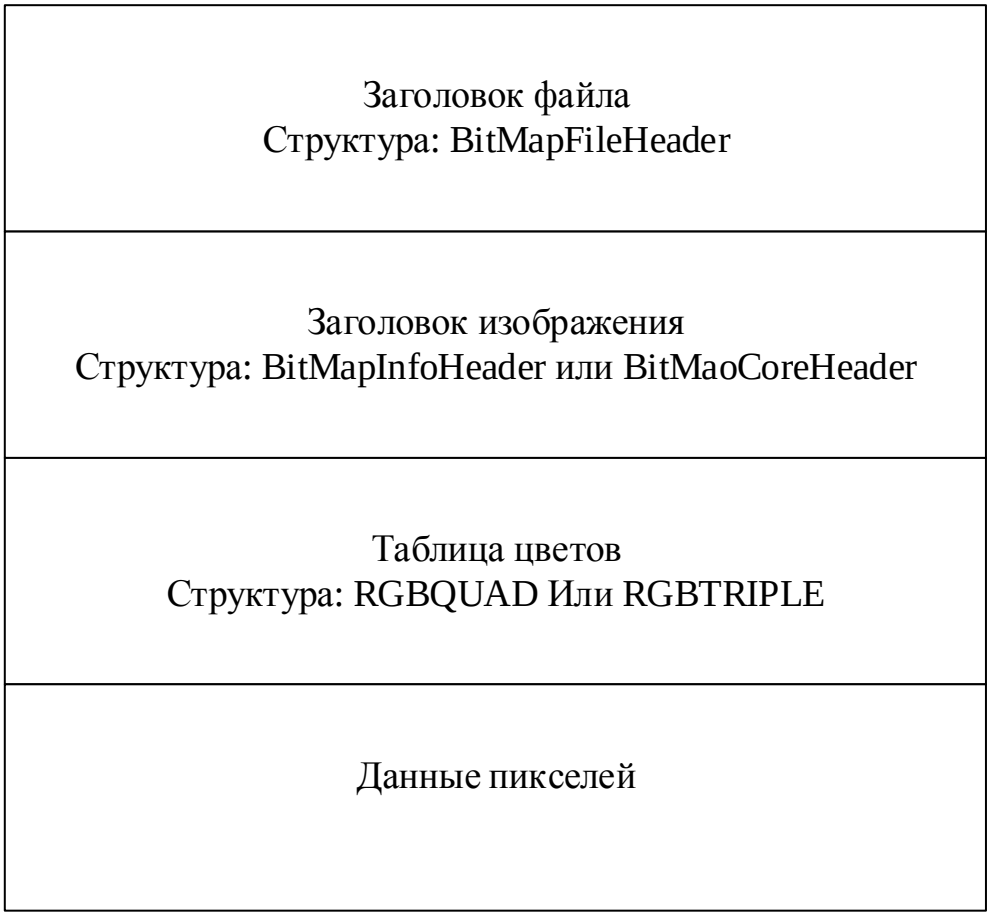

Рис.10. Структура BMP файла.

Каждый файл BMP начинается с заголовка файла. Это своеобразная проверка, BMP это или нет. Первые два байта должны содержать 2 символа «B» и следуемая за ним «M». Следом сразу следует заголовок изображения. Он может быть в одном или двух форматов. Эта структура задает размеры, глубину цвета и проверку, было ли сжатие или нет. Затем, идет цветовая палитра, которая отображает значения цветов RGB. Далее идет информация о пикселях, более подробно она описана в спецификации. Декомпрессор, работающий с форматом BMP, должен корректно распознавать все структуры. Разработка данного формата полностью ведется компанией Microsoft.

Формат TIFF.

TIFF является теговым форматом хранения растровых изображений(Tagged Image File Format). Расширение: .tif и .tiff. Разработан Microsoft и Aldus, сейчас Adobe Systems. Обеспечивает сжатие без потерь, за исключением алгоритма JPEG. Поддерживается в большинстве Unix систем. Существует LibTIFF – набор библиотек и утилит для работы с TIFF файлами. Формат TIFF поддерживает такие алгоритмы сжатия, как RLE(модификация PackBits), LZW, LZ77, JPEG, JBIG, Статический Хаффмана, Deflate(LZ77 + Хаффман)**.**