МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РЕСПУБЛИКИ КАЗАХСТАН

ПАВЛОДАРСКИЙ УНИВЕРСИТЕТ

МАГИСТРАТУРА

Кафедра "Информатика и вычислительная техника"

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

ИСПОЛЬЗОВАНИЕ ИНФОРМАЦИОННОЙ ИЗБЫТОЧНОСТИ ФАЙЛОВ ДЛЯ ПЕРЕДАЧИ И ХРАНЕНИЯ СКРЫТОЙ ИНФОРМАЦИИ

541550 "Информационные системы"

Исполнитель:

28.04.05 **Глухов С.А.**

(подпись, дата)

Научный руководитель:

Профессор, к.т.н. 28.04.05 **Фандюшин В.И.**

(подпись, дата)

Допущен к защите:

Зав. кафедрой "ИВТ"

Профессор, к.т.н. **Асамбаев А.Ж.**

(подпись, дата)

Павлодар 2005

РЕФЕРАТ

Объектом исследования магистерской диссертации различные форматы файлов (текстовые, графические, мультимедиа).

В данном проекте содержится 109 листов машинописного текста и 6 листов графической части. Магистерская диссертация выполнена с графической частью изготовленной в виде фолий, размеры которой определяются форматом А4. Содержится 19 иллюстраций 9 таблиц и 15 формул.

Целью диссертации является исследование различных форматов файлов для использования их как "контейнер" для хранения и передачи скрытых сообщений. Для достижения своей цели выделил задачи: Рассмотреть форматы файлов: текстовые (txt, doc, rtf), графические (bmp, jpeg, gif, tif), мультимедиа (avi, mpeg, mp3); Рассмотреть возможность использования младших битов для хранения и передачи скрытых сообщений; Разработать алгоритмы и программное обеспечение для стеганографической передачи и хранения информации.

В первой части проекта выполняется постановка проблемы стеганографии и план решения задач.

Вторая часть магистерской диссертации посвящена форматам файлов различной направленности, таких как: текстовые (txt, doc, rtf), графические (bmp, jpg, gif, tif) и мультимедиа (avi, mpeg, divx).

В третьей части проекта представлена, разработанная мной программа SteganoGraphyV2.2, её основные элементы (модуль Gtaph.tpu и вспомогательный модуль интерфейса) выполненные на Delphi7.

Четвертая чать содержит результаты маркетингового исследования по вопросу внедрения программы на рынке IT-технологий

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, СИМВОЛОВ И ОБОЗНАЧЕНИЙ

AES/EBU (Audio Engineers Society / European Broadcast Union) - Общество Звукоинженеров / Европейское Вещательное Объединение - цифровой интерфейс для студийной радиоаппаратуры. Цифровой интерфейс позволяет передавать звуковые сигналы между аппаратурой без потери качества, которое неизбежно теряется при передаче сигналов в аналоговой форме.

AVI - Audio Video Interleaved, оригинальная аббревиатура для Microsoft Video For Windows

AVI MPEG (Editable MPEG) - разновидность MPEG-формата. Структура AVI MPEG основана только на И-кадрах, поддающихся редактированию в любой видео-редакторе. Для дальнейшего преобразования этого формата в стандартный MPEG-файл необходимо перекодировать его на основе И-, П- и Д-последовательностей (IPB).

BitMap – битовая карта изображения. Оно же BMP. Формат хранения графических данных

CCIR-601 - CCIR является аббревиатурой Интернационального Комитета по Телеграфу и Телефонии (International Committee on Telegraph and Telephones), стандарт 601 описывает формат цифрового видео с разрешением 720 x 486 при частоте 30 Гц

CODEC (кодек) - Coder and Decoder - сокращенная аббревиатура для микросхемы или программного драйвера, осуществляющих кодирование и декодирование определенных данных (например аппаратный CODEC Motion-JPEG)

Decode (декодирование) - термин, определяющий процесс декомпрессии данных

DVI - Digital Video Interactive, схема сжатия видео-данных фирмы Intel, которая не была утверждена комитетом ISO в свете развития MPEG стандарта

Encode (кодирование) - термин, определяющий процесс сжатия данных

Field - Поле. Отдельное изображение в составе видео-потока. Каждый кадр состоит их четных и нечетных строк (черезстрочная развертка). Нечетные строки формируют нечетное поле, четные строки - четное поле. Видеосигнал может записываться с полной разверткой (25 кадров/50 полей) и частичной разверткой (25 кадров/25 полей)

IEC - International Electrotechnical Commission, Интернациональная Электротехническая Комиссия - государственная организация, работающая с ISO

Indeo - оригинальный формат цифровой видеозаписи фирмы Intel

Interpolation - интерполяция, специальный алгоритм масштабирования исходной картинки с целью получить полноэкранное изображение. На этом принципе работают MPEG-распаковщики, "растягивая" кадры с разрешением 352 x 240 до полноэкранного формата

ISO - International Standards Organization, Международная Организация по Стандартам

Киоск - автономная система для предоставления информации

NTSC - National television Standarts Committee, Национальный Комитет по телевизионным Стандартам

Pre-filtering - предварительная фильтрация, - процесс обработки изображения перед сжатием

Post-filtering - последующая фильтрация, - процесс обработки изображения после сжатия

QuickTime - системные расширения фирмы Apple для просмотра и монтажа цифрового видео; соответствующий видео формат

PCM (Pulse Code Modulation) - импульсно-кодовая модуляция - стандартный способ цифрового кодирования звукового сигнала при помощи последовательности абсолютных значений амплитуды

Real-Time - процесс, происходящий в режиме реального времени (например, на запись одной минуты MPEG требуется одна минута времени)

RGB – Цветовая палитра одного пиксела. Red - красный, Green - зеленый, Blue - голубой

SIF - термин, описывающий компьютерное разрешение 352x240 точек, соответствующее разрешению VHS (QSIF - Quarter SIF, т.е. четверть от разрешения SIF: 176x120)

S/PDIF (Sony/Philips Digital Interface Format) - Формат Цифрового Интерфейса фирм Sony и Philips - цифровой интерфейс для бытовой радиоаппаратуры; S/PDIF представляет собой упрощенный вариант AES/EBU и используется, в частности, для вывода в цифровом формате сигнала с компакт-дисков.

Square Pixel Resolution - компьютерное разрешение видеосигнала, соответствующее стандартному VGA-режиму 320x240

VFW - Video for Windows, интерфейс и формат цифрового видео фирмы Microsoft

VTR - Video Tape Recorder, магнитофон для записи и воспроизведения видео

VOD - Video on Demand, видео по требованию, термин, описывающий возможность для пользователя в данный момент времени затребовать любой выбранный им для просмотра видеофильм

SMPTE - Society of Motion Picture and Television Engineers, Сообщество Инженеров Кино и Телевидения. Одноименный формат кода для синхронизации аудио- и видео-данных.

WAV - широко распространенный звуковой формат для PC.

URL - аббревиатура HTTP адреса в Internet

ЗИ – защита информации.

Кодировки – формат представления символа в файле (DOS, ISO, KOI8, Windows)

Контейнер – исходный файл предназначенный для хранения скрытого сообщения

КС – компьютерная стеганография.

Пиксел – (Picsel- Picpure element)24х битный размер включает в себя набор RGB и так называемый альфа-канал

ПО – программное обеспечение.

Символ – цифровое представление цыфро-буквенного значения в кодировке. Размер символа 1 байт.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современная тенденция развития мирового сообщества наполнена колоссальным количеством секретов, которые в свою очередь становятся мощным оружием в руках, как белых, так и темных сил. Помимо владения ими важную роль играет и механизмы коммуникации. Ведь умелое и своевременное применение секрета в нужном месте может произвести взрыв отлаженного организма структуры (от экономического баланса страны, до локальных конфликтов в «горячих точках»). Надежными способами передачи этого разрушительного оружия являются линии телекоммуникаций. К примеру, после событий, произошедших 11 сентября в США, многие задаются вопросом, где же были спецслужбы, почему они не смогли предупредить о готовящемся теракте такого масштаба? Представители и Агентства национальной безопасности, и Центрального разведывательного управления, и Федерального бюро расследований заявляют о сложностях перехвата сообщений, которыми обмениваются террористы, и не в последнюю очередь речь идет об электронной коммуникации, которую этим ведомствам не удается засечь и расшифровать. Именно поэтому ФБР и сейчас активно призывает к сотрудничеству ведущих американских провайдеров, устанавливает собственные системы мониторинга электронного трафика и снова поднимает вопрос о необходимости существования для правительственных организаций «черного хода» для систем сокрытия информации, созданных в США. Защитники же гражданских свобод считают, что спецслужбы просто пытаются воспользоваться моментом и легально установить системы, призванные на самом деле не столько обнаруживать террористов, не ограничивающихся Интернетом, сколько мониторить личную переписку простых граждан США, что, конечно же, является нарушением гражданских прав и ущемлением демократических свобод. Однако оставим разборки о правах и свободах американцам и попробуем поиграть в шпионов.

Основная задача секрета, чтобы он оставался секретом. Одно дело, когда он находится в «покое» и совершенно другое, когда возникает необходимость

передать его. К примеру: сын студент голодает в далеком городе и просит о посильной материальной помощи родителей. За неимением возможности денежных переводов из села Гадюкино отец решается отправить 5000 тенге по почте. Как вы думаете, поест ли в этом месяце голодный студент? Я намеренно оставлю этот вопрос открытым. При транспортировке секрета задействуются различные каналы связи (от почты, до высокоскоростного оптоволоконного кабеля) и существует вероятность недобросовестного поставщика транспортных услуг нашего секрета. Ведь вложи отец в конверт с 5000-ми пару фотографий с достопримечательностями поселка, возможно и не случилось бы голодного обморока на лекции «Защита информации в компьютерных системах». Не вдаваясь в нюансы поведения властей можно сделать вывод, что передача секретной информации необходима и простым смертным.

1. ПОСТАНОВКА ПРОБЛЕМЫ

На сегодняшний день существуют два способа сокрытия сообщений. Это криптография (с др. греч. – измененное письмо) и стеганография (с др. греч.- тайное письмо)

Сообщение, требующее конфиденциальной передачи, принято называть *открытым текстом*. Процесс преобразования открытого текста с целью сделать непонятным его смысл для посторонних называется *шифрованием*. В результате шифрования сообщения получается *шифртекст*. Процесс обратного преобразования шифртекста в открытый текст называется *дешифрованием* (*расшифрованием*). Наука, изучающая методы преобразования (шифрования) информации с целью её защиты от незаконных пользователей, называется *криптографией*.

Рассмотрим основные элементы схемы традиционного шифрования (Рис.1).

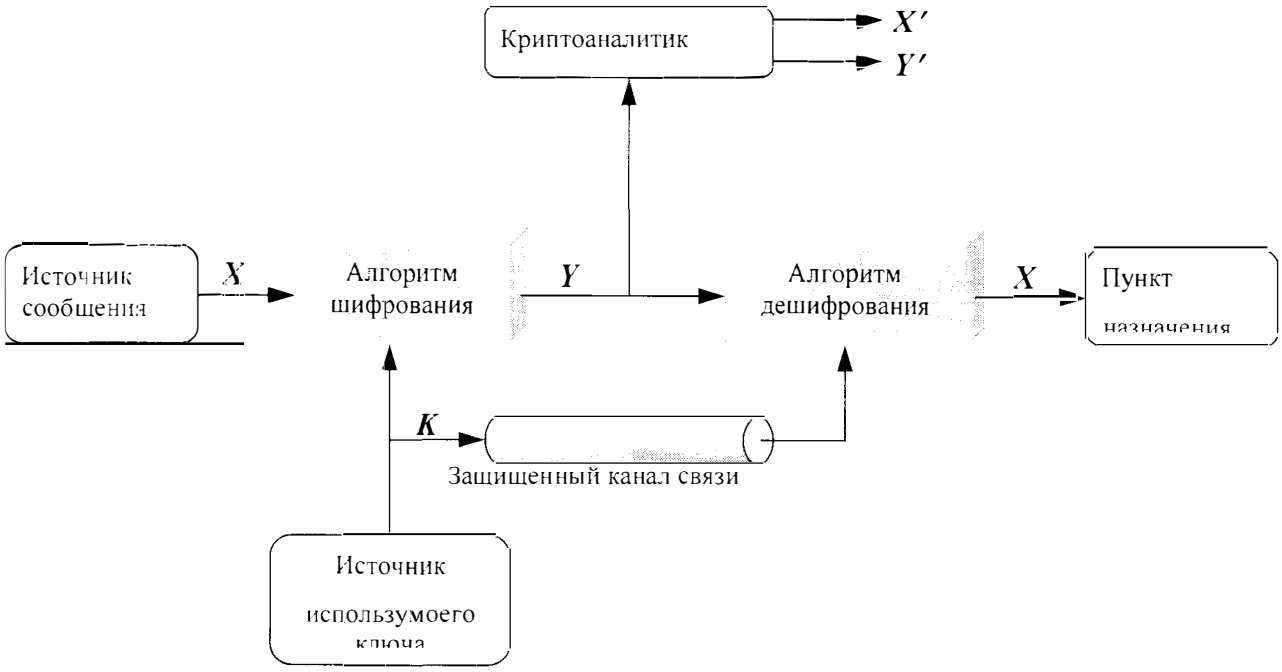


Рисунок 1

Модель традиционной криптосистемы.

Источник создает сообщение в форме открытого текста $X=[X_1,X_2,\ldots, X_M]$. Элементами X_i открытого текста X являются символы некоторого конечного алфавита. Для шифрования генерируется ключ в форме $K=[K_1,K_2,\ldots, K_J]$. При наличии в качестве исходных данных сообщения X и ключа шифрования K с помощью алгоритма шифрования формируется шифрованный текст $Y=[Y_1,Y_2,\ldots, Y_N]$. Это можно записать в виде формулы

$$Y=E_K(X). \quad (1)$$

Данная нотация означает, что Y получается путем применения алгоритма шифрования E к открытому тексту X при использовании ключа K.

Предполагаемый получатель сообщения, располагая ключом K, должен иметь возможность выполнить обратное преобразование

$$X=D_K(Y). \quad (2)$$

Противник, обладающий возможностью ознакомиться с Y, но не имеющий доступа ни к K, ни к X, может попытаться восстановить X или K или оба этих объекта. При этом подразумевается, что противник знает и алгоритм шифрования (E), и алгоритм дешифрования (D). Если противник заинтересован распознать только одно конкретное сообщение, ему следует сосредоточить свои усилия на восстановлении X путем построения вероятно соответствующего исходному открытого текста X'. Однако чаще противник бывает заинтересован в получении возможности читать и все последующие сообщения. В этом случае его основные усилия должны быть сосредоточены на восстановлении K путем построения вероятно соответствующего исходному ключа K'.

В отличие от криптографических методов с явно заметным присутствием информации, стеганография имеет преимущество в том, что сообщение скрытое этим методом не предполагает явной видимости сообщения. История стеганографических приемов начинается со времен появления письменности.

Первые следы стеганографических методов теряются в глубокой древности. Так, например, общеизвестно, что в древней Греции тексты писались на дощечках, покрытых воском. Во избежание попадания сообщения к противнику, использовали следующее ухищрение. Соскабливали воск с дощечек, писали сообщение прямо на поверхности дерева, потом снова покрывали дощечку воском. Таблички выглядели без изменений и потому не вызывали подозрений.

Хорошо известны различные способы скрытого письма между строк обычного не защищаемого письма: от применения молока до использования сложных химических реакций с последующей обработкой при чтении.

Другие методы стеганографии включают использование микрофотоснимков, незначительные различия в написании рукописных символов, маленькие проколы определенных напечатанных символов и множество других способов по скрытию истинного смысла тайного сообщения в открытой переписке. Сегодня, в то время когда компьютерные технологии придали новый импульс развитию и совершенствованию стеганографии, появилось новое направление в области защиты информации — компьютерная стеганография (КС).

Современный прогресс в области глобальных компьютерных сетей и средств мультимедиа привел к разработке новых методов, предназначенных для обеспечения безопасности передачи данных по каналам телекоммуникаций и использования их в необъявленных целях. Эти методы, учитывая естественные неточности устройств оцифровки и избыточность аналогового видео или аудио сигнала, позволяют скрывать сообщения в компьютерных файлах «контейнерах». Причем, в отличие от криптографии, данные методы скрывают сам факт передачи информации.

К. Шеннон дал нам общую теорию тайнописи, которая является базисом стеганографии как науки. В современной компьютерной стеганографии существует два основных типа файлов: сообщение— файл, который предназначен для скрытия, и контейнер—файл, который может быть

использован для скрытия в нем сообщения. При этом контейнеры бывают двух типов. Контейнер—оригинал (или “Пустой” контейнер) — это контейнер, который не содержит скрытой информации. Контейнер—результат (или “Заполненный” контейнер) — это контейнер, который содержит скрытую информацию. Под ключом понимается секретный элемент, который определяет порядок занесения сообщения в контейнер.

Основными положениями современной компьютерной стеганографии являются следующие:

1. Методы скрытия должны обеспечивать аутентичность и целостность файла.
2. Предполагается, что противнику полностью известны возможные стеганографические методы.
3. Безопасность методов основывается на сохранении стеганографическим преобразованием основных свойств открыто передаваемого файла при внесении в него секретного сообщения и некоторой неизвестной противнику информации — ключа.
4. Даже если факт скрытия сообщения стал известен противнику через сообщника, извлечение самого секретного сообщения представляет сложную вычислительную задачу.

В связи с возрастанием роли глобальных компьютерных сетей становится все более важным значение стеганографии. Анализ информационных источников компьютерной сети Internet позволяет вделать вывод, что в настоящее время стеганографические системы активно используются для решения следующих основных задач:

1. Защита конфиденциальной информации от несанкционированного доступа;
2. Преодоление систем мониторинга и управления сетевыми ресурсами;
3. Камуфлирование программного обеспечения;
4. Защита авторского права на некоторые виды интеллектуальной собственности.

Остановимся подробнее на каждой из перечисленных задач.

Защита конфиденциальной информации от несанкционированного доступа. Это область использования КС является наиболее эффективной при решении проблемы защиты конфиденциальной информации. Так, например, только одна секунда оцифрованного звука с частотой дискретизации 44100 Гц и уровнем отсчета 8 бит в стерео режиме позволяет скрыть за счет замены наименее значимых младших разрядов на скрываемое сообщение около 10 Кбайт информации. При этом, изменение значений отсчетов составляет менее 1 %. Такое изменение практически не обнаруживается при прослушивании файла большинством людей.

Преодоление систем мониторинга и управления сетевыми ресурсами. Стеганографические методы, направленные на противодействие системам мониторинга и управления сетевыми ресурсами промышленного шпионажа, позволяют противостоять попыткам контроля над информационным пространством при прохождении информации через серверы управления локальных и глобальных вычислительных сетей.

Камуфлирование программного обеспечения (ПО). Другой важной задачей стеганографии является камуфлирование ПО. В тех случаях, когда использование ПО незарегистрированными пользователями является нежелательным, оно может быть закамуфлировано под стандартные универсальные программные продукты (например, текстовые редакторы) или скрыто в файлах мультимедиа (например, в звуковом сопровождении компьютерных игр).

Защита авторских прав. Еще одной областью использования стеганографии является защита авторского права от пиратства. На компьютерные графические изображения наносится специальная метка, которая остается невидимой для глаз, но распознается специальным ПО. Такое программное обеспечение уже используется в компьютерных версиях некоторых журналов. Данное направление стеганографии предназначено не только для обработки изображений, но и для файлов с аудио- и

видеоинформацией и призвано обеспечить защиту интеллектуальной собственности.

В настоящее время методы компьютерной стеганографии развиваются по двум основным направлениям:

1. Методы, основанные на использовании специальных свойств компьютерных форматов;
2. Методы, основанные на избыточности аудио и визуальной информации.

Первое направление основано на использовании специальных свойств компьютерных форматов представления данных, а не на избыточности самих данных. Специальные свойства форматов выбираются с учетом защиты скрываемого сообщения от непосредственного прослушивания, просмотра или прочтения. На основании анализа материалов табл. 1 можно сделать вывод, что основным направлением компьютерной стеганографии является использование избыточности аудио и визуальной информации. Цифровые фотографии, цифровая музыка, цифровое видео — представляются матрицами чисел, которые кодируют интенсивность в дискретные моменты в пространстве и/или во времени. Цифровая фотография — это матрица чисел, представляющих интенсивность света в определенный момент времени. Цифровой звук — это матрица чисел, представляющая интенсивность звукового сигнала в последовательно идущие моменты времени. Все эти числа не точны, т.к. не точны устройства оцифровки аналоговых сигналов, имеются шумы квантования. Младшие разряды цифровых отсчетов содержат очень мало полезной информации о текущих параметрах звука и визуального образа. Их заполнение ощутимо не влияет на качество восприятия, что и дает возможность для скрытия дополнительной информации.

Графические цветные файлы со схемой смешения RGB кодируют каждую точку рисунка тремя байтами. Каждая такая точка состоит из аддитивных составляющих: красного, зеленого, синего. Изменение каждого из трех наименее значимых бит приводит к изменению менее 1% интенсивности данной точки. Это позволяет скрывать в стандартной графической картинке

объемом 800 Кбайт около 100 Кбайт информации, что не заметно при просмотре изображения.

Другой пример. Только одна секунда оцифрованного звука с частотой дискретизации 44100 Гц и уровнем отсчета 8 бит в стерео режиме позволяет скрыть за счет замены наименее значимых младших разрядов на скрываемое сообщение около 10 Кбайт информации. При этом изменение значений отсчетов составляет менее 1%. Такое изменение практически не обнаруживается при прослушивании файла большинством людей.

Анализируя тенденций развития КС можно предположить, что в ближайшие годы интерес к развитию методов КС будет усиливаться всё больше и больше. Предпосылки к этому уже формируются сегодня. В частности, общеизвестно, что актуальность проблемы информационной безопасности постоянно растет и стимулирует поиск новых методов защиты информации (ЗИ). С другой стороны, бурное развитие информационных технологий обеспечивает возможность реализации этих новых методов ЗИ. И конечно сильным катализатором этого процесса является лавинообразное развитие компьютерной сети общего пользования Internet, в том числе такие нерешенные противоречивые проблемы Internet, как защита авторского права, защита прав на личную тайну, организация электронной торговли, противоправная деятельность хакеров, террористов и т.п.

Весьма характерной тенденцией в настоящее время в области ЗИ является внедрение криптологических методов. Однако на этом пути много ещё нерешенных проблем, связанных с разрушительным воздействием на криптосредства таких составляющих информационного оружия как компьютерные вирусы, логические бомбы, автономные репликативные программы и т.п. Объединение методов компьютерной стеганографии и криптографии явилось бы хорошим выходом из создавшегося положения. В этом случае удалось бы устранить слабые стороны известных методов защиты информации и разработать более эффективные новые нетрадиционные методы обеспечения информационной безопасности.

Тема моей диссертации: использование информационной избыточности файлов для передачи и хранения скрытой информации. Данная тема была выбрана в связи с тем что ситуация вокруг стеганографического программного обеспечения складывается таким образом, что ввиду своей сложности программной реализации, имеется мало программ с такими возможностями и зачастую это попросту мошенничество. А литературы с описанием методов и вовсе нет.

Цель диссертации: найти возможность хранения и передачи скрытых сообщений в файлах «контейнерах», без изменения размера самого файла. Исходя из этого определил задачи необходимые для достижения цели:

1. рассмотреть форматы файлов: текстовые (txt, doc, rtf), графические (bmp, jpeg, gif, tif), мультимедиа (avi, mpeg, mp3);
2. рассмотреть возможность использования младших битов для хранения и передачи скрытых сообщений;
3. разработка алгоритмов и программного обеспечения для криптографической защиты информации;
4. разработка алгоритмов и программного обеспечения для стеганографической передачи и хранения информации.

Прежде чем пытаться что-то где-то спрятать необходимо найти где это будет уместно и не вызовет явного проявления начинки нашего пирога. Для этого рассмотрим форматы хранения данных нашего секрета.

2. ФАЙЛОВЫЕ СТРУКТУРЫ ПРЕДПОЛАГАЕМОЙ ЭКСПАНСИИ

В данном разделе львиная доля будет отведена графическим файловым структурам. Ввиду того, что они явно доминируют над другими форматами основываясь даже на общих представлениях о структуре интересующих нас файлов. И так в путь по просторам файловых структур.

2.1. Текстовые файлы

Представление текстовых данных: все используемые способы представления текстовых данных, так или иначе, сводятся к нумерации символов алфавита (или множества символов системы письменности интересующего нас языка, которое используется вместо алфавита — слоговой азбуки, иероглифов и т. д.) и хранения полученных целых чисел наравне с обычными числами. Способ нумерации называется *кодировкой*, а числа — *кодами символов*.

Для большинства кодировок языков, использующих алфавитную письменность (латиница, кириллица, арабский алфавит) достаточно 127 символов. Самая распространенная система кодирования латиницы — ASCII — использует 7 бит на символ. Другие алфавиты обычно кодируются более сложным образом: символы алфавита получают коды в диапазоне от 128 до 255, а коды от 0 до 127 соответствуют кодам ASCII. Таким образом, любой символ этих алфавитов, в том числе и в многоязычных текстах, использующих сочетание национального алфавита и латиницы, может быть представлен 8-ю битами или одним байтом. Но для японских слоговых азбук, а тем более для китайской иероглифики, 255 кодов явно недостаточно, и приходится использовать многобайтовые кодировки. Распространенное обозначение таких кодировок — DBCS — *набор символов, кодируемый двумя байтами.*

Используются две основные кодировки латиницы -- ASCII и EBCDIC. Для представления русского варианта кириллицы существует три основных кодировки: альтернативная (известная также как cp866) и KOI-8 и ряд менее широко используемых (ISO 8892-5 и др.).

Арифметические операции над такими "числами" обычно бессмысленны, зато большой смысл имеют операции сравнения. Под текстовым файлом мы будем понимать любой файл содержащий текст. С начала стоит разбить все файловые структуры на две категории:

1. неформатированный текст;
2. форматированный текст.

Неформатированный текст: txt-формат, состоит из одного блока (Рис.1). Текст такой файловой структуры не содержит ни чего кроме текста. В строках отсутствует дополнительная информация о виде отображаемых символов. Начало информационного блока не обозначено и начинается непосредственно с первой строки текста. Конец строки обозначен специальным символом, точнее двумя: #10-символ конца строки, #13-символ перехода на новую строку для текстовых редакторов (блокнот, DOS- copy con [filename]), затем цикл повторяется. Последняя строка файла несет символ конца файла (#1#13) автоматически выставляемого текстовыми редакторами, а в случаи DOS вручную выставляется Ctrl+Z.

Структура текстового файла txt-формата

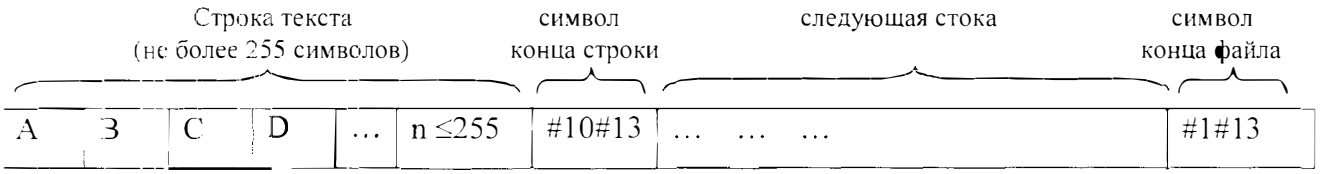


Рисунок 2

Проведя исследование этого типа файловой структуры, можно сказать, что данная структура с лихвой нагружена смыслом. Единственный вариант использования этого формата в стеганографии так это замена символов русского алфавита на аналогичные по написанию символы латиницы (таблица 1). Обратный процесс происходит путем поиска латинских букв. Однако здесь есть ограничения по словообразованию. Многие из символов не являются аналогами и появление их в тексте не желательно.

Таблица 1

Список букв имеющих идентичное написание для кириллицы и латиницы.

A	A	B	E	e	K	к	M	M	H	O	o	P	C	c	T	X	X

Экспериментальным путем выявил, что такой способ способен поглотить 2-3% от общего объема текста. А такой результат не может быть допустимым для передачи сообщений. И более того, читать полученный текст не доставляет большого удовольствия из-за отсутствия в нем пробелов, которые выставляются по смыслу. И так оставим в покое неформатированный текст, мало удовлетворяющий нашим запросам.

Форматированный текст: структура файлов блочная (Рис.3). Существует блок сопоставления целостности содержимого файла, блок содержимого файла, блок интегрированных картинок и блок хранения паролей и атрибутов защиты. Отличительной особенностью от неформатированных файловых структур файлы такого типа являются возможность изменения атрибутов текста (изменение размера, цвета, вида, наклон, подчеркивание выделение жирным), а также вставка графических объектов. На рисунке представлена общая структура файлов, с вынесенным блоком хранения текстовой информации. Проведя анализ данной структуры, пришел к выводу: любое изменение служебной информации приводит к повреждению целостности файла. Воздействие на блок текстовой информации не приносит разрушительного воздействия. Применяя методику, апробированную на неформатированном тексте можно провести аналогичные превращения. Однако, имея подобный опыт, можно с уверенностью сказать что процент в 2-3% остается и в этом формате. Но если для неформатированного текста применяется текстовые редакторы класса «Блокнот», то для форматированного текста применяются, как правило, аналоги «MS Word». А вот именно из-за этого возникают дополнительные проблемы сокрытия сообщения. Так как в эти редакторы встроены экспертные системы автоматической проверки орфографии, то явным образом видны слова с измененным текстом путем применения подмены символов. Такие слова отображаются с красным подчеркиванием волнистой линией (~~~~). Такая красота обязательно натолкнет на раздумья о природе ошибок в словах текста. Следовательно,

использование этого формата как контейнер хранения скрытых сообщений еще менс удобно по сравнению с неформатированным текстом.

Структура текстового файла doc/rtf-формата

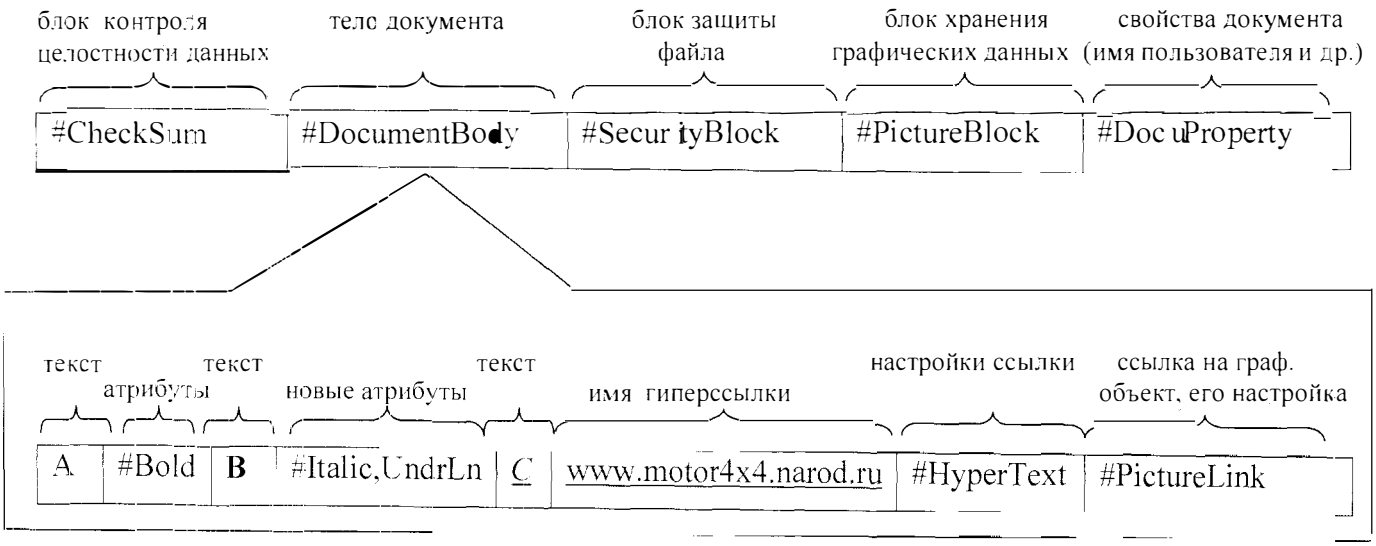


Рисунок 3

Подведя итоги по текстовым форматам файлов можно сказать что, их использование не отвечает требованиям, предъявляемым к файлам контейнерам. К примеру, средний размер сообщения способного нести в себе значимую информацию составляет 1-1,5мБ. Следовательно, размер контейнера в 50-60мБ неформатированного текста достаточно велик для того, чтобы пользоваться им как контейнером. Ведь такой объем велик для регулярного пользования, хотя и не исключает возможного использования. Что касается форматированного текста, то процент упаковки текста в контейнер остается тот же 2-3%. Но не стоит забывать то что, к примеру, пустой файл в формате MS Word имеет размер в 16кБ. Применив еще несколько колонтитулов и ссылок размер имеет тенденцию к существенному увеличению. И те самые 1-1,5мБ мы сможем поместить не в 50-60мБ, а в 100-300мБ. Что явно принесет неудобства в наших чудесных превращениях. Однако какой, никакой, но это тоже результат. И не будем списывать его со счетов, так как 2-3% я все-таки извлек. Буду

надеяться на то, что это не самые лучшие показатели среди испытуемых на нашем полигоне. Теперь на очереди картинки.

2.2. Растровая графика

Начну, пожалуй, с того, что скажу сразу: единственный формат, с которым работают компьютерные просмотрщики это BitMap files (файл битовой карты изображения). Все остальные это лишь сжатые при помощи математических алгоритмов битовой карты изображения.

Алгоритмы сжатия изображений — бурно развивающаяся область машинной графики. Основной объект приложения усилий в ней — изображения — своеобразный тип данных, характеризуемый тремя особенностями:

1) Изображения (как и видео) *обычно требует для хранения гораздо большего объема памяти, чем текст.* Так, скромная, не очень качественная иллюстрация на обложке книги размером 500x800 точек, занимает 1.2 Мб— столько же, сколько художественная книга из 400 страниц (60 знаков в строке, 42 строки на странице). В качестве примера можно рассмотреть также, сколько тысяч страниц текста мы сможем поместить на CD-ROM, и как мало там поместится не сжатых фотографий высокого качества. Эта особенность изображений определяет актуальность алгоритмов архивации графики.
2) Второй особенностью изображений является то, что человеческое зрение при анализе изображения оперирует контурами, общим переходом цветов и сравнительно нечувствительно к малым изменениям в изображении. Таким образом, мы можем создать эффективные алгоритмы архивации изображений, в которых декомпрессированное изображение не будет совпадать с оригиналом, однако человек этого не заметит. Данная особенность человеческого зрения позволила создать специальные алгоритмы сжатия, ориентированные только на изображения. Эти алгоритмы позволяют сжимать изображения с высокой степенью сжатия и незначительными с точки зрения человека потерями.
3) Мы можем легко заметить, что изображение, в отличие, например, от текста, обладает избыточностью в 2-х измерениях. То есть, как правило, соседние

точки, как по горизонтали, так и по вертикали, в изображении близки по цвету. Кроме того, мы можем использовать подобие между цветовыми плоскостями R, G и B в наших алгоритмах, что дает возможность создать еще более эффективные алгоритмы. Таким образом, при создании алгоритма компрессии графики мы используем особенности структуры изображения.

Всего на данный момент известно минимум три семейства алгоритмов, которые разработаны исключительно для сжатия изображений, и применяемые в них методы практически невозможно применить к архивации еще каких-либо видов данных. Для того чтобы говорить об алгоритмах сжатия изображений, мы должны определиться с несколькими важными вопросами:

1) Какие критерии мы можем предложить для сравнения раз личных алгоритмов?
2) Какие классы изображений существуют?
3) Какие классы приложений, использующих алгоритмы компрессии графики, существуют, и какие требования они предъявляют к алгоритмам?

Рассмотрим эти вопросы подробнее.

2.2.1. Классы изображений

Статические растровые изображения представляют собой двумерный массив чисел. Элементы этого массива называют *пикселами* (от английского «pixel» — «picture element»). Все изображения можно подразделить на две группы — с палитрой и без нее. У изображений с палитрой в пикселе хранится число — индекс в некотором одномерном векторе цветов, называемом *палитрой*. Чаще всего встречаются палитры из 16 и 24 битной палитры цвета.

Изображения без палитры бывают в какой-либо системе цветопредставления и *в градациях серого* (grayscale). Для последних значение каждого пиксела интерпретируется как яркость соответствующей точки. Чаще всего встречаются изображения с 2, 16 и 256 уровнями серого. Одна из интересных практических задач заключается в приведении цветного или черно-

белого изображения к двум градациям яркости, например, для печати на лазерном принтере. При использовании некой *системы цвето-представления* каждый пиксел представляет собой запись (структуру), полями которой являются компоненты цвета. Самой распространенной является *система RGB*, в которой цвет передается значениями интенсивности красной (R), зеленой (G) и синей (B) компонент. Существуют и другие системы цвето-представления, такие, как CMYK, CIE XYZccir60-l, YVU, YCrCb и т.п.

Для того чтобы корректнее оценивать степень сжатия, нужно ввести понятие *класса изображений*. Под классом будет пониматься совокупность изображений, применение к которым алгоритма архивации дает качественно одинаковые результаты. Например, для одного класса алгоритм дает очень высокую степень сжатия, для другого — почти не сжимает, для третьего — увеличивает файл в размере. (Например, все алгоритмы сжатия без потерь в худшем случае увеличивают файл.)

Дадим неформальное определение наиболее распространенных классов изображений:

1) Класс 1. Изображения с небольшим количеством цветов (4-16) и большими областями, заполненными одним цветом. Плавные переходы цветов отсутствуют. Примеры: деловая графика — гистограммы, диаграммы, графики и т.п.
2) Класс 2. Изображения с плавными переходами цветов, построенные на компьютере. Примеры: графика презентаций, эскизные модели в САПР, изображения, построенные по методу Гуро.
3) Класс 3. Фотореалистичные изображения. Пример: отсканированные фотографии.
4) Класс 4. Фотореалистичные изображения с наложением деловой графики.

Пример: реклама. Развивая данную классификацию, в качестве отдельных классов могут быть предложены некачественно отсканированные в 256 градаций серого цвета страницы книг или растровые изображения топографических карт. (Заметим, что этот класс не тождественен классу 4).

Формально являясь 8- или 24-битными, они несут не растровую, а чисто векторную информацию. Отдельные классы могут образовывать и совсем специфичные изображения: рентгеновские снимки или фотографии в профиль и фас из электронного досье.

Достаточно сложной и интересной задачей является поиск наилучшего алгоритма для конкретного класса изображений.

2.2.2. Формат BMP

Эта статья про то, как выглядит графический формат bmp. Хоть это и один из простых форматов, но в из-за того, что существует много вариаций этого формата, то не все моменты очевидны. Итак, хватит лить воду, начнем.

2.2.2.1Структуры формата

Формат bmp (от слов BitMaP - битовая карта, или, говоря по-русски, битовый массив) представляет из себя несжатое (в основном) изображение, которое довольно легко читается и выводится в ОС Windows, в которой есть специальные функции API, которые в этом помогают.

Для начала приведем графическое представление данных в bmp (Рис 4).

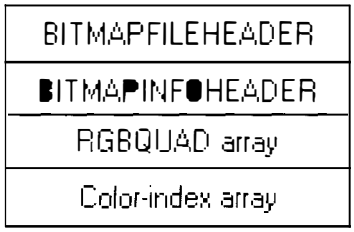


Рисунок 4

В начале стоит заголовок файла (BITMAPFILEHEADER). Он описан следующим образом:

```
typedef struct tagBITMAPFILEHEADER
{
 WORD    bfType;
 DWORD   bfSize;
 WORD    bfReserved1;
```

```
 WORD   bfReserved2;
 DWORD  bfOffBits;
} BITMAPFILEHEADER, *PBITMAPFILEHEADER;
```

bfType определяет тип файла. Здесь он должен быть BM. Если Вы откроете любой файл BMP в текстовом (а лучше в 16-ричном редакторе), то увидите, что первые два символа - это BM (от слова BitMap, как вы уже, наверное, догадались).

bfSize - это размер самого файла в байтах. Строго говоря вы должны его высчитывать (что рекомендуется), но я ставил размер файла неправильно (правда, ненарошно :)) и никаких проблем не было (ACDSee читало без проблем, моя программа работала), но я вам не рекомендую писать его заведомо неправильно, вдруг появится добросовестная программа, которая сверит этот размер с настоящим и рашит, что это не bmp, а что-нибудь другое. В идеале все программы для того, чтобы убедиться, что перед ними действительно bmp, а не подделка, должны, во-первых, проверить, что bfType содержит "BM" (без кавычек), а, во-вторых, что bfSize равен размеру файла.

bfReserved1 u bfReserved2 зарезервированы и должны быть нулями.

bfOffBits. Это один из самых важных полей в этой структуре. Он показывает, где начинается сам битовый массив относительно начала файла (или, как написано в MSDN, "от начала структуры BITMAPFILEHEADER"), который и описывает картинку. То есть, чтобы горантированно попадать на начало массива вы должны писать: SetFilePointer (hFile, bfh.bfOffBits, NULL, FILE_BEGIN);

Здесь и далее будем считать, что переменная bfh объявлена как BITMAPFILEHEADER bfh;

А дальше идет структура BITMAPINFOHEADER, которая объявлена так: typedef struct tagBITMAPINFOHEADER

```
{
 DWORD  biSize;
 LONG   biWidth;
 LONG   biHeight;
 WORD   biPlanes;
```

```
  WORD  biBitCount;
  DWORD  biCompression;
  DWORD  biSizeImage;
  LONG  biXPelsPerMeter;
  LONG  biYPelsPerMeter;
  DWORD  biClrUsed;
  DWORD  biClrImportant;
} BITMAPINFOHEADER, *PBITMAPINFOHEADER;
```

biSize - это размер самой структуры. Ее нужно инициализировать следующим образом: bih.biSize = sizeof (BITMAPINFOHEADER); Снова здесь и дальше будем считать, что bih объявлена следующим образом: BITMAPINFOHEADER bih;

biWidth и biHeight задают соответственно ширину и высоту картинки в пикселях.

biPlanes задает количество плоскостей. Пока оно всегда устанавливается в 1.

biBitCount - Количество бит на один пиксель. Подробнее про это поговорим ниже.

biCompression обозначает тип сжатия. Не удивляйтесь и не пугайтесь, что в bmp и вдруг сжатие. Я лично не видел не одной сжатой bmp (но я не говорю, что таких не существует). Если сжатия нет, то этот флаг надо устанавливать в BI_RGB. В этой статье мы говорим про несжатый формат, поэтому другие флаги я даже не буду перечислять. Похоже, что эта же структура используется и в файлах JPEG и PNG, потому что, начиная с Windows 98 тут появились варианты BI_JPEG, которая показывает, что эта картинка - JPEG и BI_PNG, что это PNG (про формат Jpeg я ничего не знаю, я только сделал эти выводы исходя из того, что написано в MSDN).

biSizeImage обозначает размер картинки в байтах. Если изображение несжато (то есть предыдущее поле установлено в BI_RGB), то здесь должен быть записан ноль. *biXPelsPerMeter* и *biYPelsPerMeter* обозначают

соответственно горизонтальное и вертикальное разрешение (в пикселях на метр) конечного устройства, на которое будет выводиться битовый массив (растр). Приложение может использовать это значение для того, чтобы выбирать из группы ресурсов наиболее подходящий битовый массив для нужного устройства. Дело в том, что формат bmp - это по сути аппаратно-независимый растр, то есть когда внешний вид того, что получается не зависит от того, на что этот растр проецируется (если можно так выразится). Например, картинка будет выглядеть одинаково вне зависимости от того, рисуется она на экране монитора или печатается на принтере. Но вот разрешение у устройств разное, и именно для того, чтобы выбрать наиболее подходящую картинку из имеющихся и используют эти параметры.

biClrUsed определяет количество используемых цветов из таблицы. Если это значение равно нулю, то в растре используется максимально возможное количество цветов, которые разрешены значением biBitCount. Это актуально только для сжатых картинок. Если biClrUsed не нуль и biBitCount меньше 16, то biClrUsed определяет текущее число цветов графического движка или доступного драйвера устройства. Если biBitCount больше или равно 16, то biClrUsed определяет размер таблицы цветов, используемой для оптимизации текущей системной палитры.

biClrImportant - это количество важных цветов. Определяет число цветов, которые необходимы для того, чтобы изобразить рисунок. Если это значение равно 0 (как это обычно и бывает), то все цвета считаются важными.

2.2.2.2. Виды формата bmp

Все разновидности формата bmp условно можно разделить на два типа: палитровые и беспалитровые (Таблица 2). То есть используется в данном сформате палитра или нет. Заметьте, что палитра может быть даже в беспалитровых форматах, только там она не используется. В беспалитровых bmp цвет высчитывается прямо из тех битов, которые идут в файле, начиная с некоторого места. А в палитровых каждый байт описывает один или несколько

пикселей, причем значения байта (или битов) - это индекс цвета в палитре. Для начала приведу таблицу, которая сравнивает возможные варианты. Вид картинки (палитровая или беспалитровая) зависит от того, сколько бит отдается на один пиксель, то есть от значения biBitCount структуры BITMAPINFOHEADER.

Таблица 2

Справочные данные на форматы BitMap

biBitCount	**Палитровый или беспалитровый формат**	**Максимально возможное количество цветов**	**Примечания**
1	Палитровый	2	Двуцветная, заметьте, не обязательно черно-белая, палитровая картинка. Если бит растра (что это такое чуть ниже) сброшен (равен 0), то это значит, что на этом месте должен быть первый цвет из палитры, а, если установлен (равен 1), то второй.
4	Палитровый	16	Каждый байт описывает 2 пикселя. Вот пример из MSDN.Если первый байт в картинке 0x1F, то она соответствует двум пикселям, цвет первого - второй цвет из палитры (потому что отсчет идут от нуля), а второй пиксель - 16-й цвет палитры.
8	Палитровый	256	Один из самых распространенных вариантов. Но в то же время и самых простых. Палитра занимает один килобайт (но на это лучше не рассчитывать). Один байт - это один цвет. Причем его значение - это номер цвета в палитре.
16	Беспалитровый	2^16 или 2^15	Это самый запутанный вариант. Начнем с того, что он беспалитровый, то есть

			каждые два байта (одно слово WORD) в растре однозначно определяют один пиксель. Но вот что получается: битов-то 16, а компонентов цветов - 3 (Красный, Зеленый, Синий). А 16 никак на 3 делиться не хочет. Поэтому здесь есть две варианта. Первый - использовать не 16, а 15 битов, тогда на каждую компоненту цвета выходит по 5 бит. Таким образом мы можем использовать максимум 2^15 = 32768 цветов и получается тройка R-G-B = 5-5-5. Но тогда за зря теряется целый бит из 16. Но так уж случилось, что наши глаза среди всех цветов лучше воспринимают зеленый цвет, поэтому и решили этот один бит отдавать на зеленую компоненту, то есть тогда получается тройка R-G-B = 5-6-5, и теперь мы может использовать 2^16 = 65536 цветов. Но что самое неприятное, что используют оба варианта. В MSDN предлагают для того, чтобы различать сколько же цветов используется, заполнять этим значением поле biClrUsed из структуры BITMAPINFOHEADER. Чтобы выделить каждую компоненту надо использовать следующие маски. Для формата 5-5-5: 0x001F для синей компоненты, 0x03E0 для зеленой и 0x7C00 для красной. Для формата 5-6-5: 0x001F - синяя, 0x07E0 - зеленая и 0xF800 красная компоненты соответственно.
24	Беспалитровый	2^24	А это самый простой формат. Здесь 3 байта определяют 3 компоненты цвета. То есть по компоненте на байт. Просто читаем по структуре RGBTRIPLE и используем его поля rgbtBlue, rgbtGreen, rgbtRed. Они идут

			именно в таком порядке.
32	Беспалитровый	2^32	Здесь 4 байта определяют 3 компоненты. Но, правда, один байт не используется. Его можно отдать, например, для альфа-канала (прозрачности). Читать растр в данном случае удобно структурами RGBQUAD, которая описана так: typedef struct tagRGBQUAD { BYTE rgbBlue; BYTE rgbGreen; BYTE rgbRed; BYTE rgbReserved; } RGBQUAD;

2.2.2.3. Хранение данных в формате BMP

Вот и подошли к самому интересному. После структур BITMAPFILEHEADER и BITMAPINFOHEADER идет палитра. Причем, если формат беспалитровый, то ее может и не быть, однако, на это рассчитывать не надо. Дело в том, что, когда я только начинал разбираться с форматом bmp, в одной книжке я вычитал, что, якобы, если формат беспалитровый, то у нее вообще нет палитры. Там даже были две картинки - схемы формата: одна с палитрой, другая без. А я в это время писал программу, которая усердно оперирует с bmp-шками. И мне надо было преобразовывать входящие картинки из 256 цветов в 24-битные (если таковые имелись) во временные файлы. И я в 24-битных палитру просто не создавал (bfOffBits из структуры BITMAPFILEHEADER у меня был равен сумме sizeof(BITMAPFILEHEADER) + sizeof (BITMAPINFOHEADER), а входящие 24-разрядные оставлял без изменений. С 256-цветными растрами все работало как надо, пока мне не попалась 24-разрядная картинка, у которой внизу вместо нужной части отобразался мусор. Я не сразу понял в чем дело. Пока не сравнил размер исходного файла с теоретическим, который должен был быть, не будь палитры.

Разница оказалась ровно 1 Кб (ровно 1024 байта). Там была палитра. Поэтому никогда не рассчитывайте на то, есть ли палитра и не надейтесь на ее размер (хотя все картинки, которые мне попадались имели размер палитры 256 цветов, или 1Кб), всегда перемещайтесь по файлу на начало растра, используя bfOffBits. Палитра представляет из себя массив структур RGBQUAD идущих друг за другом. Даже если в палитре используются не все цвета (а только, например, 16), то часто все равно под палитру отводят 256 полей. А 256 * 4 = 1024, где 4 - размер структуры RGBQUAD, то есть и получается тот самый один килобайт.

Сразу за палитрой идет сам растр. Тут уже более запутано. Во-первых, пиксели тут описываются так, как написано в таблице выше в зависимости от формата. И могут сами содержать значение компонентов цвета (для беспалитровых), а могут быть индексами массива-палитры. Сама картинка записывается построчно. Во-вторых, картинка идет как бы перевернутая вверх ногами. То есть сначала записана нижняя строка, потом предпоследняя и так далее до самого верха. И, в-третьих, если размер строки растра не кратен 4, то она дополняется от 1 до 3 пустыми (нулевыми) байтами, чтобы длина строки оказалась кратна параграфу. Вот это и есть самое неприятное. Дело в том, что для каждого формата приходится подстраивать это число пустых байтов (правда, я люблю туда записывать часть палитры, просто мне не хочется заводить лишние "нулевые" переменные, если все равно эти байты пропускают и никому они не нужны). Я привожу таблицу с формулами (Таблица 3), которые показывают для какого формата сколько байт надо дописывать в конец строки. Там под переменной Width, как можно догадаться, подразумевается ширина картинки. Все эти формулы были установлены экспериментально. Я приведу пример только для наиболее используемых форматов.

Таблица 3

Соответствие байт формату

biBitCount	Формула
8	(3 * Width) % 4
16	(2 * Width) % 4
24	Width % 4

2.2.3. Классы приложений

Примеры приложений, использующих алгоритмы компрессии графики. Рассмотрим следующую простую классификацию приложений, использующих алгоритмы компрессии:

Класс 1. Характеризуются высокими требованиями ко времени архивации и разархивации. Нередко требуется просмотр уменьшенной копии изображения и поиск в базе данных изображений. *Примеры:* Издательские системы в широком смысле этого слова, причем как готовящие качественные публикации (журналы) с заведомо высоким качеством изображений и использованием алгоритмов архивации без потерь, так и готовящие газеты, и WWW-сервера где есть возможность оперировать изображениями меньшего качества и использовать алгоритмы сжатия с потерями. В подобных системах приходится иметь дело с полноцветными изображениями самого разного размера (от 640x480 — формат цифрового фотоаппарата, до 3000x2000) и с большими двуцветными изображениями. Поскольку иллюстрации занимают львиную долю от общего объема материала в документе, проблема хранения стоит очень остро. Проблемы также создает большая разнородность иллюстраций (приходится использовать универсальные алгоритмы). Единственное, что можно сказать заранее, это то, что будут преобладать фотореалистичные изображения и деловая графика.

Класс 2. Характеризуется высокими требованиями к степени архивации и времени разархивации. Время архивации роли не играет. Иногда подобные приложения также требуют от алгоритма компрессии легкости

масштабирования изображения под конкретное разрешение монитора у пользователя. *Пример:* Справочники и энциклопедии на CD-ROM. С появлением большого количества компьютеров, оснащенных этим приводом (в США уровень в 50% машин достигнут еще в 1995), достаточно быстро сформировался рынок программ, выпускаемых на лазерных дисках. Несмотря на то, что емкость одного диска довольно велика (примерно 650 Мб), ее, как правило, не хватает. При создании энциклопедий и игр большую часть диска занимают статические изображения и видео. Таким образом, для этого класса приложений актуальность приобретают существенно асимметричные по времени алгоритмы *(симметричность по времени* — отношение времени компрессии ко времени декомпрессии).

Класс 3. Характеризуется очень высокими требованиями к степени архивации. Приложение клиента получает от сервера информацию по сети. *Пример:* Новая быстро развивающаяся система «Всемирная информационная паутина» — WWW. В этой гипертекстовой системе достаточно активно используются иллюстрации. При оформлении информационных или рекламных страниц хочется сделать их более яркими и красочными, что естественно сказывается на размере изображений. Больше всего при этом страдают пользователи, подключенные к сети с помощью медленных каналов связи. Если страница WWW перенасыщена графикой, то ожидание ее полного появления на экране может затянуться. Поскольку при этом нагрузка на процессор мала, то здесь могут найти применение эффективно сжимающие сложные алгоритмы со сравнительно большим временем разархивации. Кроме того, мы можем видоизменить алгоритм и формат данных так, чтобы просматривать огрубленное изображение файла до его полного получения. Можно привести множество более узких классов приложений. Так, свое применение машинная графика находит и в различных информационных системах. Например, уже становится привычным исследовать ультразвуковые и рентгеновские снимки не на бумаге, а на экране монитора. Постепенно в электронный вид переводят и истории болезней. Понятно, что хранить эти

материалы логичнее в единой картотеке. При этом без использования специальных алгоритмов большую часть архивов займут фотографии. *Поэтому при создании эффективных алгоритмов решения этой задачи нужно учесть специфику рентгеновских снимков — преобладание размытых участков.*

В геоинформационных системах — при хранении аэрофотоснимков местности специфическими проблемами являются большой размер изображения и *необходимость выборки лишь части изображения по требованию.* Кроме того, может потребоваться *масштабирование.* Это неизбежно накладывает свои ограничения на алгоритм компрессии.

В электронных картотеках и досье различных служб для изображений характерно подобие между фотографиями в профиль и подобие между фотографиями в фас, которое также необходимо учитывать при создании алгоритма архивации. *Подобие между фотографиями* наблюдается и в любых других специализированных справочниках. В качестве примера можно привести энциклопедии птиц или цветов.

2.2.4. Требования приложений к алгоритмам компрессии

В предыдущем разделе мы определили, какие приложения являются потребителями алгоритмов архивации изображений. Однако заметим, что приложение определяет характер использования изображений (либо большое количество изображений хранится и используется, либо изображения скачиваются по сети, либо изображения велики по размерам, и нам необходима возможность получения лишь части...). Характер использования изображений задает степень важности следующих ниже противоречивых требований к алгоритму:

1. *Высокая степень компрессии*. Заметим, что далеко не для всех приложений актуальна высокая степень компрессии. Кроме того, некоторые алгоритмы дают лучшее соотношение качества к размеру файла при высоких степенях компрессии, однако проигрывают другим алгоритмам при низких степенях..

2. *Высокое качество изображений.* Выполнение этого требования напрямую противоречит выполнению предыдущего... *Высокая скорость компрессии.* Это требование для некоторых алгоритмов с потерей информации является взаимоисключающим с первыми двумя. Интуитивно понятно, что чем больше времени мы будем анализировать изображение, пытаясь получить наивысшую степень компрессии, тем лучше будет результат. И, соответственно, чем меньше мы времени потратим на компрессию (анализ), тем ниже будет качество изображения и больше его размер.
3. *Высокая скорость декомпрессии.* Достаточно универсальное требование, актуальное для многих приложений. Однако можно привести примеры приложений, где время декомпрессии далеко не критично.
4. *Масштабирование изображений.* Данное требование подразумевает легкость изменения размеров изображения до размеров окна активного приложения. Дело в том, что одни алгоритмы позволяют легко масштабировать изображение прямо во время декомпрессии, в то время как другие не только не позволяют легко масштабировать, но и увеличивают вероятность появления неприятных артефактов после применения стандартных алгоритмов масштабирования к декомпрессированному изображению. Например, можно привести пример «плохого» изображения для алгоритма JPEG — это изображение с достаточно мелким регулярным рисунком (пиджак в мелкую клетку). Характер вносимых алгоритмом JPEG искажений таков, что уменьшение или увеличение изображения может дать неприятные эффекты.
5. *Возможность показать огрубленное изображение (низкого разрешения), использовав только начало файла.* Данная возможность актуальна для различного рода сетевых приложений, где перекачивание изображений может занять достаточно большое время, и желательно, получив начало файла, корректно показать preview. Заметим, что примитивная реализация указанного требования путем записывания в начало изображения его уменьшенной копии заметно ухудшит степень компрессии.

6. *Устойчивость к ошибкам.* Данное требование означает локальность нарушений в изображении при порче или потере фрагмента передаваемого файла. Данная возможность используется при широковещании (broadcasting - передача по многим адресам) изображений по сети, то есть в тех случаях, когда невозможно использовать протокол передачи, повторно запрашивающий данные у сервера при ошибках. Например, если передается видеоряд, то было бы неправильно использовать алгоритм, у которого сбой приводил бы к прекращению правильного показа всех последующих кадров. Данное требование противоречит высокой степени архивации, поскольку интуитивно понятно, что мы должны вводить в поток избыточную информацию. Однако для разных алгоритмов объем этой избыточной информации может существенно отличаться.
7. *Учет специфики изображения.* Более высокая степень архивации для класса изображений, которые статистически чаще будут применяться в нашем приложении. В предыдущих раз делах это требование уже обсуждалось.
8. *Редактируемость.* Под редактируемостью понимается минимальная степень ухудшения качества изображения при его повторном сохранении после редактирования. Многие алгоритмы с потерей информации могут существенно испортить изображение за несколько итераций редактирования.
9. *Небольшая стоимость аппаратной реализации. Эффективность программной реализации.* Данные требования к алгоритму реально предъявляют не только производители игровых приставок, но и производители многих информационных систем. Так, декомпрессор фрактального алгоритма очень эффективно и коротко реализуется с использованием технологии MMX и распараллеливания вычислений, а сжатие по стандарту CCITT Group 3 легко реализуется аппаратно. Очевидно, что для конкретной задачи нам будут очень важны одни требования и менее важны (и даже абсолютно безразличны) другие.

2.2.5. Критерии сравнения алгоритмов

Заметим, что характеристики алгоритма относительно некоторых требований приложений, сформулированные выше, зависят от конкретных условий, в которые будет поставлен алгоритм. Так, *степень компрессии* зависит от того, на каком классе изображений алгоритм тестируется. Аналогично, *скорость компрессии* нередко зависит от того, на какой платформе реализован алгоритм. Преимущество одному алгоритму перед другим может дать, например, возможность использования в вычислениях алгоритма технологий нижнего уровня, типа MMX, а это возможно далеко не для всех алгоритмов. Так, JPEG существенно выигрывает от применения технологии MMX, а LZW нет. Кроме того, нам придется учитывать, что некоторые алгоритмы распараллеливаются легко, а некоторые 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, интерес к показу огрубленного изображения, используя только начало файла, был чисто абстрактным. Реально эта возможность практически нигде не требовалась и класс приложений, использующих данную технологию, был крайне невелик. С взрывным распространением Internet, 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.

В то же время мы можем рассмотреть такое редкое на сегодня требование, как устойчивость к ошибкам. Можно предположить, что в скором времени (через 5-10 лет) с распространением широковещания в сети Internet для его обеспечения будут использоваться именно алгоритмы, устойчивые к ошибкам, даже не рассматриваемые в сегодняшних статьях и обзорах.

Со всеми сделанными выше оговорками, выделим несколько наиболее важных для нас критериев сравнения алгоритмов компрессии, которые и будем использовать в дальнейшем. Как легко заметить, мы будем обсуждать меньше критериев, чем было сформулировано выше. Это позволит избежать лишних деталей при кратком изложении данного материала.

1. *Худшая, средняя и лучшая степень сжатия*. То есть доля, на которую возрастет изображение, если исходные данные будут наихудшими; некая среднестатистическая степень для того класса изображений, *на который ориентирован алгоритм;* и, наконец, лучшая степень. Последняя необходима лишь теоретически, поскольку показывает степень сжатия наилучшего (как правило, абсолютно черного) изображения, иногда фиксированного размера.
2. *Класс изображений*, на который ориентирован алгоритм. Иногда указано также, почему на других классах изображений получаются худшие результаты.
3. *Симметричность*. Отношение характеристики алгоритма кодирования к аналогичной характеристике при декодировании. Характеризует ресурсоемкость процессов кодирования и декодирования. Для нас наиболее важной является симметричность по времени: отношение времени кодирования ко времени декодирования. Иногда нам потребуется симметричность по памяти.
4. *Есть ли потери качества?* И если есть, то за счет чего изменяется степень сжатия? Дело в том, что у большинства алгоритмов сжатия с потерей информации существует возможность изменения степени сжатия.
5. Характерные особенности алгоритма и изображений, к которым его применяют. Здесь могут указываться наиболее важные для алгоритма свойства, которые могут стать определяющими при его выборе.

Используя данные критерии, приступим к рассмотрению алгоритмов архивации изображений. Прежде, чем непосредственно начать разговор об алгоритмах, хотелось бы сделать оговорку. Один и тот же алгоритм часто

можно реализовать разными способами. Многие известные алгоритмы, такие как RLE, LZW или JPEG, имеют десятки различающихся реализаций. Кроме того, у алгоритмов бывает несколько явных параметров, варьируя которые, можно изменять характеристики процессов архивации и разархивации. (См. примеры в разделе о форматах). При конкретной реализации эти параметры фиксируются, исходя из наиболее вероятных характеристик входных изображений, требований на экономию памяти, требований на время архивации и т.д. Поэтому у алгоритмов одного семейства лучшая и худшая степени сжатия могут отличаться, но качественно картина не изменится.

2.2.6. Общие моменты для прямоугольных методов

В любом случае можно начать с одного из четырех углов, и дальше двигаться в одном из двух направлений: по вертикали и по горизонтали. Первое, то есть положение первого угла, влияния на степень сжатия почти не оказывает, особенно при сжатии изображений. А вот второе, выбор направления, может существенно улучшить сжатие, поскольку области в этих двух случаях (основное направление по вертикали или по горизонтали) будут сгруппированы по-разному.

Например, отсканированный текст лучше сжимать, обходя по вертикали, поскольку в нем больше длинных сплошных вертикальных линий, чем горизонтальных.

Общие моменты для методов сложной формы. Может возникнуть необходимость помечать уже обработанные точки плоскости, чтобы избежать лишних вычислений, предотвращающих повторное их занесение в *D*. Тогда есть два основных варианта: либо добавить по одному «флаговому» биту для каждой точки плоскости, либо выбрать (или добавить) значение для «флага», показывающего, что точка уже внесена в *D*, и записывать это значение на место уже внесенных точек.

2.2.7. Алгоритмы архивации без потерь

2.2.7.1. Алгоритм RLE

Первый вариант алгоритма. Данный алгоритм необычайно прост в реализации. Групповое кодирование — от английского Run Length Encoding (RLE) — один из самых старых и самых простых алгоритмов архивации графики (рис. 5). Изображение в нем (как и в нескольких алгоритмах, описанных ниже) вытягивается в цепочку байт по строкам растра. Само сжатие в RLE происходит за счет того, что в исходном изображении встречаются цепочки одинаковых байт. Замена их на пары <счетчик повторений, значение> уменьшает избыточность данных. В данном алгоритме признаком счетчика (counter) служат единицы в двух верхних битах считанного файла:

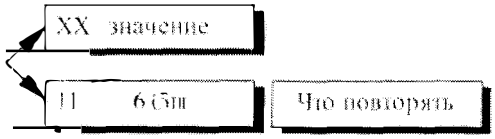


Рисунок 5.

Соответственно оставшиеся 6 бит расходуются на счетчик, который может принимать значения от 1 до 64. Строку из 64 повторяющихся байтов мы превращаем в два байта, т.е. сожмем в 32 раза.

Алгоритм рассчитан на деловую графику — изображения с большими областями повторяющегося цвета. Ситуация, когда файл увеличивается, для этого простого алгоритма не так уж редка. Ее можно легко получить, применяя групповое кодирование к обработанным цветным фотографиям. Для того, чтобы увеличить изображение в два раза, его надо применить к изображению, в котором значения всех пикселов больше двоичного 11000000 и подряд попарно не повторяются. Данный алгоритм реализован в формате PCX.

Второй вариант алгоритма. Второй вариант этого алгоритма имеет большую максимальную степень сжатия и меньше увеличивает в размерах исходный файл. Признаком повтора в данном алгоритме является единица в старшем разряде соответствующего байта (рис. 6):

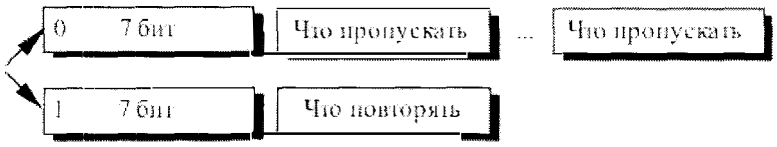


Рисунок 6.

Так можно легко подсчитать, в лучшем случае этот алгоритм сжимает файл в 64 раза (а не в 32 раза, как в предыдущем варианте), в худшем увеличивает на 1/128. Средние показатели степени компрессии данного алгоритма находятся на уровне показателей первого варианта.

Класс изображений:

Ориентирован алгоритм на изображения с небольшим количеством цветов: деловую и научную графику.

Симметричность: Примерно единица.

Характерные особенности: К положительным сторонам алгоритма, пожалуй, можно отнести только то, что он не требует дополнительной памяти при архивации и разархивации, а также быстро работает. Интересная особенность группового кодирования состоит в том, что степень архивации для некоторых изображений может быть существенно повышена всего лишь за счет изменения порядка цветов в палитре изображения.

2.2.7.2. Алгоритм LZW

Название алгоритм получил по первым буквам фамилий его разработчиков — Lempel, Ziv и Welch. Сжатие в нем, в отличие от RLE, осуществляется уже за счет одинаковых цепочек байт.

2.2.7.3. Алгоритм LZ

Существует довольно большое семейство LZ-подобных алгоритмов, различающихся, например, методом поиска повторяющихся цепочек. Один из

достаточно простых вариантов этого алгоритма, например, предполагает, что во входном потоке идет либо пара <счетчик, смещение относительно текущей позиции>, либо просто <счетчик> «пропускаемых» байт и сами значения байтов (как во втором варианте алгоритма RLE). При разархивации для пары <счетчик, смещение> копируются <счетчик> байт из выходного массива, полученного в результате разархивации, на <смещение> байт раньше, а <счетчик> (т.е. число равное счетчику) значений «пропускаемых» байт просто копируются в выходной массив из входного потока (рис. 7). Данный алгоритм является несимметричным по времени, поскольку требует полного перебора буфера при поиске одинаковых подстрок. В результате нам сложно задать большой буфер из-за резкого возрастания времени компрессии. Однако потенциально построение алгоритма, в котором на <счетчик> и на <смещение> будет выделено по 2 байта (старший бит старшего байта счетчика — признак повтора строки / копирования потока), даст нам возможность сжимать все повторяющиеся подстроки размером до 32Кб в буфере размером 64Кб.

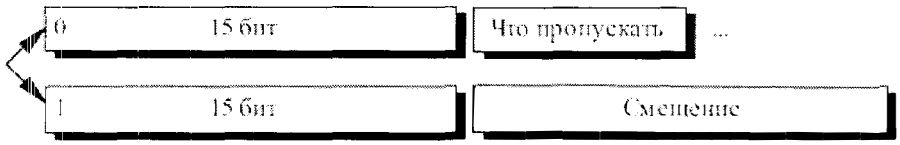


Рисунок 7.

При этом мы получим увеличение размера файла в худшем случае на 2770/32768 (в двух байтах записано, что нужно переписать в выходной поток следующие 2^{15} байт), что совсем неплохо. Максимальная степень сжатия составит в пределе 8192 раза. В пределе, поскольку максимальное сжатие мы получаем, превращая 32Кб буфера в 4 байта, а буфер такого размера мы накопим не сразу. Однако, минимальная подстрока, для которой нам выгодно проводить сжатие, должна состоять в общем случае минимум из 5 байт, что и определяет малую ценность данного алгоритма. К достоинствам LZ можно отнести чрезвычайную простоту алгоритма декомпрессии.

2.2.8. Алгоритмы архивации с потерями

Проблемы алгоритмов архивации с потерями. Первыми для архивации изображений стали применяться привычные алгоритмы. Те, что использовались и используются в системах резервного копирования, при создании дистрибутивов и т.п. Эти алгоритмы архивировали информацию без изменений. Однако основной тенденцией в последнее время стало использование новых классов изображений. Старые алгоритмы перестали удовлетворять требованиям, предъявляемым к архивации. Многие изображения практически не сжимались, хотя «на взгляд» обладали явной избыточностью. Это привело к созданию нового типа алгоритмов — сжимающих с потерей информации. Как правило, степень сжатия и, следовательно, степень потерь качества в них можно задавать. При этом достигается компромисс между размером и качеством изображений.

Одна из серьезных проблем машинной графики заключается в том, что до сих пор не найден адекватный критерий оценки потерь качества изображения. А теряется оно постоянно — при оцифровке, при переводе в ограниченную палитру цветов, при переводе в другую систему цвето-представления для печати, и, что для нас особенно важно, при архивации с потерями. Можно привести пример простого критерия: среднеквадратичное отклонение значений пикселов (L_2 мера, или root mean square — RMS):

$$d(x,y) = \sqrt{\frac{\sum_{i,j=1}^{n,m} (x_{ij} - y_{ij})^2}{n^2}} \qquad (3)$$

По нему изображение будет сильно испорчено при понижении яркости всего на 5% (глаз этого не заметит — у разных мониторов настройка яркости варьируется гораздо сильнее). В то же время изображения со «снегом» — резким изменением цвета отдельных точек, слабыми полосами или «муаром» будут признаны «почти не изменившимися». Свои неприятные стороны есть и у других критериев.

Рассмотрим, например, максимальное отклонение:

$$d(x,y) = \max_{i,j} |x_{ij} - y_{ij}| \quad (4)$$

Эта мера, как можно догадаться, крайне чувствительна к биению отдельных пикселов. Т.е. во всем изображении может существенно измениться только значение одного пиксела (что практически незаметно для глаза), однако согласно этой мере изображение будет сильно испорчено.

Мера, которую сейчас используют на практике, называется мерой отношения сигнала к шуму (peak-to-peak signal-to-noise ratio — PSNR).

$$d(x,y) = 10 \cdot \log_{10} \frac{255^2 \cdot n^2}{\sum_{i,j=1}^{n,n} (x_{ij} - y_{ij})^2} \quad (5)$$

Данная мера, по сути, аналогична среднеквадратичному отклонению, однако пользоваться ей несколько удобнее за счет логарифмического масштаба шкалы. Ей присущи те же недостатки, что и среднеквадратичному отклонению.

Лучше всего потери качества изображений оценивают наши глаза. Отличной считается архивация, при которой невозможно на глаз различить первоначальное и разархивированное изображения. Хорошей — когда сказать, какое из изображений подвергалось архивации, можно только сравнивая две находящихся рядом картинки. При дальнейшем увеличении степени сжатия, как правило, становятся заметны побочные эффекты, характерные для данного алгоритма. На практике, даже при отличном сохранении качества, в изображение могут быть внесены регулярные специфические изменения. Поэтому алгоритмы архивации с потерями не рекомендуется использовать при сжатии изображений, которые в дальнейшем собираются либо печатать с высоким качеством, либо обрабатывать программами распознавания образов. Неприятные эффекты с такими изображениями, как мы уже говорили, могут возникнуть даже при простом масштабировании изображения.

2.2.8.1. Алгоритм JPEG

JPEG— один из самых новых и достаточно мощных алгоритмов. Практически он является стандартом де-факто для полноцветных

изображений. Оперирует алгоритм областями 8x8, на которых яркость и цвет меняются сравнительно плавно. Вследствие этого, при разложении матрицы такой области в двойной ряд по косинусам (см. формулы ниже) значимыми оказываются только первые коэффициенты. Таким образом, сжатие в JPEG осуществляется за счет плавности изменения цветов в изображении.

Алгоритм разработан группой экспертов в области фотографии специально для сжатия 24-битных изображений. JPEG — Joint Photographic Expert Group — подразделение в рамках ISO — Международной организации по стандартизации. Название алгоритма читается ['jei'peg]. В целом алгоритм основан на дискретном косинусоидальном преобразовании (в дальнейшем ДКП), применяемом к матрице изображения для получения некоторой новой матрицы коэффициентов. Для получения исходного изображения применяется обратное преобразование.

ДКП раскладывает изображение по амплитудам некоторых частот. Таким образом, при преобразовании мы получаем матрицу, в которой многие коэффициенты либо близки, либо равны нулю. Кроме того, благодаря несовершенству человеческого зрения, можно аппроксимировать коэффициенты более грубо без заметной потери качества изображения.

Для этого используется квантование коэффициентов (quantization). В самом простом случае — это арифметический побитовый сдвиг вправо. При этом преобразовании теряется часть информации, но может достигаться большая степень сжатия.

Как работает алгоритм. Итак, рассмотрим алгоритм подробнее. Пусть мы сжимаем 24-битное изображение.

Шаг1. Переводим изображение из цветового пространства RGB, с компонентами, отвечающими за красную (Red), зеленую (Green) и синюю (Blue) составляющие цвета точки, в цветовое пространство YCrCb (иногда называют YUV).

В нем Y — яркостная составляющая, а Cr, Cb — компоненты, отвечающие за цвет (хроматический красный и хроматический синий). За счет того, что

человеческий глаз менее чувствителен к цвету, чем к яркости, появляется возможность архивировать массивы для Cr и Cb компонент с большими потерями и, соответственно, большими степенями сжатия. Подобное преобразование уже давно используется в телевидении. На сигналы, отвечающие за цвет, там выделяется более узкая полоса частот.

Упрощенно перевод из цветового пространства RGB в цветовое пространство YCrCb можно представить с помощью матрицы перехода (Таблица 12):

Таблица 4.

$$\begin{vmatrix} Y \\ Cb \\ Cr \end{vmatrix} = \begin{vmatrix} 0.299 & 0.587 & 0.114 \\ 0.5 & -0.4187 & -0.0813 \\ 0.1687 & -0.3313 & 0.5 \end{vmatrix} * \begin{vmatrix} R \\ G \\ B \end{vmatrix} + \begin{vmatrix} 0 \\ 128 \\ 128 \end{vmatrix}$$

Обратное преобразование осуществляется умножением вектора YUV на обратную матрицу.

Шаг 2. Разбиваем исходное изображение на матрицы 8x8. Формируем из каждой три рабочие матрицы ДКП — по 8 бит отдельно для каждой компоненты. При больших степенях сжатия этот шаг может выполняться чуть сложнее. Изображение делится по компоненте Y — как и в первом случае, а для компонент Cr и Cb матрицы набираются через строчку и через столбец. т.е. из исходной матрицы размером 16x16 получается только одна рабочая матрица ДКП. При этом, как нетрудно заметить, мы теряем 3/4 полезной информации о цветовых составляющих изображения и получаем сразу сжатие в два раза. Мы можем поступать так благодаря работе в пространстве YCrCb. На результирующем RGB изображении, как показала практика, это сказывается несильно.

Шаг 3. В упрощенном виде ДКП при n=8 можно представить так:

$$Y[u,v] = \frac{1}{4}\sum_{i=0}^{7}\sum_{j=0}^{7} C(i,u)\times C(j,v)\times \quad y[i,j]$$

$$\text{где} \quad C(i,u) = A(u)\times \quad \cos\left(\frac{(2\times i+1)\times u\times \pi}{2\cdot n}\right)$$

$$A(u) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{for} \quad u = 0 \\ 1, & \text{for} \quad u \neq 0 \end{cases} \qquad (4)$$

Применяем ДКП к каждой рабочей матрице. При этом мы получаем матрицу, в которой коэффициенты в левом верхнем углу соответствуют низкочастотной составляющей изображения, а в правом нижнем — высокочастотной. Понятие частоты следует из рассмотрения изображения как двумерного сигнала (аналогично рассмотрению звука как сигнала). Плавное изменение цвета соответствует низкочастотной составляющей, а резкие скачки — низкочастотной.

Шаг 4. Производим квантование. В принципе, это просто деление рабочей матрицы на матрицу квантования поэлементно. Для каждой компоненты (Y, U и V), в общем случае, задается своя матрица квантования q[u,v] (далее МК).

$$Yq[u,v] = \text{IntegerRound}\left(\frac{Y[u,v]}{q[u,v]}\right) \quad (5)$$

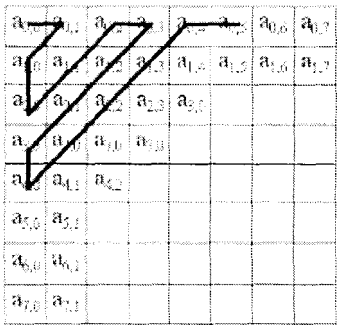
На этом шаге осуществляется управление степенью сжатия, и происходят самые большие потери. Понятно, что, задавая МК с большими коэффициентами, мы получим больше нулей и, следовательно, большую степень сжатия.

В стандарт JPEG включены рекомендованные МК, построенные опытным путем. Матрицы для большей или меньшей степени сжатия получают путем умножения исходной матрицы на некоторое число gamma.

С квантованием связаны и специфические эффекты алгоритма. При больших значениях коэффициента gamma потери в низких частотах могут быть настолько велики, что изображение распадется на квадраты 8x8. Потери в высоких частотах могут проявиться в так называемом «эффекте Гиббса», когда вокруг контуров с резким переходом цвета образуется своеобразный «нимб».

Шаг 5. Переводим матрицу 8x8 в 64-элементный вектор (Таблица 13) при помощи «зигзаг»-сканирования, т.е. берем элементы с индексами (0,0), (0,1), (1,0), (2,0)...

Таблица 5.



Таким образом, в начале вектора мы получаем коэффициенты матрицы, соответствующие низким частотам, а в конце — высоким.

Шаг 6. Свертываем вектор с помощью алгоритма группового кодирования. При этом получаем пары типа (пропустить, число), где «пропустить» является счетчиком пропускаемых нулей, а «число»— значение, которое необходимо поставить в следующую ячейку. Так, вектор 42 3 0 0 0 -2 0 0 0 0 1 ... будет свернут в пары (0,42) (0,3) (3,-2) (4,1)....

Шаг 7. Свертываем получившиеся пары кодированием по Хаффману с фиксированной таблицей.

Процесс восстановления изображения в этом алгоритме полностью симметричен. Метод позволяет сжимать некоторые изображения в 10-15 раз без серьезных потерь.

Конвейер операций, используемый в алгоритме JPEG.

Существенными положительными сторонами алгоритма является то, что:

1) Задается степень сжатия.
2) Выходное цветное изображение может иметь 24 бита на точку.

Отрицательными сторонами алгоритма является то, что:

3) При повышении степени сжатия изображение распадается на отдельные квадраты (8x8). Это связано с тем, что проис ходят большие потери в низких частотах при квантовании, и восстановить исходные данные становится невозможно.
4) Проявляется эффект Гиббса — ореолы по границам резких переходов цветов.

Как уже говорилось, стандартизован JPEG относительно недавно— в 1991 году. Но уже тогда существовали алгоритмы, сжимающие сильнее

при меньших потерях качества. Дело в том, что действия разработчиков стандарта были ограничены мощностью существовавшей на тот момент техники. То есть даже на персональном компьютере алгоритм должен был работать меньше минуты на среднем изображении, а его аппаратная реализация должна быть относительно простой и дешевой. Алгоритм должен был быть симметричным (время разархивации примерно равно времени архивации).

Выполнение последнего требования сделало возможным появление таких устройств, как цифровые фотоаппараты, снимающие 24-битовые фотографии на 8-256 Мб флэш карту. Потом эта карта вставляется в разъем на вашем ноутбуке и соответствующая программа позволяет считать изображения. Не правда ли, если бы алгоритм был несимметричен, было бы неприятно долго ждать, пока аппарат «перезарядится»— сожмет изображение.

Не очень приятным свойством JPEG является также то, что нередко горизонтальные и вертикальные полосы на дисплее абсолютно не видны и могут проявиться только при печати в виде муарового узора. Он возникает при наложении наклонного растра печати на горизонтальные и вертикальные полосы изображения. Из-за этих сюрпризов JPEG не рекомендуется активно использовать в полиграфии, задавая высокие коэффициенты матрицы квантования. Однако при архивации изображений, предназначенных для просмотра человеком, он на данный момент незаменим.

Широкое применение JPEG долгое время сдерживалось, пожалуй, лишь тем, что он оперирует 24-битными изображениями. Поэтому для того, чтобы с приемлемым качеством посмотреть картинку на обычном мониторе в 256-цветной палитре, требовалось применение соответствующих алгоритмов и, следовательно, определенное время. В приложениях, ориентированных на придирчивого пользователя, таких, например, как игры, подобные задержки неприемлемы. Кроме того, если имеющиеся у вас изображения, допустим, в 8-битном формате GIF перевести в 24-битный JPEG, а потом обратно в GIF для

просмотра, то потеря качества произойдет дважды при обоих преобразованиях. Тем не менее, выигрыш в размерах архивов зачастую настолько велик (в 3-20 раз), а потери качества настолько малы, что хранение изображений в JPEG оказывается очень эффективным.

Несколько слов необходимо сказать о модификациях этого алгоритма. Хотя JPEG и является стандартом ISO, формат его файлов не был зафиксирован. Пользуясь этим, производители создают свои, несовместимые между собой форматы, и, следовательно, могут изменить алгоритм. Так, внутренние таблицы алгоритма, рекомендованные ISO, заменяются ими на свои собственные. Кроме того, легкая неразбериха присутствует при задании степени потерь. Например, при тестировании выясняется, что «отличное» качество, «100%» и «10 баллов» дают существенно различающиеся картинки. При этом, кстати, «100%» качества не означают сжатие без потерь. Встречаются также варианты JPEG для специфических приложений.

Как стандарт ISO JPEG начинает все шире использоваться при обмене изображениями в компьютерных сетях. Поддерживается алгоритм JPEG в форматах Quick Time, PostScript Level 2, Tiff 6.0 и, на данный момент, занимает видное место в системах мультимедиа.

Характеристики алгоритма JPEG:

Степень сжатия: 2-200 (Задается пользователем).

Класс изображений: Полноцветные 24 битные изображения или изображения в градациях серого без резких переходов цветов (фотографии).

Симметричность: 1

Характерные особенности: В некоторых случаях, алгоритм создает «ореол» вокруг резких горизонтальных и вертикальных границ в изображении (эффект Гиббса). Кроме того, при высокой степени сжатия изображение распадается на блоки 8x8 пикселов.

2.2.8.2. Фрактальный алгоритм

Идея метода. Фрактальная архивация основана на том, что мы представляем изображение в более компактной форме — с помощью коэффициентов системы итерируемых функций (Iterated Function System— далее по тексту как IFS). Прежде, чем рассматривать сам процесс архивации, разберем, как IFS строит изображение, т.е. процесс декомпрессии.

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

Наиболее наглядно этот процесс продемонстрировал Барнсли в своей книге «Fractal Image Compression» (Рис. 7). Там введено понятие фотокопировальной Машины, состоящей из экрана, на котором изображена исходная картинка, и системы линз, проецирующих изображение на другой экран:

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

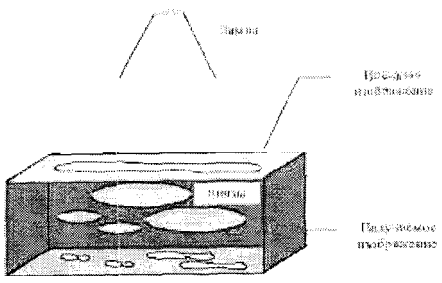


Рисунок. Машина Барнсли

Рисунок 7.

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

Поскольку отображение линз является сжимающим, каждая линза в явном виде задает *самоподобные* области в нашем изображении. Благодаря самоподобию мы получаем сложную структуру изображения при любом увеличении. Таким образом, интуитивно понятно, что система итерируемых функций задает *фрактал* (нестрого --- самоподобный математический объект).

Наиболее известны два изображения, полученных с помощью IFS: «треугольник Серпинского» (Рис. 8) и «папоротник Барнсли» (Рис.9).

Треугольник Серпинского

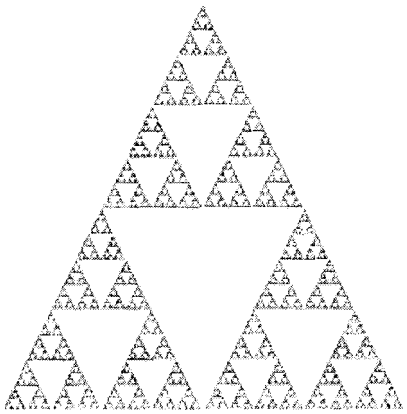


Рисунок 8.

«Треугольник Серпинского» задается тремя, а «папоротник Барнсли» четырьмя аффинными преобразованиями (или, в нашей терминологии,

«линзами»). Каждое преобразование кодируется буквально считанными байтами. в то время как изображение, построенное с их помощью, может занимать и несколько мегабайт.

Папоротник Барнсли. Задается 4 преобразованиями.

Рисунок 9.

Из вышесказанного становится понятно, как работает архиватор, и почему ему требуется так много времени. Фактически, фрактальная компрессия — это поиск самоподобных областей в изображении и определение для них параметров аффинных преобразований. В худшем случае, если не будет применяться оптимизирующий алгоритм, потребуется перебор и сравнение всех возможных фрагментов изображения разного размера. Даже для небольших изображений при учете дискретности мы получим астрономическое число перебираемых вариантов. Причем, даже резкое сужение классов преобразований, например, за счет масштабирования только в определенное количество раз, не дает заметного выигрыша во времени. Кроме того, при этом теряется качество изображения. Подавляющее большинство исследований в области фрактальной компрессии сейчас направлены на уменьшение времени архивации, необходимого для получения качественного изображения.

Далее приводятся основные определения и теоремы, на которых базируется фрактальная компрессия. Этот материал более детально и с доказательствами рассматривается в [3] и в [4].

Определение. Преобразование w: $R^2 \longrightarrow R^2$, представимое в виде

$$w(\vec{r}) = w\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \bullet \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix} \qquad (6)$$

где *a, b, c, d, e, f* действительные числа и (*x y)e* R^2 называется *двумерным аффинным преобразованием.*

Определение. Преобразование *w:*R^3 —>R^3,

$$w(\vec{r}) = w\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a & b & t \\ c & d & u \\ r & s & p \end{pmatrix} \bullet \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} e \\ f \\ q \end{pmatrix} \qquad (7)$$

где a, b, c, d, e, f, p, q, r, s, t, u действительные числа и (*x y z)* e R^3 называется трехмерным аффинным преобразованием.

Определение. Пусть *f* : X —> X — преобразование в пространстве X. Точка Xf e X такая, что *f(Xf) = Xf* называется *неподвижной точкой (аттрактором)* преобразования.

Определение. Преобразование *f* : X —> X в метрическом пространстве (X, d) называется сжимающим, если существует число s: 0 <= *s* < 1, такое, что

$$d(f(x),f(y)) <= s * d(x,y) \text{ ->x,y} \supset \text{ x}$$

Замечание: Формально мы можем использовать любое сжимающее отображение при фрактальной компрессии, но реально используются лишь трехмерные аффинные преобразования с достаточно сильными ограничениями на коэффициенты.

Теорема. (О сжимающем преобразовании)

Пусть *f*: X —-> X — сжимающее преобразование в полном метрическом пространстве (X, *d).* Тогда существует в точности одна неподвижная точка x_f *пенадлежит* X этого преобразования, и для любой точки x $\supset$ X последовательность {fn(x): n =0,1,2... } сходится к *Xf.*

Более общая формулировка этой теоремы гарантирует нам сходимость.

Определение. Изображением называется функция S, определенная на единичном квадрате и принимающая значения от 0 до 1 или $S(x,y) \supset [0...1]$ $\forall x,y \supset [0...1]$

Пусть трехмерное аффинное преобразование $w_i : R3 \longrightarrow R3$, записано в виде

$$w_i(\bar{r}) = w_i\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & p \end{pmatrix} \bullet \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} e \\ f \\ q \end{pmatrix} \quad (8)$$

и определено на компактном подмножестве *Di* декартова квадрата [0..1]x[0..1] (мы пользуемся особым видом матрицы преобразования, чтобы уменьшить размерность области определения с R^3 до R^2). Тогда оно переведет часть поверхности *S* в область R_i, расположенную со сдвигом *(e,f)* и поворотом, заданным матрицей

$$\begin{pmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad (9)$$

При этом, если интерпретировать значения функции *S(x,y)* е [0...1J как яркость соответствующих точек, она уменьшится в *p* раз (преобразование обязано быть сжимающим) и изменится на сдвиг *q*.

Построение алгоритма. Как уже стало очевидным из изложенного выше, основной задачей при компрессии фрактальным алгоритмом является нахождение соответствующих аффинных преобразований. В самом общем случае мы можем переводить любые по размеру и форме области изображения, однако в этом случае получается астрономическое число перебираемых вариантов разных фрагментов, которое невозможно обработать на текущий момент даже на суперкомпьютере.

В данном *варианте алгоритма,* изложенном далее, сделаны следующие ограничения на области:

1) Все области *являются квадратами со сторонами, параллельными сторонам изображения.* Это ограничение достаточно жесткое. Фактически мы собираемся аппроксимировать все многообразие геометрических фигур

лишь квадратами.

2) При переводе доменной области в ранговую уменьшение размеров производится *ровно в два раза.* Это существенно упрощает как компрессор, так и декомпрессор, т.к. задача масштабирования небольших областей является нетривиальной.
3) Все доменные блоки — квадраты и имеют *фиксированный размер.* Изображение равномерной сеткой разбивается на набор доменных блоков.
4) Доменные области берутся *«через точку» и по X, и по Y,* что сразу уменьшает перебор в 4 раза.
5) При переводе доменной области в ранговую поворот куба возможен *только на 0°, 90°, 180° или 270°.* Также допускается зеркальное отражение. Общее число возможных преобразований (считая пустое)
6) Масштабирование (Рис. 10) (сжатие) по вертикали (яркости) осуществляется *в фиксированное число раз* — в 0,75.

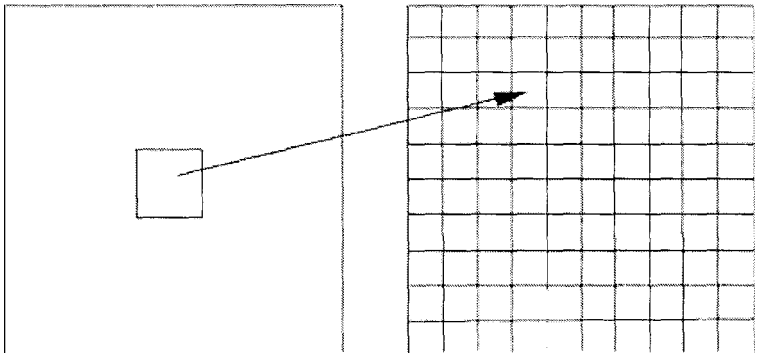


Рисунок 10.

Эти ограничения позволяют:

1) Построить алгоритм, для которого требуется сравнительно малое число операций даже на достаточно больших изображениях.
2) Очень компактно представить данные для записи в файл. Нам требуется на каждое аффинное преобразование в IFS:
   - два числа для того, чтобы задать смещение доменного блока. Если мы ограничим входные изображения размером 512x512, то

достаточно будет по 8 бит на каждое число.

- три бита для того, чтобы задать преобразование симметрии при переводе доменного блока в ранговый.
- 7-9 бит для того, чтобы задать сдвиг по яркости при пере воде.

Информацию о размере блоков можно хранить в заголовке файла. Таким образом, мы затратили менее 4 байт на одно аффинное преобразование. В зависимости от того, каков размер

блока, можно высчитать, сколько блоков будет в изображении. Таким образом, мы можем получить оценку степени компрессии.

Например, для файла в градациях серого 256 цветов 512x512 пикселов при размере блока 8 пикселов аффинных преобразований будет 4096 (512/8*512/8). На каждое потребуется 3.5 байта. Следовательно, если исходный файл занимал 262144 (512*512) байт (без учета заголовка), то файл с коэффициентами будет занимать 14336 байт. Степень сжатия— 18 раз. При этом мы не учитываем, что файл с коэффициентами тоже может обладать избыточностью и архивироваться методом архивации без потерь, например LZW.

Отрицательные стороны предложенных ограничений:

1) Поскольку все области являются квадратами, невозможно воспользоваться подобием объектов, по форме далеких от квадратов (которые встречаются в реальных изображениях достаточно часто.)
2) Аналогично мы не сможем воспользоваться подобием объектов в изображении, коэффициент подобия между которыми сильно отличается.
3) Алгоритм не сможет воспользоваться подобием объектов в изображении, угол между которыми не кратен 90°.

Такова плата за скорость компрессии и за простоту упаковки коэффициентов в файл. Сам алгоритм упаковки сводится к перебору всех доменных блоков и подбору для каждого соответствующего ему рангового блока.

Как видно из приведенного алгоритма, для каждого рангового блока делаем его проверку со всеми возможными доменными блоками (в том числе с

прошедшими преобразование симметрии), находим вариант с наименьшей мерой L_2 (наименьшим среднеквадратичным отклонением) и сохраняем коэффициенты этого преобразования в файл. Коэффициенты — это (1) координаты найденного блока, (2) число от 0 до 7, характеризующее преобразование симметрии (поворот, отражение блока), и (3) сдвиг по яркости для этой пары блоков. Сдвиг по яркости вычисляется как:

$$q = \left[\sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} - \sum_{i=1}^{n} \sum_{j=1}^{n} r_{ij} \right] / n^2, \quad (10)$$

При этом мера считается как:

$$d(R,D) = \sum_{i=1}^{n} \sum_{j=1}^{n} (0.75 r_{ij} + q - d_{ij})^2 \quad (11)$$

Мы не вычисляем квадратного корня из L_2 меры и не делим ее на n, поскольку данные преобразования монотонны и не помешают нам найти экстремум, однако мы сможем выполнять на две операции меньше для каждого блока.

Посчитаем количество операций, необходимых нам для сжатия изображения в градациях серого 256 цветов 512x512 пикселов при размере блока 8 пикселов. Таким образом, нам удалось уменьшить число операций алгоритма компрессии до вполне вычисляемых (пусть и за несколько часов) величин.

Схема алгоритма декомпрессии изображений. Декомпрессия алгоритма фрактального сжатия чрезвычайно проста. Необходимо провести несколько итераций трехмерных аффинных преобразований, коэффициенты которых были получены на этапе компрессии.

В качестве начального может быть взято абсолютно любое изображение (например, абсолютно черное), поскольку соответствующий математический аппарат гарантирует нам сходимость последовательности изображений, получаемых в ходе итераций IFS, к неподвижному изображению (близкому к исходному). Обычно для этого достаточно 16 итераций.

Прочитаем из файла коэффициенты всех блоков; Создадим черное изображение нужного размера;

Until(изображение не станет неподвижным)

{ For(every range (R))

{D=image1>CopyBlock(D coord for R);

For(every pixel *(i,j)* in the block

{$Rij = 0.75Dij - o_R$; } //Next pixel

//Next block

Until end

Поскольку мы записывали коэффициенты для блоков Rij (которые, как мы оговорили, в нашем частном случае являются квадратами одинакового размера) *последовательно,* то получается, что мы последовательно заполняем изображение по квадратам сетки разбиения использованием аффинного преобразования.

Как можно подсчитать, количество операций на один пиксел изображения в градациях серого при восстановлении необычайно мало (N операций сложения «+» и N операций умножения «*», где N — количество итераций, т.е. 7-16). Благодаря этому, декомпрессия изображений для фрактального алгоритма проходит быстрее декомпрессии, например, для алгоритма JPEG. В простой реализации JPEG на точку приходится 64 операции сложения «+» и 64 операции умножения «*». При реализации быстрого ДКП можно получить, 7 сложений и 5 умножений на точку, но это без учета шагов RLE, квантования и кодирования по Хаффману. При этом для фрактального алгоритма умножение происходит на рациональное число, одно для каждого блока. Это означает, что мы можем, во-первых, использовать целочисленную рациональную арифметику, которая быстрее арифметики с плавающей точкой. Во-вторых, можно использовать умножение вектора на число — более простую и быструю операцию, часто закладываемую в архитектуру процессора (процессоры SGI, Intel MMX, векторные операции Athlon и т.д.). Для

полноцветного изображения ситуация качественно не изменяется, поскольку перевод в другое цветовое пространство используют оба алгоритма.

Оценка потерь и способы их регулирования. При кратком изложении упрощенного варианта алгоритма были пропущены многие важные вопросы. Например, что делать, если алгоритм не может подобрать для какого-либо фрагмента изображения подобный ему? Достаточно очевидное решение — разбить этот фрагмент на более мелкие, и попытаться поискать для них. В то же время понятно, что эту процедуру нельзя повторять до бесконечности, иначе количество необходимых преобразований станет так велико, что алгоритм перестанет быть алгоритмом компрессии. Следовательно, мы допускаем потери в какой-то части изображения.

Для фрактального алгоритма компрессии, как и для других алгоритмов сжатия с потерями, очень важны механизмы, с помощью которых можно будет регулировать степень сжатия и степень потерь. К настоящему времени разработан достаточно большой набор таких методов. Во-первых, можно ограничить количество аффинных преобразований, заведомо обеспечив степень сжатия не ниже фиксированной величины. Во-вторых, можно по требовать, чтобы в ситуации, когда разница между обрабатываемым фрагментом и наилучшим его приближением будет выше определенного порогового значения, этот фрагмент дробился обязательно (для него обязательно заводится несколько «линз»). В- третьих, можно запретить дробить фрагменты размером меньше, допустим, четырех точек. Изменяя пороговые значения и приори тет этих условий, мы будем очень гибко управлять коэффициен том компрессии изображения в диапазоне от побитового соответ ствия до любой степени сжатия. Заметим, что эта гибкость будет гораздо выше, чем у ближайшего «конкурента»-— алгоритма JPEG.

Характеристики фрактального алгоритма: Степень сжатия: 2-2000 (Задается пользователем).

Класс изображений: Полноцветные 24 битные изображения или изображения в градациях серого без резких переходов цветов (фотографии).

Желательно, чтобы области большей значимости (для восприятия) были более контрастными и резкими, а области меньшей значимости — неконтрастными и размытыми.

Симметричность: 100-100000

Характерные особенности: Может свободно масштабировать изображение при разархивации, увеличивая его в 2-4 раза без появления «лестничного эффекта». При увеличении степени компрессии появляется «блочный» эффект на границах блоков в изображении.

2.2.8.3. Рекурсивный (волновой) алгоритм

Английское название рекурсивного сжатия — wavelet. На русский язык оно переводится как волновое сжатие, как сжатие с использованием всплесков, а в последнее время и калькой вэйвлет-сжатие. Этот вид архивации известен довольно давно и напрямую исходит из идеи использования когерентности областей. Ориентирован алгоритм на цветные и черно-белые изображения с плавными переходами. Идеален для картинок типа рентгеновских снимков. Степень сжатия задается и варьируется в пределах 5-100. При попытке задать больший коэффициент на резких границах, особенно проходящих по диагонали, проявляется «лестничный эффект» –- ступеньки разной яркости размером в несколько пикселов.

Идея алгоритма заключается в том, что мы сохраняем в файл разницу — число между средними значениями соседних блоков в изображении, которая обычно принимает значения, близкие к 0. Так два числа a_{2i} и $a_{2i-}i$ всегда можно представить в виде b^1 $i=(a_2i+a2i+i)/2$ и $b^2j=(a_{2i}-a_{2m})/2$. Аналогично последовательность a_t может быть попарно переведена в последовательность $b^{1,2}{}_t$. Разберем конкретный пример: пусть мы сжимаем строку из 8 значений яркости пикселов *(a,):* (220, 211, 212, 218, 217, 214, 210, 202). Мы получим следующие последовательности b^1_t и $b^2{}_t$: (215.5, 215, 215.5, 206) и (4.5, -3, 1.5, 4). Заметим, что значения b^2_t достаточно близки к 0. Повторим операцию, рассматривая b^1_t как a_t. Данное действие выполняется как бы рекурсивно, откуда и название алгоритма. Мы получим из (215.5, 215, 215.5, 206):(215.25,

210.75) (0.25, 4.75). Полученные коэффициенты, округлив до целых и сжав, например, с помощью алгоритма Хаффмана с фиксированными таблицами, мы можем поместить в файл.

Заметим, что мы применяли наше преобразование к цепочке только два раза. Реально мы можем позволить себе применение wavelet- преобразования 4-6 раз. Более того, дополнительное сжатие можно получить, используя таблицы алгоритма Хаффмана с неравномерным шагом (т.е. нам придется сохранять код Хаффмана для ближайшего в таблице значения). Эти приемы позволяют достичь заметных степеней сжатия.

В первой, как легко догадаться, будет храниться уменьшенная копия изображения. Во второй — усредненные разности пар значений пикселов по горизонтали. В третьей — усредненные разности пар значений пикселов по вертикали. В четвертой — усредненные разности значений пикселов по диагонали. По аналогии с двумерным случаем мы можем повторить наше преобразование и получить вместо первой матрицы 4 матрицы размером 128x128. Повторив наше преобразование в третий раз, мы получим в итоге: 4 матрицы 64x64, 3 матрицы 128x128 и 3 матрицы 256x256. На практике при записи в файл, значениями, получаемыми в последней строке ($b4_{t,j}$), обычно пренебрегают (сразу получая выигрыш примерно на треть размера файла — 1- 1/4 -1/16 - 1/64...). К достоинствам этого алгоритма можно отнести то, что он очень легко позволяет реализовать возможность постепенного «проявления» изображения при передаче изображения по сети. Кроме того, поскольку в начале изображения мы фактически храним его уменьшенную копию, упрощается показ «огрубленного» изображения по заголовку.

В отличие от JPEG и фрактального алгоритма данный метод не оперирует блоками, например, 8x8 пикселов. Точнее, мы оперируем блоками 2x2, 4x4, 8x8 и т.д. Однако за счет того, что коэффициенты для этих блоков мы сохраняем независимо, мы можем достаточно легко избежать дробления изображения на «мозаичные» квадраты.

Характеристики волнового алгоритма: Степень: 2-200 (Задается пользователем).

Класс изображений: Как у фрактального и JPEG.

Симметричность: ~1.5

Характерные особенности: Кроме того, при высокой степени сжатия изображение распадается на отдельные блоки.

2.2.8.4. Алгоритм JPEG 2000

Алгоритм JPEG-2000 разработан той же группой экспертов в области фотографии, что и JPEG. Формирование JPEG как международного стандарта было закончено в 1992 году. В 1997 стало ясно, что необходим новый, более гибкий и мощный стандарт, который и был доработан к зиме 2000 года. Основные отличия алгоритма в JPEG 2000 от алгоритма в JPEG заключаются в следующем:

Лучшее качество изображения при сильной степени сжатия. Или, что то же самое, большая степень сжатия при том же качестве для высоких степеней сжатия. Фактически это означает заметное уменьшение размеров графики «Web-качества», используемой большинством сайтов.

Поддержка кодирования отдельных областей с лучшим качеством. Известно, что отдельные области изображения критичны для восприятия человеком (например, глаза на фотографии), в то время как качеством других можно пожертвовать (например, задний план). При «ручной» оптимизации увеличение степени сжатия проводится до тех пор, пока не будет потеряно качество в какой-то важной части изображения. Сейчас появляется возможность задать качество в критичных областях, сжав остальные области сильнее, т.е. мы получаем еще большую окончательную степень сжатия при субъективно равном качестве изображения.

Основной алгоритм сжатия заменен на wavelet. Помимо указанного повышения степени сжатия это позволило изба виться от 8-пиксельной блочности, возникающей при повышении степени сжатия. Кроме того,

плавное проявление изображения теперь изначально заложено в стандарт (Pro gressive JPEG, активно применяемый в Интернет, появился много позднее JPEG).

Для повышения степени сжатия в алгоритме используется арифметическое сжатие. Изначально в стандарте JPEG также было заложено арифметическое сжатие, однако позднее оно было заменено менее эффективным сжатием по Хаффману, поскольку арифметическое сжатие было защищено патентами. Сейчас срок действия основного патента истек, и появилась возможность улучшить алгоритм.

Поддержка сжатия без потерь. Помимо привычного сжатия с потерями новый JPEG теперь будет поддерживать и сжатие без потерь. Таким образом, становится возможным использование JPEG для сжатия медицинских изображений, в полиграфии, при сохранении текста под распознавание OCR системами и т.д.

Поддержка сжатия однобитных (2-цветных) изображений. Для сохранения однобитных изображений (рисунки тушью, отсканированный текст и т.п.) ранее повсеместно рекомендовался формат GIF, поскольку сжатие с использованием ДКП весьма неэффективно к изображениям с резкими переходами цветов. В JPEG при сжатии 1-битная картинка приводилась к 8-битной, т.е. увеличивалась в 8 раз, после чего делалась попытка сжимать, нередко менее чем в 8 раз. Сейчас можно рекомендовать JPEG 2000 как универсальный алгоритм.

На уровне формата поддерживается прозрачность. Плавно накладывать фон при создании WWW страниц теперь можно будет не только в GIF, но и в JPEG 2000. Кроме того, поддерживается не только 1 бит прозрачности (пиксел прозрачен/непрозрачен), а отдельный канал, что позволит задавать плавный переход от непрозрачного изображения к прозрачному фону.

Кроме того, на уровне формата поддерживаются включение в изображение информации о копирайте, поддержка устойчивости к битовым

ошибкам при передаче и широковещании, можно запрашивать для декомпрессии или обработки внешние средства (plug-ins), можно включать в изображение его описание, информацию для поиска и т.д.

Идея алгоритма. Базовая схема JPEG-2000 очень похожа на базовую схему JPEG. Отличия заключаются в следующем:

1) Вместо дискретного косинусного преобразования (DCT) используется дискретное вэйвлет-преобразование (DWT).
2) Вместо кодирования по Хаффману используется арифметическое сжатие.
3) В алгоритм изначально заложено управление качеством областей изображения.

Не используется явно дискретизация компонент U и V после преобразования цветовых пространств, поскольку при DWT можно достичь того же результата, но более аккуратно.

Конвейер операций, используемый в алгоритме JPEG-2000.

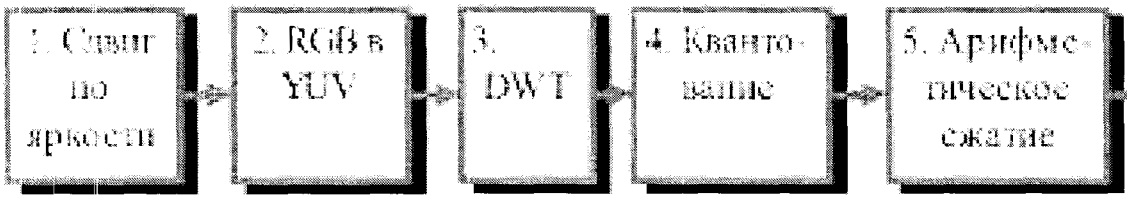


Рисунок 11.

Рассмотрим алгоритм по шагам.

Шаг 1. В JPEG-2000 предусмотрен сдвиг яркости (DC level shift) каждой компоненты (RGB) изображения перед преобразованием в YUV. Это делается для выравнивания динамического диапазона (приближения к 0 гистограммы частот), что приводит к увеличению степени сжатия. Значение степени ST для как каждой компоненты R, G и B свое (определяется при сжатии компрессором). При восстановлении изображения выполняется обратное преобразование:

$$I'(x, y) = I(x, y) + 2^{ST-1} \qquad (12)$$

Шаг 2. Переводим изображение из цветового пространства RGB, с компонентами, отвечающими за красную (Red), зеленую (Green) и синюю

(Blue) составляющие цвета точки, в цветовое пространство YUV. Этот шаг аналогичен JPEG (см. матрицы преобразования в описании JPEG), за тем исключением, что кроме преобразования с потерями предусмотрено также и преобразование без потерь.

$$\begin{vmatrix} R \\ G \\ B \end{vmatrix} = \begin{vmatrix} U + G \\ Y - \left\lfloor \frac{U + V}{4} \right\rfloor \\ V + G \end{vmatrix} \quad (13)$$

Шаг 3. Дискретное wavelet преобразование (DWT) также может быть двух видов для случая сжатия с потерями и для сжатия без потерь. Его коэффициенты задаются таблицами (Таблица 14), приведенными ниже. Для сжатия с потерями коэффициенты выглядят как:

Таблица 6.

Коэффициенты при упаковке		
i	Низкочастотные коэффициенты $h_l(i)$	Высокочастотные коэффициенты $h_h(i)$
0	1.115087052456994	0.6029490182363579
±1	0.5912717631142470	-0.2668641184428723
±2	-0.05754352622849957	-0.07822326652898785
±3	-0.09127176311424948	0.01686411844287495
±4	0	0.02674875741080976
Другие i	0	0

Коэффициенты при распаковке		
i	Низкочастотные коэффициенты $g_l(i)$	Высокочастотные коэффициенты $g_h(i)$
0	0.6029490182363579	1.115087052456994
±1	-0.2668641184428723	0.5912717631142470
±2	-0.07822326652898785	-0.05754352622849957

Само преобразование в одномерном случае представляет собой скалярное произведение коэффициентов фильтра на строку преобразуемых значений (в нашем случае — на строку изображения). При этом четные выходящие значения формируются с помощью низкочастотного преобразования, а нечетные с помощью высокочастотного:

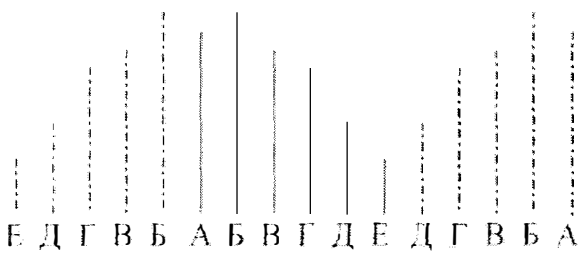
Таблица 7.

i	**Низкочастотные коэффициенты $h_L(i)$**	**Высокочастотные коэффициенты $h_H(i)$**	**Низкочастотные коэффициенты $g_L(i)$**	**Высокочастотные коэффициенты $g_H(i)$**
0	6/8	1	1	6/8
±1	2/8	-1/2	1/2	-2/8
±2	-1/8	0	0	-1/8

$$y_{output}(2n) = \sum_{j=0}^{X_{max}} x_{input}(j) \cdot h_H(j-2n)$$
$$y_{output}(2n+1) = \sum_{j=0}^{X_{max}} x_{input}(j) \cdot h_L(j-2n-1) \quad (14)$$

Поскольку большинство $h_L(i)$, кроме окрестности /=0, равны О, то можно переписать приведенные формулы с меньшим количеством операций. Для простоты рассмотрим случай сжатия без потерь. Легко показать, что данную запись можно эквивалентно переписать, уменьшив еще втрое количество операций умножения и деления (однако теперь необходимо будет подсчитать сначала все нечетные *y).* Добавим также операции округления до ближайшего целого, не превышающего заданное число *a,* обозначаемые как [a]*:*

$$y_{out}(2n+1) = x_{in}(2n+1) - \left\lfloor \frac{x_{in}(2n) + x_{in}(2n+2)}{2} \right\rfloor$$
$$y_{out}(2n) = x_{in}(2n) + \left\lfloor \frac{y_{out}(2n-1) + y_{out}(2n+1) + 2}{4} \right\rfloor \quad (15)$$



Симметричное расширение изображения (яркости АБ...Е) по строке

Рисунок 12.

Пусть мы преобразуем строку из 10 пикселов. Расширим ее значения вправо и влево и применим DWT преобразование(Таблица 16):

Таблица 16.

n	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11
x_{in}	3	2	1	2	3	7	10	15	12	9	10	5	10	9
y_{out}		0	1	0	3	1	11	4	13	-2	8	-5		

Получившаяся строка 1, 0, 3, 1, 11, 4, 13, -2, 8, -5 и является цепочкой, однозначно задающей исходные данные. Совершив аналогичные преобразования с коэффициентами для распаковки, приведенными выше в таблице, получим необходимые формулы:

$$x_{out}(2n) = y_{out}(2n) - \left\lfloor \frac{y_{out}(2n-1) + y_{out}(2n+1) + 2}{4} \right\rfloor$$
$$x_{out}(2n+1) = y_{out}(2n+1) + \left\lfloor \frac{x_{out}(2n) + x_{out}(2n+2)}{2} \right\rfloor \quad (15)$$

Упражнение: Докажите, что во всех случаях округления мы будем получать одинаковые входную и выходную цепочки. Легко проверить (используя преобразование упаковки), что значения на концах строк в y_{out} также симметричны относительно $n=0$ и 9. Как видим (Таблица 17), мы получили исходную цепочку (Xin= x_{out}).

Таблица 8.

n	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11
y_{out}		0	1	0	3	1	11	4	13	-2	8	-5	8	-2
x_{out}			1	2	3	7	10	15	12	9	10	5	10	

Далее к строке применяется чересстрочное преобразование, суть которого заключается в том, что все четные коэффициенты переписываются в начало строки, а все нечетные — в конец. В результате этого преобразования в начале строки формируется «уменьшенная копия» всей строки (низкочастотная составляющая), а в конце строки — информация о колебаниях значений промежуточных пикселов (Таблица 18) (высокочастотная составляющая).

Таблица 9.

$y_{исх}$	1	0	3	1	11	4	13	-2	8	-5
$y'_{исх}$	1	3	11	13	8	0	1	4	-2	-5

Это преобразование применяется сначала ко всем строкам изображения, а затем ко всем столбцам изображения. В результате изображение делится на 4 квадранта (примеры смотрите в описании рекурсивного сжатия). В первом квадранте будет сформирована уменьшенная копия изображения, а в остальных трех — высокочастотная информация. После чего преобразование повторно применяется уже только к первому квадранту изображения по тем же правилам (преобразование второго уровня).

Для корректного сохранения результатов под данные 2 и 3 квадрантов выделяется на один бит больше, а под данные 4-го квадранта — на 2 бита больше. т.е. если исходные данные были 8-битные, то на 2 и 3 квадранты нужно 9 бит, а на 4-й — 10, независимо от уровня применения DWT. При записи коэффициентов в файл можно использовать иерархическую структуру DWT, помещая коэффициенты преобразований с большего уровня в начало файла. Это позволяет получить «изображение для предварительного просмотра», прочитав небольшой участок данных из начала файла, а не распаковывая весь файл, как это приходилось делать при сжатии изображения целиком. Иерархичность преобразования может также использоваться для плавного улучшения качества изображения при передаче его по сети.

Шаг 4. Так же, как и в алгоритме JPEG, после DWT применяется квантование. Коэффициенты квадрантов делятся на заранее заданное число. При увеличении этого числа снижается динамический диапазон коэффициентов, они становятся ближе к 0, и мы получаем большую степень сжатия. Варьируя эти числа для разных уровней преобразования, для разных цветовых компонент и для разных квадрантов, мы очень гибко управляем степенью потерь в изображении. Рассчитанные в компрессоре оптимальные коэффициенты квантования передаются в декомпрессор для однозначной распаковки.

Шаг 5. Для сжатия получающихся массивов данных в JPEG 2000 используется вариант арифметического сжатия, называемый MQ-кодер, прообраз которого (QM-кодер) рассматривался еще в стандарте JPEG, но реально не использовался из-за патентных ограничений. Подробнее об алгоритме арифметического сжатия читайте в соответствующей главе раздела.

Области повышенного качества. Основная задача, которую мы решаем — повышение степени сжатия изображений. Когда практически достигнут предел сжатия изображения в целом и различные методы дают очень небольшой выигрыш, мы можем существенно (в разы) увеличить степень сжатия за счет изменения качества разных участков изображения.

Проблемой этого подхода является то, что необходимо каким-то образом получать расположение наиболее важных для человека участков изображения. Например (Рис. 13), таким участком на фотографии человека является лицо, а на лице — глаза. Если при сжатии портрета с большими потерями будут размыты предметы, находящиеся на заднем плане — это будет несущественно. Однако если будет размыто лицо или глаза — экспертная оценка степени потерь будет хуже.

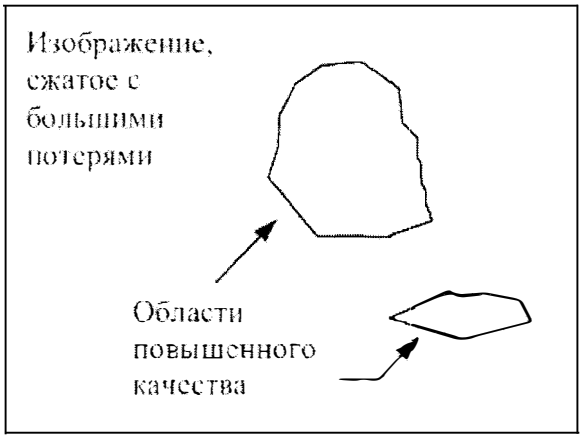


Рисунок 13

Локальное улучшение качества областей изображения. Работы по автоматическому выделению таких областей активно ведутся. В частности, созданы алгоритмы автоматического выделения лиц на изображениях. Продолжаются исследования методов выделения наиболее значимых (при

анализе изображения мозгом человека) контуров и т.д. Однако очевидно, что универсальный алгоритм в ближайшее время создан не будет, поскольку для этого требуется построить полную схему восприятия изображений мозгом человека.

На сегодня вполне реально применение полуавтоматических систем, в которых качество областей изображения будет задаваться интерактивно. Данный подход уменьшает количество возможных областей применения модифицированного алгоритма, но *позволяет достичь большей степени сжатия.*

Такой подход логично применять, если:

1) Для приложения должна быть критична (максимальна) степень сжатия, причем настолько, что возможен индивидуальный подход к каждому изображению.
2) Изображение сжимается один раз, а разжимается множество раз.

В качестве примеров приложений, удовлетворяющим этим ограничениям, можно привести практически все мультимедийные продукты на CD-ROM. И для CD-ROM энциклопедий, и для игр важно записать на диск как можно больше информации, а графика, как правило, занимает до 70% всего объема диска. При этом технология производства дисков позволяет сжимать каждое изображение индивидуально, максимально повышая степень сжатия.

Интересным примером являются WWW-сервера. Для них тоже, как правило, выполняются оба изложенных выше условия. При этом совершенно не обязательно индивидуально подходить к каждому изображению, поскольку по статистике 10% изображений будут запрашиваться 90% раз. т.е. для крупных справочных или игровых серверов появляется возможность уменьшать время загрузки изображений и степень загруженности каналов связи адаптивно.

В JPEG-2000 используется однобитное изображение-маска (Рис. 14), задающее повышение качества в данной области изображения. Поскольку за качество областей у нас отвечают коэффициенты DWT преобразования во 2, 3 и

4 квадрантах, то маска преобразуется таким образом, чтобы указывать на все коэффициенты, соответствующие областям повышения качества:

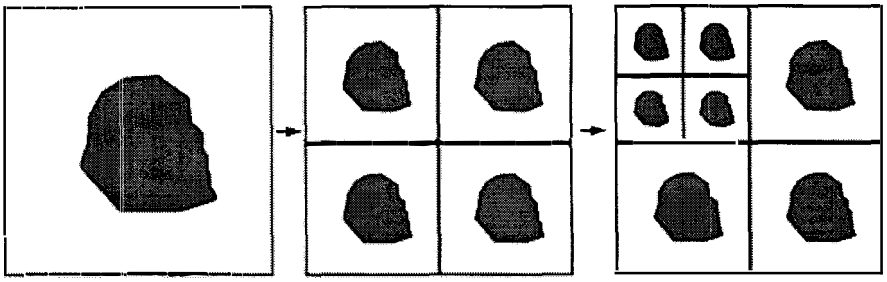


Рисунок 14.

Преобразование маски области повышения качества для обработки DWT коэффициентов. Эти области обрабатываются далее другими алгоритмами (с меньшими потерями), что и позволяет достичь искомого баланса по общему качеству и степени сжатия.

Характеристики алгоритма JPEG-2000:

Степень сжатия: 2-200 (Задается пользователем). Возможно сжатие без потерь.

Класс изображений: Полноцветные 24-битные изображения. Изображения в градациях серого без резких переходов цветов (фотографии). 1-битные изображения.

Симметричность: 1-1,5

Характерные особенности: Позволяет удалять визуально неприятные эффекты, повышая качество в отельных областях. При сильном сжатии появляется блочность и большие волны в вертикальном и горизонтальном направлениях.

Приоритеты развития алгоритмов графики последних лет:

1) ориентация на фотореалистичные изображения с 16 миллионами цветов (24 бита);
2) использование сжатия с потерями, возможность за счет потерь регулировать качество сжатых изображений;
3) использование избыточности изображений в двух измерениях;

4) появление существенно несимметричных алгоритмов;

5) увеличивающаяся степень сжатия изображений.

Различия между форматом и алгоритмом. Напоследок несколько замечаний относительно разницы в терминологии, путаницы при сравнении рейтингов алгоритмов и т.п. Посмотрите на краткий перечень *форматов,* достаточно часто используемых на PC, Apple и UNIX платформах: ADEX, Alpha Microsystems BMP, Autologic, AVHRR, Binary Information File (BIF), Calcomp CCRF, CALS, Core IDC, Cubicomp PictureMaker, Dr. Halo CUT, Encapsulated PostScript, ER Mapper Raster, Erdas LAN/GIS, First Publisher ART, GEM VDI Image File, GIF, GOES, Hitachi Raster Format, PCL, RTL, HP-48sx Graphic Object (GROB), HSI JPEG, HSI Raw, IFF/ILBM, Img Software Set, Jovian VI, JPEG/JFIF, Lumena CEL, Macintosh PICT/PICT2, MacPaint, MTV Ray Tracer Format, OS/2 Bitmap, PCPAINT/Pictor Page Format, PCX, PDS, Portable BitMap (PBM), QDV, QRT Raw, RIX, Scodl, Silicon Graphics Image, SPOT Image, Stork, Sun Icon, Sun Raster, Targa, TIFF, Utah Raster Toolkit Format, VITec, Vivid Format, Windows Bitmap, WordPerfect Graphic File, XBM, XPM, XWD.

В оглавлении вы можете видеть список *алгоритмов компрессии.* Единственным совпадением оказывается JPEG, а это, согласитесь, не повод, чтобы повсеместно использовать слова *«формат»* и *«алгоритм компрессии»* как синонимы (что, увы, можно часто наблюдать).

Между этими двумя множествами нет взаимно однозначного соответствия. Так, различные модификации алгоритма RLE реализованы в огромном количестве форматов. В том числе в TIFF, BMP, PCX. И, если в определенном формате какой-либо файл занимает много места, это не означает, что плох соответствующий алгоритм компрессии. Это означат, зачастую лишь то, что реализация алгоритма, использованная в этом формате, дает для данного изображения плохие результаты. Не более того.

В то же время многие современные *форматы* поддерживают запись с использованием нескольких *алгоритмов архивации* либо без использования архивации. Например, *формат* TIFF 6.0 может сохранять изображения с

использованием *алгоритмов* RLE-PackBits, RLE-CCITT, LZW, Хаффмана с фиксированной таблицей, JPEG, а может сохранять изображение без архивации. Аналогично *форматы* BMP и TGA позволяют сохранять файлы как с использованием *алгоритма* компрессии *RLE* (разных модификаций!), так и без использования оной.

Вывод: Для многих *форматов,* говоря о размере файлов, необходимо указывать, использовался ли *алгоритм* компрессии и если использовался, то какой.

Можно пополнить перечень ситуаций некорректного сравнения алгоритмов. При сохранении абсолютно черного изображения в формате 1000x1000x256 цветов в формате BMP без компрессии мы получаем, как и положено, файл размером чуть более 1000000 байт, а при сохранении с компрессией RLE, можно получить файл размером 64 байта. Это был бы превосходный результат — сжатие в 15 000 раз(!), если бы к нему имела отношение компрессия. Дело в том, что данный файл в 64 байта состоит только из заголовка изображения, в котором указаны все его данные. Несмотря на то, что такая короткая запись изображения стала возможна именно благодаря особенности реализации RLE в BMP, еще раз подчеркнем, что в данном случае *алгоритм* компрессии *даже не применялся.* И то, что для абсолютно черного изображения 4000x4000x256 мы получаем коэффициент сжатия 250 тысяч раз, совсем не повод для продолжительных эмоций по поводу эффективности RLE. Кстати --- данный результат возможен лишь при определенном положении цветов в палитре и далеко не на всех программах, которые умеют записывать BMP с архивацией RLE (однако все стандартные средства, в т.ч. средства системы Windows, читают такой сжатый файл нормально).

Всегда полезно помнить, что на размер файла оказывают существенное влияние большое количество параметров (вариант реализации алгоритма, параметры алгоритма (как внутренние, так и задаваемые пользователем), порядок цветов в палитре и многое другое). Например, для абсолютно черного изображения 1000x1000x256 градаций серого в формате JPEG с помощью одной

программы при различных параметрах *всегда* получался файл примерно в 7 килобайт. В то же время, меняя опции в другой программе, я получил файлы размером от 4 до 68 Кб (всего-то на порядок разницы). При этом декомпрессированное изображение для всех файлов было одинаковым — *абсолютно* черный квадрат (яркость 0 для всех точек изображения).

Дело в том, что даже для простых форматов *одно и то же изображение в одном и том же формате* с использованием *одного и того же алгоритма* архивации можно записать в файл *несколькими корректными способами.* Для сложных форматов и алгоритмов архивации возникают ситуации, когда многие программы сохраняют изображения разными способами. Такая ситуация, например, сложилась с форматом TIFF (в силу его большой гибкости). Долгое время по-разному сохраняли изображения в *формат* JPEG, поскольку соответствующая группа ISO (Международной Организации по Стандартизации) подготовила только стандарт *алгоритма,* но не стандарт *формата.* Сделано так было для того, чтобы не вызывать «войны форматов». Абсолютно противоположное положение сейчас с фрактальной компрессией, поскольку есть стандарт «де-факто» на сохранение фрактальных коэффициентов в файл (стандарт *формата),* но *алгоритм* их нахождения (быстрого нахождения!) является технологической тайной создателей программ-компрессоров. В результате для вполне стандартной программы-декомпрессора могут быть подготовлены файлы с коэффициентами, существенно различающиеся как по размеру, так и по качеству получающегося изображения.

Приведенные примеры показывают, что встречаются ситуации, когда алгоритмы записи изображения в файл в различных программах различаются. Однако гораздо чаще причиной разницы файлов являются разные *параметры алгоритма.* Как уже говорилось, многие алгоритмы позволяют в известных пределах менять свои параметры, но не все программы позволяют это делать пользователю.

Вывод 2: Если вы не умеете пользоваться программами архивации или пользуетесь программами, в которых «для простоты использования» убрано управление параметрами алгоритма — не удивляйтесь, почему для отличного алгоритма компрессии в результате получаются большие файлы.

2.3. Мультимедийные структуры

2.3.1. Формат AVI-файла

Сегодняшние кодеки позволяют хранить полутора-двухчасовой фильм на одном компакт-диске, а объемы жестких дисков перевалили за сотню гигабайт. Естественно, что хранить видео- и аудиоинформацию в цифровом виде значительно удобнее: цифровая система хранения упрощает оформление и систематизацию, значительно облегчает поиск необходимого, а также повышает удобство и качество просмотра. Кроме того, цифровое хранение упрощает закрытие доступа к информации и реализацию защиты от несанкционированного копирования видео- и аудиоматериалов.

Наверное, самым распространенным форматом для хранения видеоинформации является сегодня формат AVI. Компания Microsoft впервые выпустила пакет Video for Windows для операционной системы Windows 3.1 в ноябре 1992 года, и с тех пор формат AVI (Audio Video Interleave) является основным для хранения видео в операционных системах семейства Windows.

Основное преимущество универсального формата AVI (и, кстати, «секрет» его долголетия), в отличие от потоковых форматов типа MPEG, а тем более от таких специализированных разновидностей, как MP3 (MPEG Audio Layer 3), в том, что «стандартных» AVI-файлов практически не существует: AVI — фактически не более чем «контейнер», который содержит общее описание содержимого в стандартизованном виде. Наверное, многим приходилось сталкиваться с ситуацией, когда плейер, установленный в их системе, отказывался проигрывать перенесенный с другого компьютера AVI-файл. Дело в том, что видео- и аудиопотоки, которые содержатся в AVI-файле, могут использовать любую комбинацию кодеков. Эти кодеки устанавливаются

и регистрируются в операционной системе независимо друг от друга, а программы кодирования хранятся в отдельных DLL-файлах. Таким образом, помимо стандартных кодеков, которые были установлены при инсталляции самой системы, пакета обновления или очередной версии программы Media Player, со временем на вашем компьютере могут появиться и нестандартные кодеки, установленные какой-нибудь компьютерной игрой, программой видеомонтажа, драйвером видеокарты и пр. Если после этого вы воспользуетесь какой-либо программой, которая сохраняет видео в AVI-формате, и в качестве кодека выберите какой-нибудь нестандартный компрессор, то с большой вероятностью полученное видео можно будет просмотреть только на вашем компьютере.

Таким образом, AVI-файлы только внешне выглядят одинаково, но внутри они могут сильно различаться, и, в то время как MPEG-, MP3- и MJPEG-файлы содержат потоки только определенного вида сжатия (компрессии), AVI может содержать много различных видов компрессии в любых сочетаниях (например, в настоящее время особой популярностью пользуется такая пара: DivX для видео и WMA для аудио). Однако работать с AVI-файлом можно будет только пока необходимый кодек доступен для кодирования/декодирования. Впрочем, информация о кодеке может содержаться в AVI-файле, и если вы имеете доступ в Интернет, то плейер сам обратится на сайт Microsoft или другой компании, скачает необходимый кодек и установит его в системе. Из других подобных универсальных форматов компьютерного видео наиболее известен формат QuickTime MOV компании Apple. Windows Multimedia System

Воспроизведением AVI-файлов в ОС Windows управляет универсальная среда компании Microsoft, состоящая из интерфейса высокого уровня — MCI (Media Control Interface) или MCI API (Application Programming Interface) и низкоуровневых MCI-драйверов. Формат AVI (Audio Video Interleave — чередование аудио и видео) позволяет одновременно хранить изображение и звук. При воспроизведении видеопоток и дорожки звукового канала (waveform

audio, или WAV) анализируются, разделяются и обрабатываются драйверами различных устройств.

Простые аудио- и видеопотоки могут содержаться в WAV- или AVI-файле без какого-либо сжатия (компрессии). Простейший WAV имеет стандартный заголовок типа PCM (Pulse Coded Modulation) и содержит аудиоданные, обычно некомпрессированные, полученные 8- или 16-разрядным сэмлированием аналогового звука. Однако так же, как и в AVI, в WAV-файле может содержаться звуковая информация со сжатием, а также прописываться информация о необходимом для воспроизведения кодеке.

Структура AVI-формата является вариантом формата RIFF (Resource Interchange File Format), в свое время разработанного недолгим альянсом компаний IBM и Microsoft для обмена мультимедийными данными. Файлы этого формата имеют вложенную блочную структуру, то есть состоят из блоков (chunks), которые, в свою очередь, могут содержать другие вложенные блоки. Основной RIFF-блок содержит идентификатор формата «avi», который указывает на тип файла (на самом деле для идентификатора отведено четыре символа, но один из них не используется). Впрочем, и сам формат RIFF является клоном известного формата IFF, еще в 1984 году разработанного компанией Electronic Arts для платформы Amiga компании Commodore.

2.3.1.1 Структура AVI-файла

Итак, в соответствии с общей структурой RIFF-типа, AVI-файл должен иметь следующий вид:

RIFF 'AVI ' // четырехбуквенный идентификатор файла (в RIFF-формате)

LIST 'hdrl' // список заголовков блоков, определяющих форматы потоков

LIST 'movi' // блоки данных (потоков) AVI-файла

'idx1' // необязательный блок, определяющий размещение блоков данных внутри AVI-файла <AVI Index>

То есть в AVI-файле должно быть по крайней мере два обязательных блока: заголовка и данных, которые, в свою очередь, могут содержать

подблоки. Первый блок будет содержать общую информацию о видеоролике: разрешение кадров и их частоту, формат аудио и т.д. Сначала в заголовке для записи длины потока отводилось 32 байт, поскольку в файловой системе FAT 16 максимальный раздел диска не мог превышать 2 Гбайт, поэтому и максимальный кусок видео, который можно было записывать в AVI-файле, не мог превышать 2 Гбайт (с учетом знака переменной размера). Во времена возникновения формата казалось естественным, что длина файла не может превышать размер логического диска. С появлением файловых систем FAT 32 и NTFS верхняя граница размера раздела значительно отодвинулась, однако потребовалось еще немало времени, чтобы ввести расширение формата и дождаться программ, способных это ограничение обходить.

Список ‘hdrl’ может состоять из подсписков:

LIST 'hdrl' // список заголовков блоков, определяющих форматы потоков

'avih' // главный заголовок AVI-файла

LIST 'strl'

'strh' // заголовок потока

'strf' // формат потока

'strd' // дополнительный заголовок данных

Список 'movi', в свою очередь, состоит из подблоков:

LIST 'movi' // блоки данных (потоков) AVI-файла

SubChunk | LIST 'rec ' // подблок | список записей

'##wb' (размер блока 4 байта) (data) // звуковые данные (блок)

'##dc' (размер блока 4 байта) (data) // видеоданные (блок)

'##db' (размер блока 4 байта) (data) // видеоданные (блок)

Таким образом, подблок данных организован в виде последовательности записей, каждая из которых состоит из одного кадра видео и соответствующего звукового сопровождения.

Первоначально ##dc-блок был предназначен для хранения сжатого изображения, а ##db-блок — для несжатого DIB (Device Independent Bitmap). Но фактически они оба могут содержать сжатые данные.

Серьезным недостатком AVI-формата является то, что аудио- и видеофрагменты не содержат никаких меток времени или индексов кадра. Данные упорядочиваются по времени последовательно, в порядке поступления. Приложение для захвата или проигрывания видео должно само позаботиться о синхронизации видео- и аудиопотоков. Но если деление видео на кадры совершенно естественно, то звук представляет собой непрерывный поток, который приходится искусственно расчленять на фрагменты, соответствующие кадрам (из-за этого точная синхронизация изображения и звука часто отсутствует и звук может «расходиться» с изображением). В простейшем случае каждому кадру видео соответствует кусок звукового сопровождения, но далеко не все AVI-файлы делаются по этой простой схеме.

Недостаток временных меток был устранен в расширении AVI-формата — OpenDML AVI (поддержанный затем в DirectShow и в ActiveMovie), которое добавляет новые куски по меткам времени.

2.3.1.2. Цветовые палитры

Видео в AVI-формате может быть подготовлено и сохранено в различных цветовых пространствах, которые можно условно разделить по глубине цвета:

- 8-битная ч/б палитра (256 градаций серого);
- 8-битная RGB-палитра (256 цветов);
- 9-битная палитра YUV9;
- 12-битная YUV (4:1:1);
- 16-битная YUV2 (4:2:2);
- 16-битная RGB (5 бит для красного, 6 — для зеленого и 5 — для синего);
- 24-битная RGB (стандартная RGB-палитра);
- 32-битная RGB (с альфа-каналом).

По цветовой информативности аналогична 24-битной (старший байт служит для определения прозрачности). Остановимся подробнее на цветовом пространстве YUV, которое широко применяется в телевещании и соответственно пришло на компьютер вместе с MPEG-форматом. Дело в том,

что глаз человека наиболее чувствителен к яркости изображения и несколько менее — к цветности.

Если аддитивные составляющие RGB-сигнала (где R, G и B — соответственно красная, зеленая и синяя) представить в виде яркости (Y) и двух различных составляющих сигнала цветности (U и V) по следующим формулам:

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

$U = 0{,}493\ (B - Y)$

$V = 0{,}877\ (R - Y)$,

то U в таком соотношении выражает различия между синей и желтой составляющими цветного изображения, а V — между красной и зеленой компонентами цветного изображения. Известно, что глаза легче различают градации яркости и оттенки зеленого, поэтому U и V можно отображать с меньшей точностью, что способно существенно уменьшить количество сохраняемой информации. Таким образом, используя особенности человеческого зрения, еще до того, как будет применяться компрессия данных, мы можем получить выигрыш только за счет перехода к другому цветовому пространству. Поэтому, когда говорят, например, что сжатие в MPEG осуществляется с коэффициентом 100:1 и более, часто забывают, что часть информации была «потеряна» уже при переходе к другому цветовому пространству.

Формат кодирования YUV 4:1:1 отличается от 4:2:2 способом дискретизации сигнала, которая вычисляется для конкретного канала как произведение базовой частоты цифрового кодирования на соответствующий коэффициент: например на 4 отсчета для канала Y берется по одному отсчету для каждого из цветоразностных каналов.

2.3.1.3. Видеокодеки

Запись видео и аудио в AVI-формате может производиться как без сжатия, так и со сжатием (компрессией). Причем компрессия может быть как без потерь информации (изображение при декодировании восстанавливается

полностью), так и с потерями, которые, тем не менее, не отражаются на визуальном восприятии. Некоторые алгоритмы допускают значительное увеличение степени компрессии и проявляющиеся при этом ошибки визуализации остаются на вполне приемлемом уровне. Другие алгоритмы при повышении коэффициента компрессии демонстрируют легко обнаруживаемые и очень раздражающие дефекты. Существует очень много различных технологий, используемых компьютерными кодеками, но в настоящее время наиболее популярны основанные на различных вариациях DCT-алгоритма (дискретном косинусоидальном преобразовании) и алгоритмах компенсации движения кодеки типа H.261, H.263, MPEG-1, MPEG-2 и MPEG-4 и др.

Начиная с Video for Windows 1.1e, в системе Windows по умолчанию устанавливается несколько стандартных кодеков:

- Intel Indeo;
- Microsoft Video 1 (формат работает только с 8- и 16-битным цветом);
- Microsoft RLE (Run Length Encoding), который использует только 8-битный цвет;
- Cinepak (пришедший с платформы Macintosh вместе с QuickTime).

Впоследствии к ним добавились полноцветные кодеки, которые пришли вместе с платами для захвата и монтажа видео на компьютере:

- Motion JPEG;
- Editable MPEG, который использует только I-кадры.

Есть и довольно редкие, коммерческие кодеки, которые были созданы для специального использования. Так, например, появился кодек Sorenson Video, который использовался на компьютерах компании Apple в 1999 году для кодирования и передачи по Интернету фильма «Звездные войны. Эпизод I: Призрачная угроза». А с развитием Интернета к стандартным добавились и другие кодеки из так называемой новой волны, порожденные продвинутыми технологиями в области психологического восприятия и сложных алгоритмов компенсации движения. Например, если выделять из изображения, движущиеся и неподвижные части в отдельные объекты, то, применяя к ним раздельные

методы и коэффициенты сжатия (в простейшем случае — чем быстрее движение предмета в кадре, тем выше может быть коэффициент компрессии для его отображения), можно добиться значительного улучшения визуального восприятия при том же или даже меньшем потоке данных.

Современные кодеки (в последнее время все большую популярность приобретает формат компрессии по алгоритму DivX) обеспечивают высокое качество воспроизведения при относительно невысоком потоке данных.

2.3.2 Цифровое видео: MPEG

Большинству пользователей ПК хорошо знаком термин MPEG. В первую очередь с ним ассоциируется возможность смотреть на компьютере полнометражные полноэкранные видеофильмы (CD-i, Video-CD), что всего несколько лет назад казалось просто невозможным. Правда, до сих пор большинство людей воспринимают идею об использовании компьютера в качестве видеомагнитофона и телевизора как некую экзотику, функцию, предназначенную больше для демонстрации возможностей своего любимца, чем для повседневного применения. Но постепенно цена плат для проигрывания MPEG становится все более доступной, да и большинство современных компьютеров Pentium могут справиться с этой задачей за счет программных средств. Грядущая эра MMX окончательно сделает компьютерное видео привычным элементом на любом домашнем ПК. И если раньше найти Video-CD фильмы было довольно сложно, то сейчас их ассортимент состоит уже из сотен наименований, причем каждый месяц на Российском рынке появляются десятки новых дисков, многие из которых - прекрасные отечественные фильмы.

В отличие от AVI-формата, который обычно использует алгоритмы сжатия Motion-JPEG, MPEG-файлы занимают значительно меньше места. Сейчас, когда идея домашней видео-студии уже витает в воздухе, чтобы оцифровать и отредактировать одну минуту видео с качеством VHS (например, при помощи платы miroVideo DC20) требуется от 100 до 200 Мбайт дискового

пространства. Легко подсчитать, что для монтажа получасового свадебного ролика необходимо такое количество свободных Мбайт, которое вряд ли имеется в наличии на вашем жестком диске. В то же время с помощью MPEG-сжатия мы бы без проблем уложились в 1Гбайт. Но здесь есть свои проблемы: во-первых MPEG, в отличии от AVI очень тяжело редактировать и до последнего времени не было соответствующих редакторов; во-вторых, цена систем для оцифровки MPEG несколько выше, чем на обычные платы ввода видео. В материале "Почем нынче MPEG для народа?" мы рассмотрим три новые системы для оцифровки и редактирования MPEG, которые укладываются в ценовые рамки до двух тысяч долларов и предназначены в первую очередь для домашнего и корпоративного применения. Но вначале краткий обзор по структуре и спецификациям MPEG.

MPEG (Moving Picture Experts Group) - объединенный комитет Интернациональной Организации по Стандартизации (ISO) и Интернациональной Электротехнической Комиссии (IEC). Эта группа экспертов встречается примерно четыре раза в год, чтобы разработать и утвердить стандарты на сжатие цифрового видео и звука. Основной критерий, который обсуждается комитетом MPEG - это интенсивность потока сжатых данных, определяемая в зависимости от современного уровня компьютерных технологий и сферы применения данного формата. Так, MPEG-1 был разработан с учетом возможностей 2-скоростных дисководов CD-ROM и компьютеров с 486-процессором. Алгоритмы сжатия могут определяться самими производителями оборудования и микросхем, поэтому в этой области идет постоянная конкуренция за достижение лучших результатов.

Статус MPEG: в январе 1992 года комитет MPEG опубликовал общие характеристики MPEG-1, а к декабрю 1993 г. они были приняты в качестве стандарта (дополнительные материалы вы можете найти в статье "Живое видео на выставке Comptek`95", Мир ПК №7-8/95). По этим спецификациям интенсивность потока данных сжатого видео и звука должна укладываться в 1.5 Мбайт/с, хотя были предусмотрены режимы вплоть до 4-5 Мбайт/с. Важность

пониженного потока данных определялась существующими на тот момент стандартами Video-CD, CD-i и характеристиками дисководов CD-ROM. Базовый алгоритм ограничивает скорость передачи данных диапазоном 150-225 Кбайт/с с разрешении 352 x 288 (PAL) или 320 x 240 (NTSC) при частоте смены 25 или 30 кадров в секунду соответственно.

Окончательное утверждение MPEG-2 в качестве международного стандарта было дано на 29-м съезде MPEG (Сингапур, ноябре 1994). В его спецификациях определена допустимая интенсивность потока данных от 2 до 10 Мбайт/с. Первично MPEG-2 разрабатывался для цифровой передачи и отображения видео вещательного качества, но позже в нем был предусмотрен формат телевидения высокой четкости (ТВЧ). До этого необходимые спецификации для ТВЧ должны были быть включены в последующий MPEG-3 стандарт, с разрешением вплоть до 1920 x 1080 при частоте 30 Гц и интенсивности потока данных от 20 до 40 Мбайт/с. Оказалось, что с некоторой доработкой код MPEG-2 и даже MPEG-1 работает вполне нормально для задач HDTV. В результате разработка MPEG-3 была прекращена.

Зато начались работы над MPEG-4. Новый стандарт будет рассчитан на очень низкие потоки данных для применения в видеотелефонах, мультимедийной электронной почте, электронных информационных изданиях и т.п. Объявлено об этом было на встрече в Брюсселе в сентябре 1993. MPEG-4 будет оптимизирован для минимальных разрешений, вплоть до 176 x 144 при частоте 10 Гц, с интенсивностью потока данных от 4,800 до 64,000 бит в секунду. Для достижения нормальной производительности и приемлемого качества при столь низких требованиях к ресурсам скорей всего будет применена новая технология сжатия видео и аудио данных. Примерные технические спецификации нового стандарта ожидаются в 1997 году, а официальное его утверждение состоится не ранее ноября 1998 года.

2.3.2.1 Структура MPEG

В MPEG используется поточное сжатие видео, т. е. обрабатывается не каждый кадр по отдельности (как это происходит при сжатии видео с помощью алгоритмов Motion-JPEG), а анализируется динамика изменений видеофрагментов и устраняются избыточные данные, т. к. в большинстве фрагментов фон остается достаточно стабильным, а действие тем временем происходит на переднем плане. MPEG начинает сжатие с создания исходного (ключевого) кадра, называемого "I" или "Intra" (И) кадр. И-кадры играют роль опорных при восстановлении остальных изображений и размещаются последовательно через каждые 10-15 кадров. Только некоторые фрагменты изображений, которые находятся между И-кадрами претерпевают изменения, и именно эта разница сохраняется при сжатии. Кроме И-кадров в MPEG-последовательности имеется еще два типа изображений:

- Predicted (P) - предсказуемые (П) кадры, содержащие разность текущего изображения с предыдущим И или П с учетом смещений отдельных фрагментов;
- Bi-directional Interpolated (B) - двунаправленные (Д) кадры, содержащие только отсылки к предыдущим или последующим изображениям типа И или П с учетом смещений отдельных фрагментов.

И-кадры составляют основу MPEG файла и через них осуществляется случайный доступ к какому-либо отрывку видео, но при этом у них довольно низкий коэффициент сжатия. П-кадры кодируются относительно предыдущих кадров (И или П), и, обычно, используются как сравнительный образец для дальнейшей последовательности П-кадров. В этом случае достигается высокий коэффициент сжатия. Д-кадры обеспечивают наибольший коэффициент сжатия, но при этом для их привязки к видеопоследовательности необходимо использовать не только предыдущее, но и последующее изображение. Сами Д-кадры никогда не используются для сравнения. Изображения объединяются в группы (GOP - Group Of Pictures), представляющие собой минимальный

повторяемый набор последовательных изображений. Типичной является группа вида:

(И0 Д1 Д2 П3 Д4 Д5 П6 Д7 Д8 П9 Д10 Д11) (И12 Д13 Д14 П15 Д16 Д17 П18 ...)

Отдельные изображения состоят из структурных единиц - макроблоков, соответствующих участку изображения размером 16x16 пикселов. Компьютер анализирует изображение и ищет идентичные или практически похожие макроблоки, сравнивая базовый и последующие кадры. В результате сохраняются только данные о различиях между кадрами, называемые вектором смещения (vector movement code). Макроблоки, которые не претерпевают изменений игнорируются, и количество данных для реального сжатия и хранения существенно снижается. Для повышения устойчивости процесса восстановления изображений к возможным ошибкам передачи данных последовательные макроблоки объединяют в независимые друг от друга разделы (slices). В свою очередь каждый макроблок состоит из шести блоков, четыре из которых несут информацию о яркости Y, а по одному определяют цветовые U- и V-компоненты. Блоки являются базовыми структурными единицами, над которыми осуществляются основные операции кодирования, в том числе выполняется дискретное косинусное преобразование (DCT - discrete cosine transform). В результате при использовании MPEG можно добиться рабочего коэффициента сжатия более чем 200:1, хотя это приводит к некоторой потери качества.

2.3.2.2 MPEG-1

По своим качественным параметрам во многом аналогичен обычному VHS-видео, поэтому ему находится применение в первую очередь там, где неудобно или непрактично использовать стандартные аналоговые видео-носители.

Видео киоски (или информационные киоски) дают возможность по новому организовать и автоматизировать информационный сервис в рамках вашей организации. Особенно это важно для розничных магазинов, автомобильных салонов и банков. Продавец не всегда способен уделить

достаточно внимания сразу нескольким клиентам, тем более он просто не имеет возможности подробно рассказать о всех особенностях и спецификациях того или иного продукта, наглядно и эффектно его продемонстрировать. А видео киоск всегда будет в вашем распоряжении. В нем можно разместить не только подробную информационную базу о имеющихся продуктах и услугах, но и включить туда интерактивные видеофильмы, позволяющие просто и наглядно ответить на многие вопросы. К примеру: "Какие у вас имеются модели автомобилей?", "Расскажите об их особенностях?", "Какой цвет я могу выбрать?". Информация, которая ранее выводилась в виде слайдов и сопроводительного текста теперь становится более доступной и эффектной благодаря внедрению полноэкранного видео. Используя MPEG-1 разработчик может регулярно и без особых дополнительных затрат обновлять содержание видео киоска. Развитие программных средств и эволюция пользовательского интерфейса ведут к все более впечатляющей интерактивности подобных аппаратов.

Видео по требованию (Video on Demand). Термин "видео по требованию" появился сравнительно недавно. Вначале подобный сервис можно было встретить только в дорогих отелях, а теперь уже полным ходом идет реализация глобальной идеи об интерактивной цифровой телевизионной системе, через которую любой пользователь сможет запросить какой-либо фильм или передачу в определенное время и прямо на дом. Современные технологии позволяют говорить об этом проекте, как о грядущей реальности, хотя до появления подобного устройства в широком употреблении пройдет еще несколько лет.

Некоторые телефонные компании сейчас разрабатывают системы, которые позволят нам заказывать фильмы через обыкновенные телефонные линии. Правда, приходится учитывать ограниченную пропускную способность имеющихся телефонных коммуникаций, но повсеместное внедрение ISDN и других новых технологий связи помогут решить эту проблему.

Видео библиотеки: Организации, имеющие большие видео-архивы, могут существенно выиграть, перекодировав их в цифровой формат и поместив на CD-носители или специальный сервер. В отличии от аналоговых носителей данный метод гарантирует длительное хранение, многократное проигрывание без потери качества и быстрый доступ к любому фрагменту. К тому же, обладая подобным видеоматериалом, вы легко сможете открыть удаленный доступ к нему через внутреннюю локальную сеть (Интранет) или через WWW. Поэтому музеи, библиотеки, государственные предприятия и научные учреждения, так же как рекламные фирмы и информационные агентства переходят сейчас на цифровое видео.

2.3.2.3 MPEG-2

Спецификации MPEG-2 подразумевают использование высоких разрешений для достижения максимального качества изображения, поэтому этот формат применяется в первую очередь в профессиональных сферах.

Кабельное телевидение (CATV: Cable Television). Идея перевести кабельное телевидение на цифровое вещание напрашивается сама собой. Имеющиеся магистрали для передачи видеосигнала вполне могут выдержать интенсивность и объем данных, необходимые для вещания MPEG-видео высокого разрешения (MPEG-2). Уже в ближайшее время должны появиться первые подобные системы. Впервые пользователь реально сможет принимать телепередачи в высоком разрешении со стерео и Dolby Surround звуком.

Направленное спутниковое вещание (DBS: Direct Broadcast Satellite). Консорциум Hughes/USSB собирается использовать MPEG-2 для направленного вещания. Компания Thomson уже производит специальные декодеры, установив которые вы сможете принимать до 150 каналов. Правда, подобные системы работают пока только в Северной Америке.

ТВЧ - телевидение высокой четкости (HDTV: High-Definition Television). В США создан консорциум компаний (U.S. Grand Alliance), который разрабатывает новый стандарт HDTV для телевидения высокого разрешения. В

нем будет использоваться MPEG-2 с поддержкой следующих режимов: 1440 x 960 при 30 Гц и 1280 x 720 при 60 Гц. Легко себе представить, сколь высоко будет качество подобных телепередач.

В итоге можно утверждать, что MPEG является доминирующим стандартом для полнометражного цифрового видео, за исключением нелинейного цифрового монтажа, где в данный момент более распространен Motion-JPEG. Однако, по мере того как все большее количество кодирующих MPEG-систем будет появляться на рынке, MPEG-2 внедрится и в эту нишу. В ближайшее время появится большое количество новых сфер применения для технологии MPEG, начиная от высококачественных цифровых DVD-видеодисков и новейших игровых систем и заканчивая совершенными цифровыми вещательными и монтажными комплексами.

2.3.2.4. Вопросы и ответы по MPEG.

Различие между MPEG-1 и MPEG-2. MPEG-1 и MPEG-2 различаются объемом информации, содержащейся в видео-потоке, сжатом в соответствии с этими форматами, и, как следствие, качеством. Хотя MPEG-1 может работать с разрешениями вплоть до CCIR-601 (720 x 480), обычно он кодируется при значительно более низкой интенсивности потока данных, что приводит к худшему качеству видео. MPEG-1 обычно ассоциируется с SIF-разрешением (352x240), это аналогично качеству VHS. При воспроизведении такое изображение растягивается аппаратными или программными средствами вплоть до полного экрана, и хотя при этом теряется качество, зато остается возможность проигрывать полноэкранное видео даже с двухскоростных дисководов CD-ROM.

MPEG-2 поддерживает более высокие разрешения (в т.ч. CCIR-601). При этом объем данных в MPEG-2 более чем в четыре раза больше относительно MPEG-1, что позволяет записывать с его помощью полноэкранные фильмы "вещательного" (Betacam) качества. Этот формат избран для использования в новом поколении видеодисков на основе технологии DVD, а в скором времени

станет доминировать и на PC. В отличии от MPEG-1 для MPEG-2 необязательно наличие GOP-групп, и даже при отсутствии GOP-заголовка можно получить прямой доступ к видеофрагменту. Другой ключевой особенностью MPEG-2 является наличие в нем расширений, которые позволяют при записи видеосигнала разделение его на два или более независимо кодируемых потока данных, представляющих видео в различных разрешениях, с лучшим или худшим качеством изображения. Это делается с целью создания независимых потоков данных определенной интенсивности в рамках одного видео сигнала. Такая функция важна для приложений типа ТВЧ, когда необходимо одновременно транслировать ТВЧ и стандартный телевизионный сигнал.

QuickTime и Indeo не позволяют полноэкранного воспроизведения с вещательным качеством и не поддерживают оцифровки изображений с полной разверткой (два поля на кадр), что необходимо для профессионального применения. Это оригинальные разработки компаний Apple и Intel, ориентированные в первую очередь на мультимедиа-рынок. Только MPEG может быть реализован как программно, так и аппаратно на обеих платформах (PC и Mac). К тому же, в то время как MPEG позволяет достичь коэффициента сжатия до 200:1, QuickTime предусматривает 50:1, а Indeo обеспечивает уровень сжатия не более 10:1. При этом QuickTime и Indeo даже при низких коэффициентах сжатия не достигают качества MPEG. А возможно ли программное воспроизведение MPEG? Да. В зависимости от конфигурации можно добиться вполне приличного качества программного воспроизведения MPEG и Video-CD на компьютере с процессором Pentium. Три фактора, влияющие на качество проигрывания MPEG:

1. Производительность процессора
2. Производительность графического адаптера и поддержка MS DirectDraw
3. Скорость дисковода CD-ROM

Программные проигрыватели MPEG обычно прилагаются в поставке с графическими платами; так же их можно найти в Internet.

Почему имеется столь большая разница в цене между системами для оцифровки и воспроизведения MPEG? Оцифровка MPEG, особенно если это происходит в режиме реального времени требует очень больших аппаратных и вычислительных ресурсов, поэтому соответствующее оборудование имеет достаточно высокую стоимость. На сегодня минимальные цены на системы записи MPEG колеблются в рамках от 1,000 до 10,000 долл. MPEG-проигрыватель можно купить за 100-300 долл.

Что такое Video-CD? Video-CD - это процесс (и соответствующий формат), который позволяет записывать MPEG-видео на CD диск и воспроизводить его на любом оборудовании, поддерживающем данный формат.

2.3.3. Подведение итогов по мультимедийным форматам

Что касается мультимедиа форматов как объекта исследований, то можно с уверенностью сказать, что единственным открытым для нашей экспансии форматом, не имеющим компрессионного сжатия остается AVI формат. Но в свою очередь AVI, как таковой, в чистом виде найти сложно. Так как с появлением всевозможных кодаков в наше время видео потока AVI нет. Следовательно, его использование в нашем деле доставит трудности еще на этапе поиска и выбора контейнера. Хотя для большого объема скрытого сообщения он является лакомым кусочком для стеганографиста. Если сравнить то, что AVI формат очень похож на BitMap формат то зная степень упаковки BMP можно предположить, что AVI имеет такой же процент. Что в принципе я доказал практически. В 204 мБ AVI файла (мультфильм "Том и Джерри") я упаковал 18,2мБ (18634 кБ) текстовой информации. Ура победа! Однако есть два "но":

1. Мне пришлось искусственно копировать неоднократно текст и без того не малого сообщения;
2. Время на упаковку составило 56 минут.

Результат упаковки, в общем, порадовал. Но вот понадобится ли нам такой объем? Я склоняюсь все же к BitMap. А такие потрясающие результаты лучше применять как вариант для ПО. Кстати говоря, это место для вируса или еще чего страшней. Ведь для его активации нужна лишь программка декодер. Но до этого пока далеко.

3. СТЕГАНОГРАФИЧЕСКИЕ РЕШЕНИЯ, СКРЫТЫЕ СООБЩЕНИЯ НА БАЗЕ BITMAP

Проведя массированную атаку на различные форматы файловых структур хранения не менее различных по роду данных, пришел к выводу, что наиболее рационально следует сделать упор на золотую середину. В результате моих исследований файловых структур были получены очевидные данные. Исходя из полученных результатов, я остановлюсь на BMP формате (и как вариант источник JPEG). А делаю я это вот почему:

1. BMP формат имеет достаточно места (10-12%) для упаковки сообщения, имеющего вполне подходящий размер именно как сообщения призванного донести до читателя именно то что я и имел в виду, передавая такое сообщение;
2. В свою очередь BMP сам является средним(50-2 300кБ), по размерам, занимаемым на носителе по сравнению с текстовыми(10-100кБ) и мультимедийными (150-800мБ) файлами. Что выгодно сказывается на возможности транспортировки, как в ЛВС, так и в сети Интернет. Среднее время передачи через WWW при средней скорости 26кБит займет от 40 секунд до 5 минут, в зависимости от размера BMP контейнера;
3. Процесс упаковки в " BMP-контейнер" по количеству операций не будет перегружать процессор. Ведь даже если сообщение будет состоять из одного символа, придется все равно обрабатывать весь файл целиком, так как невозможно переписать лишь часть файла. К примеру, элементарное переписывание файла размером 2.5мБ по битам составляет 1 минуту и 32 секунды для Celeron 633 класса PentiumII. А решись я упаковать свое сообщение в 700мБ формата AVI пришлось бы откинуться на спинку кресла по методике инсталляции MS Windows на 7 часов. Я думаю, этот довод поставит точку в выборе приоритетов.

Для начала определимся с языком разработки прикладной программы. Языки программирования, такие как Pascal, C, C++ оставим в покое ввиду того

что современный пользователь испытывает ужас, видя строку запроса на действие. Наиболее распространенный вопрос: а почему мышь не работает? Что делать? прогресс! Да кстати назревает уместный вопрос: а почему автор забыл о могущественном Assembler. А вот здесь еще интересней. Если мы пугали Pascal 'ем обычного пользователя, то Assembler'а боится уже сама операционная оболочка. Нет, она не спрашивает где мышь, но прямые дисковые операции с файлами в операционных системах "голубых кровей" недопустимы. Поколение Windows NT следует упрашивать на выполнение того или иного действия с файлом, а согласится она или нет, это уже её стезя. Хотя Assembler может облегчить многие страдания в написании программы, но все же нет желания в требованиях к программе заявлять Windows'98 и ниже.

И так на лавке остались языки (точнее сказать среды разработки) объектно-ориентированные(Delphi7, C++Builder, MS VisualBasic). На VB написать толком можно только макрос для тётеньки с бухгалтерии, чтобы она не мучилась с созданием нового листа в Excel. C++Builder вот достойный ученик Assembler'a, но есть одно НО. В нашем регионе он не очень популярен, впрочем, как и на других "планетах". Найти помощь в нестандартных ситуациях становится негде. И так добрались до Delphi. Это среда объектно-ориентированной разработки Object Pascal 7.1. Конечно есть масса недостатков относительно C++ версии от Borland, но с этим можно мириться, используя обходные маневры.

Далее речь пойдет непосредственно о программной реализации стеганографических принципов в BMP формате файлов. Читателям неискушенным в программировании рекомендуется просто читать текст игнорируя неясные закорючки присвоений и инициализаций. Свою программу я назвал очень незамысловато. Это название сложилось само собой по инициалам фамилии и имени. СтеганоГрафия версии 2.2, так как первая версия была оставлена из-за уже описанных конфликтов с операционными оболочками.

Для непосредственной реализации своей задачи применю стратегию минимализма визуальных объектов. Ведь это нагрузка и без того насыщенного циклами программного кода. Я также не стану искусственно разбивать ротацию файла для прорисовки промежуточного результата из-за вероятности летального исхода преобразования. Все операции с файлом будут вынесены в отдельный модуль и разбиты на процедуры и функции что существенно ускорит работу с графикой. Так как каждая процедура оперирует лишь локальными переменными, не общаясь с внешней средой приложения. По своей сути программа состоит из двух отдельных модулей. Которые несут разную нагрузку.

Первый лишь вспомогательный, в меру дружелюбный интерфейс программы предлагает операционные действия над контейнером и сообщением. Где особой достопримечательностью являются сведения о разработчике.

Вторая часть программы с лихвой нагружена преобразовательными функциями по работе с графическими форматами и битовыми преобразованиями. По сути, это и есть сердце программы. Модуль TGraph.pas был откомпилирован в библиотеку правил использования ресурсов графики CryptOperation.dll и тем самым не вызывает конфликт с операционной системой.

На рисунке 15 представлена схема программы, предлагающая разъяснить общие принципы её работы.

Общие принципы работы программы SteganoGraphy V2.2

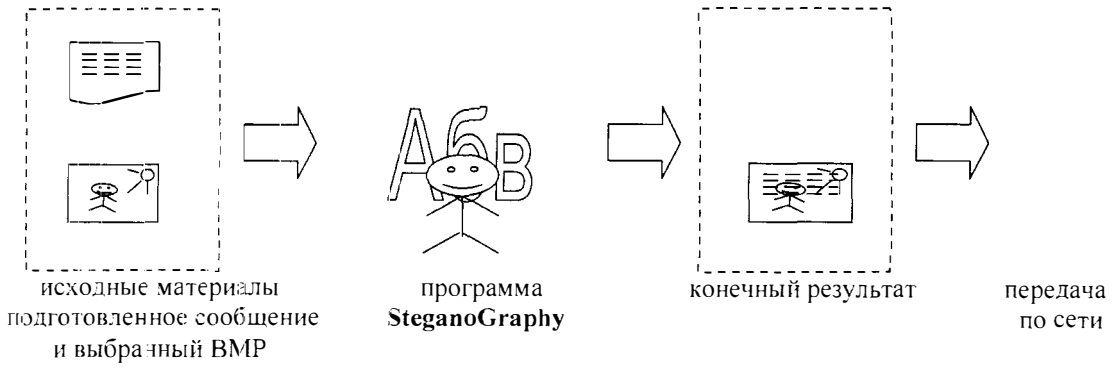


Рисунок 15

Программа SteganoGraphy V2.2 позволяет сохранять текстовую информацию в растровых картинках. Суть программы состоит в следующем: берется текстовый файл и набор файлов рисунков. Далее выбирается подходящий рисунок и на его основе создается второй рисунок, несколько измененный. Причем степень изменения зависит от количества "вшиваемых" данных. При небольшом количестве информации и высокой степени фрагментизации изменений отличить рисунки практически невозможно. А получить обратно текстовую информацию, можно только лишь совместив эти два рисунка, а также имея библиотеку, реализующую алгоритм "склеивания" этих рисунков.

Так выглядит программа в "рабочем" состоянии (Рис. 16). А текст я взял из какого-то файла, валявшегося у меня под рукой.

Интерфейс SteganoGraphy V2.2

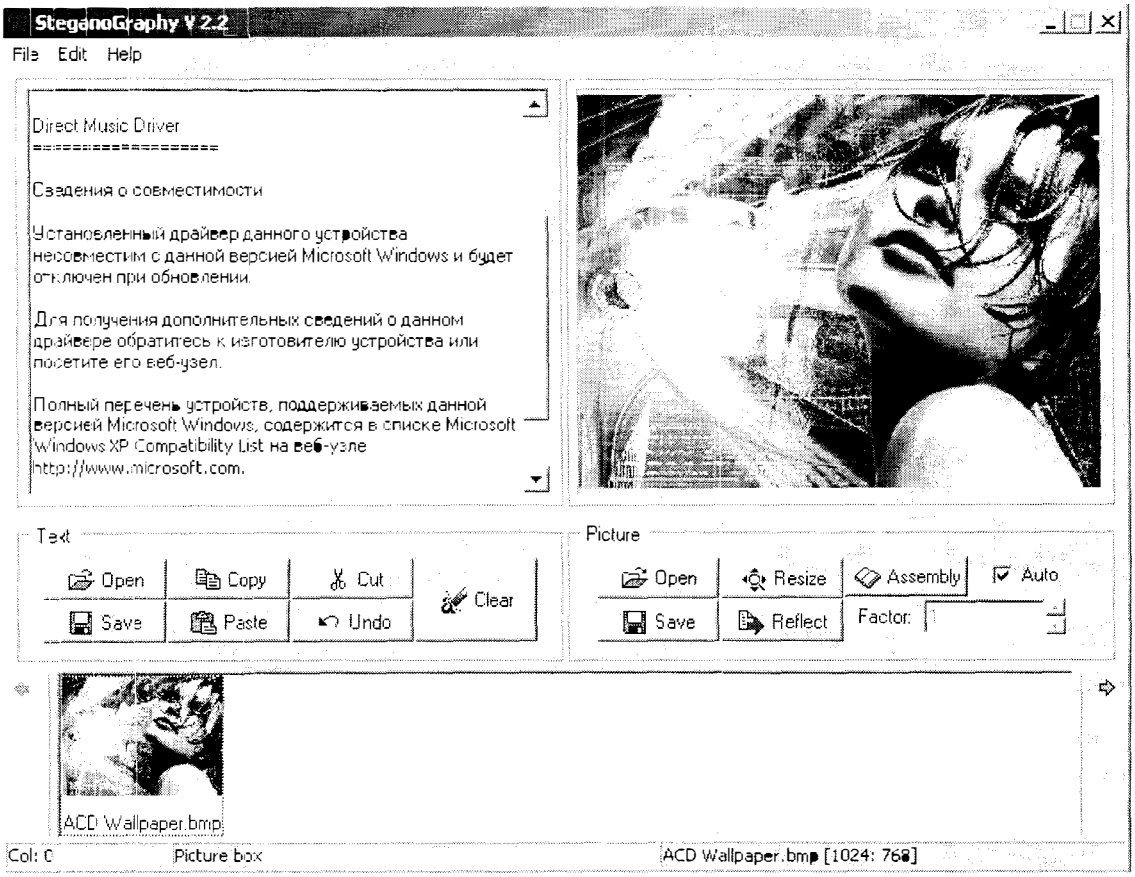


Рисунок 16

Как пользоваться программой. Для того чтобы создать "слепок" существующего рисунка, нужно:

- Загрузить текстовый файл (секция text, кнопка Open)
- Загрузить понравившиеся рисунки (секция Picture, кнопка Open, можно загружать несколько раз, загруженные ранее картинки не пропадут)
- Выделить нужный рисунок в компоненте TGraphGrid
- При желании, отмасштабировать его (кнопка Resize, масштабирует рисунок в соответствии с его пропорциями)
- Нажать на кнопку Reflect. Через несколько секунд (это если картинка большая, а так - через несколько миллисекунд) в компоненте TGraphGrid появится сгенерированное изображение, созданное на основе выбраной картинки и текстового файла. Новая картинка будет иметь имя "Data".
- Сохранить при необходимости нужный файл изображения, выбрав его в компоненте TDrawGrid и нажав на кнопку Save.

Для того чтобы получить текст из двух одинаковых (на первый взгляд) картинок, нужно:

- Загрузить картинку, на основе которой был сделан "слепок"
- Нажать на кнопку Assembly. При этом появится диалоговое окно загрузки изображения и на этот раз Вам надо будет выбрать файл "слепка", т.е. измененную картинку.
- Далее, через некоторое время (в зависимости от количества данных, содержащихся в картинке и размера самой картинки) в компоненте TRichEdit появится этот самый закодированный текст.

Некоторые важные моменты. Выше было сказано, что при больших размерах картинки и при небольшем объеме текстового файла отличить исходную и "слепленную" картинку практически невозможно. Это правильно, но только отчасти. Если взглянуть на изображения, в котором "зашит" большой текстовый файл, то сразу же в глаза бросаются чужеродные пиксели,

распределенные по всему изображению (кстати, коэффициент разброса можно менять) а особенно хорошо эти пиксели видны на рисунке, с однородным фоном. Сравните следующие два рисунка (Рис 17):

Пример использования программы

А – изображение до внедрения текста;

Б – изображение с внедренным текстом.

Рисунок 17

На правом рисунке отчетливо виден шум. Этого отчасти можно было бы избежать, используя неоднородные рисунки с резкими переходами цвета, а также рисунки большего формата. Или можно написать такой хитрый алгоритм кодирования, что второе изображение будет невозможно отличить от первого. В примере вместо увеличения размера рисунка я просто уменьшил количество информации (Рис. 18):

Рисунок 18.

Как работает программа. Программа использует компонент Graph, о который непосредственно выполняет все задачи необходимые для работы с графикой. Прежде всего, он предназначена для программистов Delphi, работающих с графикой. Представленные компоненты имеют такие возможности, как:

1. работа с прозрачностью;
2. быстрая и гибкая замена необходимых цветов;
3. улучшенный компонент TDrawGrid;
4. работа с массивами картинок (в т.ч. быстрое восстановление картинки, после того как она была изменена, быстрое сохранение и загрузка картинок, различные процедуры для их обработки (поддерживаются два формата: bmp и jpg);
5. некоторые полезные дополнения, например, такие как движущийся текст на неоднородном фоне - полезно для таких случаев, как создание диалогового окна About.

А теперь рассмотрим вышеперечисленные пункты немного подробнее. Но для начала, несколько слов о принципах работы этих компонентов. Прежде всего, они построены на основе массива объектов типа TBitmap. В 32 битных изображениях цвет кодируется 3 байтами - по одному на синий, зеленый и красный (стандартный набор RGB), плюс добавляется один байт на альфа канал. В этот альфа канал записывается информация о том, следует ли при вставке одной картинки в другую использовать соответствующий пиксель (или игнорировать его). То есть по идее он может принимать два значения: 0 и 1. В этих компонентах использован несколько иной принцип использования альфа канала. Туда записывается число, которое представляет собой коэффициент, используемый при расчете двух пикселей двух различных картинок. То есть, если, скажем, альфа канал равен 127, то результирующий пиксель будет являться средним арифметическим двух других пикселей. Значение альфа канала, равное 0 соответствует полной непрозрачности, а равное 255 - полной

прозрачности вставляемой картинки. Процедура SetAlphaChannel устанавливает заданное значение альфа канала для всех пикселей картинки.

Замена цветов. Также в компонент TGraph добавлено несколько функция для замены цветов. Все эти процедуры работают только с 24 битными рисунками.

procedure ChangeColor(ChangeColorProc: TChangeColorProc; Bitmap: TBitmap; Color: TColo; TChangeColorProc = procedure(var R, G, B: Byte);

Наверное, самая простая процедура. Процедура TChangeColorProc вызывается каждый раз при замене очередного пикселя.

procedure ChangeColorEx(Bitmap: TBitmap; OldColor, NewColor: TColor; ChangingType: TChangingType);

Замена цветом NewColor происходит, когда значение заменяемого пикселя оказывается между значениями OldColor и NewColor если ChangingType = ctEqual, если ChangingType = ctNotEqual, то замена цвета происходит, когда значение заменяемого пикселя оказывается вне этого диапазона. Во всех последующих процедурах этот параметр имеет тот же самый смысл, поэтому я не буду описывать его действие.

procedure ChangeColorRange(Bitmap: TBitmap; IntensityRecLo, IntensityRecHi, NewIntensityRec: TIntensityRec; ChangingType: TChangingType);

TIntensityType = (itRed, itGreen, itBlue);

TIntensityRec = array[TIntensityType] of Byte;

Как видно выше, тип TIntensityRec представляет собой не что иное, как 24 битный пиксель. Действие этой процедуры аналогично процедуре ChangeColorEx, за исключением того, что эта немного более гибкая, что ли.

procedure ChangeColorRange(Bitmap: TBitmap; IntensityType: TIntensityType; IntensityLo, IntensityHi: Byte; NewIntensityRec: TIntensityRec; ChangingType: TChangingType);

Самая быстрая процедура этого типа. Производит замену пикселя по одной из его составляющий: синему, зеленому или красному цвету. Соответственно IntensityLo и IntensityHi это диапазон одного из составляющих

цвета пикселей, по границам которого производится замена пикселем NewIntensityRec.

Вот, собственно все процедуры созданные для замены цвета. Вместе они представляют из себя довольно гибкое средство для работы с графикой. Еще одно достоинство - они работают достаточно быстро. Это касается также и других процедур, например на компьютере PentiumII Celeron 633 расчет прозрачности двух картинок (обе были конечно же 32 битные, размером 1024x768) занимал не более 500 миллисекунд.

Компонент TgraphGrid. Следующее, что мы рассмотрим, это компонент TGraphGrid, созданный на основе компонента TDrawGrid. Этот компонент способен отображать картинки с текстом. Чтобы картинки отображались в компоненте в порядке из загрузки, следует установить свойство DefaultOrder в True. Вообще, это не лучший способ для отображения картинки. В компоненте TDrawGrid есть свойство:

```
RefArray: TIntArray;
TIntArray = array of Integer;
```

Оно представляет из себя массив ссылок. Поясню на примере. Пусть у на прорисовывается ячейка к координатами Col = 2, Row = 1, где в компоненте TDrawGrid 10 столбцов. В таком случае индекс ячейки (ее порядковый номер) будет равен 12 (нумерация столбцов и рядов начинается с нуля, с нуля также начинается нумерация индекса ячейки). При этом RefArray[12] = 1. В таком случае в ячейке Col = 2, Row = 1 будет прорисовываться картинку с индексом 1. При отсутствии ссылки не другую ячейку RefArray[N] = -1. В таком случае, при условии DefaultOrder = False в ячейке Col = 2, Row = 1 прорисовывалось бы именно картинка с индексом 12. Для более удобного доступа к массиву ссылок есть свойство:

```
property Reference[Col, Row: Integer]: TGridCoord;
        TGridCoord = record
          Col, Row: Integer;
        end;
```

Для задания переменной типа есть функция:

function GridCoord(Col, Row: Integer): TGridCoord;

Рассмотрим теперь событие, возникающее при прорисовки картинки в ячейке:

TDrawPictureEvent = procedure(Sender: TObject; Rect: TRect; Index, Col, Row: Integer; var X, Y: Integer; Picture: TBitmap; var Background: TBitmap; var Continue: Boolean) of object;

Параметры X и Y представляют собой координаты левого верхнего угла картинки (по умолчанию равны 0), которые можно изменять, Picture: собственно, сама картинка, Background картинка, по умолчанию равная nil. Если присвоить этому параметру значение, то картинка Background будет представлять собой фон ячейки. Также, если картинка Background по размерам меньше, ячейки, то она автоматически будет размножена. Также в компоненте TGraphGrid есть ряд свойств, которые, я думаю, объяснять не нужно, так как они интуитивно понятны (например, свойство Scale - для автоматического масштабирования картинки и т.д.)

И, наконец, последний пункт - дополнение в виде движущегося текста. В компоненте TGraph также еще есть одна функция Appearance, но о ней я пока рассказывать не буду.

procedure TGraph.AnimatedText(Canvas: TCanvas; Application: TApplication; Source: TBitmap; List: TStrings; Alignment: TAlignment; X, Y, Decrement: Integer; Delay: LongWord);

Свойство Canvas представляет собой объект типа TCanvas, на котором и будет осуществляться прорисовка. Application служит для вызова функции ProcessMessages. Source представляет собой фоновую картинку, Alignment - способ выравнивания текста, X, Y, координаты, по ним определяется границы движения текста, Decrement величина, определяющая через сколько пикселей текст будет перерисован. Delay - соответственно, задержка текста в миллисекундах. Помимо этого есть еще вспомогательные функции, в нем появилось две новых функции:

1. function ReflectData(BaseBitmap, DataBitmap: TBitmap; List: TStrings; Reflection: TReflectionProc; ScatterType: TScatterType; Factor: Integer = 1): TReflectionResult; virtual;
2. function AssemblyData(BaseBitmap, DataBitmap: TBitmap; Assembling: TAssemblingProc): Boolean; virtual;

Первая функция создает изображение, содержащее информацию; вторая - получает информацию из двух изображений, как я рассказывал выше. Параметры функции ReflectData:

- *BaseBitmap* - Базовое изображение, на его основе создается изображение, содержащее информацию.
- *DataBitmap* - Изображение, содержащее информацию.
- *Reflection* - Процедура, которая изменяет соответствующий байт исходного изображения и формирует DataBitmap. Эта процедура отвечает за сохранение информации во второй картинке. Но о ней - чуть позже.
- *ScatterType* - Тип распределения информации Может принимать два значения: stGiven и stEvenly. По умолчанию в программе установлено значение stEvenly. Это означает, что вся информация будет равномерно распределена по всей картинке. Если так, то задавать значение следующего параметра Factor не нужно. Если ScatterType установить в stGiven, то распределение информации зависит от коэффициента распределения.
- *Factor* - Коэффициент распределения. Если параметр ScatterType равен stGiven, то фактически Factor означает через, сколько байт до начала картинки (не совсем от начала - первые восемь байт идут на длину записываемого текста и на этот коэффициент) будет вписан следующий код символа текста. В программе этот коэффициент задается в компоненте TEdit.

Теперь несколько слов об используемой функции Reflection. Эту функцию я вынес в библиотеку. Эта библиотека является специфическим ключем, паролем для соединения двух картинок. В нее могут быть заложены

самые разные методы шифрования, в данный же момент исполь3зуется самый примитивный: при шифровании код очередного символа просто прибавляется к значению из картинки BaseBitmap, при расшифровке - наоборот. Эта процедура имеет тип:

TReflectionProc = procedure(var Value: Byte; Text: string; Index: Integer);

Параметры функции Reflection:

- Value - Величина из BaseBitmap. Именно это значение шифруется.
- Text - Текст в виде единой строки.
- Index - Текущий символ в тексте, который должен быть зашифрован. В этой программе код этого символа складывается с параметром Value при зашифровке.

В процедуре AsseblyData происходит обратный процесс, выделение данных на основе двух изображений. Параметры функции AsseblyData:

- BaseBitmap - Базовое изображение.
- DataBitmap - Изображение, содержащее информацию.
- Assembling - Процедура, формирующая текстовые данные.

Процедура Assembling имеет следующий тип:

TAssemblingProc = procedure(BaseValue, DataValue: Byte);

Ее параметры:

- BaseValue - Текущая величина из BaseBitmap.
- DataValue - Текущая величина из DestBitmap.

В этой программе каждое значение символа высчитывается путем вычитания DataValue из BaseValue.

И последнее что стоит сказать так это то, что в случае если даже взять контейнер размером 1024x768 (что уже не мало), но при этом сообщение будет очень емким, то возникнет проблема, которая очевидна на рисунке 19.

Избыточность сообщения в "контейнере"

Рисунок 19

Так как объем сообщения составляет 10% от размера "контейнера", то применение блочной упаковки не представляется возможным. И к тому же структура картинки явно изменилась.

И последнее, не стоит результат конвертировать в JPG формат. Мы рискуем серьезно повредить текст. Происходит это из-за того, что JPG является сжатым BitMap'ом. И при обратной трансформации мы теряем сообщение. Сохранение идет с потерей части данных, что делает сообщение практически нечитаемым.

На этих рекомендациях пожалуй и закончим наши первые шаги в Стеганографию. А применение данного продукта найдет своих пользователей.

4. МАРКЕТИНГОВОЕ ИССЛЕДОВАНИЕ

Наше время характеризуется бурной деятельностью, как политической, так и экономической. А отношения между людьми часто называются рыночными. И за небольшой срок общество успело накопить массу секретов друг от друга. По этому мной было проведено маркетинговое исследование на предмет необходимости программных продуктов подобного рода.

Среда респондентов: частные предприятия малого и среднего бизнеса, банковские структуры.

Респонденты:

1. АО «*БанкТуранАлем*» г. Павлодар, ул. Советов 22тел.: 51-07-04, 51-08-02
2. ТОО "*Ralot Ltd*" г. Алматы, ул. Пушкина 129, офис 1тел.: 8(3272)507878. e-mail: ralot-kazakhstan@itt.kz
3. ОАО "*Казэнергокабель*"г. Павлодар, ул. Советов 56 офис 1 тел.: 39-66-53, 39-65-57
4. ТОО "*Завод металлоизделий*" г. Экибастуз, ул. Новоселов 64 тел.: 5-15-82, 5-15-83
5. Банк АО "*Альянс Банк*" г. Павлодар, ул. Торайгырова 64 тел.: 55-39-34
6. ТОО"*ПАВЛОДАРЭКСПЕРТИЗА*"г. Павлодар, ул. Торайгырова 95, офис 1
7. ТОО фирма "*РОСКОМ*"г. Павлодар, ул. Кривенко 49, офис 17 тел.: 51-07-88
8. ОАО "*Казтранссервис*"г. Павлодар, ул. Советов 1/1 тел.: 32-15-64

Были заданны следующие вопросы:

1. необходимость защиты информации;
2. имеющийся уровень защиты информации;
3. необходимость повышения уровня защиты;
4. имеет ли для Вас интерес наше предложение;
5. приемлемая цена ПО.

На что получил следующие результаты. Необходимость защиты понимают все респонденты. Уровень защиты в основном средний, за исключением ОАО "КазЭнергоКабель" которые утверждают что их уровень защищенности максимален. Необходимость повышения уровня защищенности испытывают 80%. Заинтересовалось предложением 60%. С ценовой политикой в 10-20 у.е.

Вывод: в современном мире существует масса частных секретов нуждающихся в защите. На что тратятся не малые средства. Однако вопреки высокому проценту заинтересованных респондентов, есть тенденция к смещению в сторону сторонних разработчиков. И цена за ПО составляет 15 у.е.

ЗАКЛЮЧЕНИЕ

Мной были рассмотрены различные форматы файлов и алгоритмы их компрессии (текстовые, графические и мультимедийные файлы). В поисках подходящего "контейнера" могут выступать все форматы. Однако я отдаю предпочтение формату BitMap. Это золотая середина среди многообразия вариантов.

Пробираясь сквозь тернистые модули Delphi7 считаю что добился результата поставленного себе в требовании. Мне удалось найти способ синтеза двух разных форматов, таких как BitMap (BMP) и текстовый формат (DOS, ISO, KOI8, Windows). Созданная мной программа под названием *SteganoGraphyV2.2* позволяет, используя любое 24-х битное изображение, имеющий формат битовой карты, упаковывать в него текстовую информацию. По результатам исследования выявлено что, емкость "контейнера" составляет до 10% от его собственного размера. Это один из лучших показателей среди рассмотренных вариантов. Помимо этого был разработан собственный компонент в Delphi, для работы с битовой информацией.

В рамках моей программы я добился поставленной цели и смог воспользоваться "излишками" формата BMP. Где в качестве носителя скрытого сообщения выступает так называемый альфа-канал. Способный нести в себе значение от 0 до 255 он отлично вмещает в себя любой диапазон символьных кодировок от латиницы до кириллицы расширенной казахским алфавитом.

В планах разработка своего алгоритм сжатия по принципу jpg, не искажающего смысл упакованного сообщения. Тем самым повысить процент емкости "контейнера" до 30% за счет сжатия BitMap обходя зоны с измененным альфа-каналом

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Standardisation of Group 3 Facsimile apparatus for document transmission. - CCITT Recommendations. Fascicle VII.2. T.4. 1980.
2. Александров В.В., Горский Н.Д. «Представление и обработка изображений: рекурсивный подход» - Л-д.: Наука 1985, 190 стр.
3. Климов А.С. «Форматы графических файлов». - С.Петербург, Изд. «ДиаСофт» 1995.
4. Ватолин Д.С. «MPEG - стандарт ISO на видео в системах мультимедиа»- Открытые системы. Номер 2. Лето 1995
5. Ватолин Д.С. «Тенденции развития алгоритмов архивации графики»- Открытые системы. Номер 4. Зима 1995
6. Ватолин Д.С. «Алгоритмы сжатия изображений» - ISBN 5-
7. 89407-041-4 М.: Диалог-МГУ, 1999.
8. Добеши И. «Десять лекций по вейвлетам» - Открытые системы. Номер 4. Зима 1995
9. Мищенко, под ред. А.П.Петухова. - М.: Ижевск 2001, 464 стр.
10. Яншин В.В. «Анализ и обработка изображений (принципы и алгоритмы)» - М.: Машиностроение 1995
11. Павлидис Т. «Алгоритмы машинной графики и обработка изображений» - М.: Радио и связь 1986, 400 стр.
12. Претт У. «Цифровая обработка изображений» в двух томах - М.: Мир 1982, 790 стр.
13. Розеншелъд А. «Распознавание и обработка изображений» - М.:Мир 1972,232 стр.
14. Ярославский Л.П. «Введение в цифровую обработку изображений» - М.:Сов.
15. Грин Н., Стаут У. Физиология – Москва: Изд.Мир
16. Глухов С.А., Деревягин С.И. Стеганография, или как мне сохранить тайну - Вестник Павлодарского Университета 2004 – 183с.
17. Сван Том «Форматы файлов Windows» - М. «Бином», 1995

18. Hamilton E.«JPEG File Interchange Format» - Version 1.2. September 1, 1992, San Jose CA: C-Cube Microsystems, Inc.

19. Aldus Corporation Developer's Desk. «TIFF - Revision 6.0, Final», June 3, 1992