Д. Ватолин, А. Ратушняк,
М. Смирнов, В. Юкин

Методы сжатия данных

Алгоритмы сжатия изображений

Содержание книги:

ISBN 5-86404-170-X
2002

Раздел 2. Алгоритмы сжатия изображений

Введение

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 несжатых 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. Т.е. 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 различных 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.

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 256 цветов.

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 цветопредставления* 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 цветопредставления, такие, как CMYK, CIE XYZccir60-1, YVU, YCrCb и т.п. Ниже мы увидим, как они используются при сжатии изображений с потерями.

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.

> **Итог:** Нет смысла говорить о том, что какой-то алгоритм сжатия лучше другого, если мы не обозначили **классы изображений**, на которых сравниваются наши алгоритмы.

Классы приложений

Примеры приложений, использующих алгоритмы компрессии графики

Рассмотрим следующую простую классификацию приложений, использующих алгоритмы компрессии:

1) **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.

2) **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).

3) **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 потре-

боваться *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.

> **Итог:** Нет смысла говорить о том, что какой-то конкретный алгоритм компрессии лучше другого, если мы не обозначили **класс приложений**, относительно которого мы эти алгоритмы собираемся сравнивать.

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...
3. **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.
4. **Высокая скорость декомпрессии.** Достаточно универсальное требование, актуальное для многих приложений. Однако можно привести примеры приложений, где время декомпрессии далеко не критично.
5. **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.
6. **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 реализа-

ция указанного требования путем записывания в начало изображения его уменьшенной копии заметно ухудшит степень компрессии.

7. **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.
8. **Учет специфики изображения.** Более высокая степень архивации для класса изображений, которые статистически чаще будут применяться в нашем приложении. В предыдущих разделах это требование уже обсуждалось.
9. **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.
10. **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.

> **Итог:** На практике для каждой задачи мы можем сформулировать **набор приоритетов из требований,** изложенных выше, который и определит наиболее подходящий в наших условиях алгоритм (либо набор алгоритмов) для ее решения.

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 (требование 6),

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 декодирова-

нии. 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.

Методы обхода плоскости

Задача обхода плоскости возникает при обработке двухмерных данных. Цель: создание одномерного массива D из двухмерного S. Причем если предполагается последующее сжатие D, то желательно создавать его так, чтобы «разрывов» было как

можно меньше: каждый следующий элемент D_i, заносимый в D на i–ом шаге, является соседним (в плоскости) для предыдущего, занесенного в D на (i-1)–ом шаге, D_{i-1}.

Змейка (зигзаг-сканирование)

Обход массива S начинается с одного угла плоскости, заканчивается в противоположном по диагонали. Например, из левого верхнего в правый нижний:

1	2	6	7	15	16
3	5	8	14	17	24
4	9	13	18	23	25
10	12	19	22	26	29
11	20	21	27	28	30

На иллюстрации показан порядок выборки элементов из плоскости (с последующим занесением в одномерный массив). Значение из ячейки массива S, помеченной на рисунке номером i, заносится в $D[i]$.

Змейка выгодна в случаях, когда в одном из углов «особенность» — например, сосредоточены самые крупные коэффициенты. Применяется в алгоритме JPEG для обхода квадрантов (размером 8x8 точек).

Обход строками

Самый тривиальный метод. Именно он используется в самых распространенных графических форматах (BMP, TGA, RAS…) для хранения элементов изображений.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30

В варианте строк с разворотами для каждой второй строки делаем выборку в обратном направлении:

1	2	3	4	5	6
12	11	10	9	8	7
13	14	15	16	17	18
24	23	22	21	20	19
25	26	27	28	29	30

Точек «разрыва» нет, в отличие от варианта без разворотов. Совершенно аналогично можно делать обход столбцами.

Упражнение: Нарисуйте пример обхода плоскости столбцами с разворотами.

Обход полосами

Чаще всего сжатие лучше, если каждая область двухмерного массива *S* не рассредоточена (равномерно «размазана») по всему одномерному *D*, а сконцентрирована в *D* компактно. В случае обхода строками понятие «области» отсутствует: каждый элемент считается «областью». Пытаясь обходить плоскость квадратами размером NxN, приходим к идее обхода горизонтальными «полосами» шириной N:

1	4	7	10	13
2	5	8	11	14
3	6	9	12	15
16	19	22	25	28
17	20	23	26	29
18	21	24	27	30

В данном примере ширина полосы N=3. Если N=1, получаем обход строками.

Полосами с разворотами

То же самое, но с разворотами и столбцов внутри полос, и направлений самих полос:

1	6	7	12	13
2	5	8	11	14
3	4	9	10	15
28	27	22	21	16
29	26	23	20	17
30	25	24	19	18

Разрывов опять нет, но теперь еще и каждая точка принадлежит **области**, записанной в D **компактно**, без разрывов: ее элементы расположены внутри одного интервала ($D[i]$, $D[i+1]$, …, $D[i+j]$), и элементов из других областей внутри этого интервала нет. Примеры таких областей — каждый из четырех углов размером 3x3 элемента.

Упражнение: Нарисуйте схему обхода квадрата 7x7 полосами шириной 3 с разворотами. Какой вариант лучше: 3+3+1 или 3+2+2? Какие области заносятся в D компактно?

Обход решетками

Для первой порции берем элементы из каждого N-го столбца каждой M-ой строки. Для второй – то же, но со сдвигом на один столбец. Так же и для следующих, а затем – со сдвигом на одну,

две, (M–1) строки. Например, если M=N=2, то имеем четыре порции:

1	13	2	14	3	15	4	16
25	37	26	38	27	39	28	40
5	17	6	18	7	19	8	20
29	41	30	42	31	43	32	44
9	21	10	22	11	23	12	24
33	45	34	46	35	47	36	48

То есть плоскость разбивается на прямоугольники размером MxN, задается обход плоскости прямоугольниками, а также обход внутри самих прямоугольников, и далее делается «одновременный» обход по каждому из них: сначала выбираются их первые элементы, затем вторые, третьи, и так далее, до последнего.

Обход решетками с учетом значений элементов

После того как обработаны первые элементы прямоугольников (в нашем примере – квадратов 2x2), если предположить, что они являются атрибутами своих областей, выгодно (для улучшения сжатия) группировать области с одинаковыми атрибутами, то есть с одинаковыми значениями этих первых элементов.

Допустим, атрибуты распределены так:

R		G		G		G	
L		R		G		G	
L		L		R		R	

Тогда имеет смысл дальше действовать так:

R	13	G	25	G	28	G	31
15	14	27	26	30	29	33	32
L	40	R	16	G	34	G	37
42	41	18	17	36	35	39	38
L	43	L	46	R	19	R	22
45	44	48	47	21	20	24	23

То есть сначала обходим квадраты с атрибутом «R», затем с атрибутом «G», и наконец с «L».

Контурный обход

Часть элементов принадлежит одной группе, часть – другой, причем контур задан:

1	1	1	1	1	1	1	1
1	1	2	2	1	1	1	1
1	2	2	2	2	1	1	1
1	2	2	2	2	2	1	1
1	1	1	2	1	1	1	1
1	1	1	1	1	1	1	1

36 элементов – из группы «1», а 12 – из группы «2».

Очевидно, что имеет смысл отдельно оформить элементы группы «1»:

1	29	28	27	26	22	21	31
2	30			25	23	20	32
3					24	19	33
4						18	34
5	8	9		13	14	17	35
6	7	10	11	12	15	16	36

и затем точно так же остальные элементы, принадлежащие группе «2».

КОНТУРНЫЙ ОБХОД С НЕИЗВЕСТНЫМИ КОНТУРАМИ

Рассмотрим предыдущий пример, то есть такое же распределение элементов групп по плоскости, но изначально, при начале обхода плоскости, это распределение неизвестно. Будем действовать таким методом:

1	44	41	40	35	34	29	28
2	43	42	39	36	33	30	27
3	**45**	**46**	38	37	32	31	26
4	9	10	**47**	**48**	19	20	25
5	8	11	14	15	18	21	24
6	7	12	13	16	17	22	23

Обходим плоскость «столбцами с разворотами», и, обнаруживая элемент другой группы (в элементах 9, 10, 14…), также делаем разворот на 180 градусов. Затем (шаги 45…48) обходим оставшуюся часть плоскости, содержащую (предположительно) элементы другой группы.

В итоге имеем:

- среди первых 36-и элементов 4 из группы «2», а 32 из группы «1».
- из последних 12-и элементов 8 из группы «2», а 4 из группы «1».

В первой части 4/36=1/9 «исключений», во второй 4/12=1/3.

А если бы делали просто обход «столбцами с разворотами»:

1	12	13	24	25	36	37	48
2	11	14	23	26	35	38	47
3	10	15	22	27	34	39	46
4	9	16	21	28	33	40	45
5	8	17	20	29	32	41	44
6	7	18	19	30	31	42	43

В итоге:

- среди первых 33-х элементов 12 из группы «2», а 21 из группы «1».
- из последних 15-и элементов все из группы «1».

То есть в большей части получившегося блока – более чем 1/3 «исключений».

«Квадратная змейка»

Рекурсивный метод для квадратных областей.

Если принять левый верхний элемент за первый, то **для квадрата 2x2** возможны два варианта обхода без разрывов:

1	4
2	3

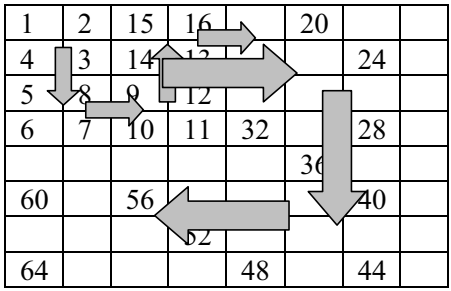
и

1	2
4	3

То есть либо первый переход внутри квадрата был сверху **вниз**, тогда пятым шагом будет переход к левому верхнему элементу квадрата **справа** (или к правому нижнему элементу квадрата **сверху**). Либо наоборот: первый переход внутри квадрата был **вправо**, тогда пятым шагом будет переход к квадрату **снизу** (или **слева**). Переформулируем так:

- если нужно выйти **к правому** (или верхнему) квадрату, то первый шаг – **вниз**, если **к нижнему** (или левому), то первый шаг – **вправо**;
- пройденным путем однозначно задается, к какому квадрату нужно выйти в каждый конкретный момент;
- только в самом начале есть выбор одного из двух вариантов обхода.

Например:



Первый шаг внутри квадрата 2x2 был вправо – значит в результате обхода этого квадрата 2x2 мы должны выйти к нижнему квадрату 2x2;

первый шаг перехода от 2x2 к 2x2 внутри 4x4 был вниз – значит мы должны выйти к правому 4x4;

первый переход внутри квадрата 16x16 был вправо – значит в результате обхода 16x16 – должны прийти к нижнему, и так далее.

Разрывов нет, и каждый элемент принадлежит области, записанной компактно.

Упражнение: Определите, какие числа должны быть в незаполненных клетках примера. Нарисуйте другой пример: при выборе второго элемента квадрата делаем переход не слева направо, а сверху вниз.

Для квадрата 3x3 можно взять такие два «шаблона»:

1	4	5
2	3	6
9	8	7

и

1	2	9
4	3	8
5	6	7

Первое правило будет «противоположным» первому правилу случая 2x2, но остальные два – такие же:

- если нужно выйти **к правому** (или верхнему) квадрату, то первый шаг – **вправо**, если **к нижнему** (или левому), то первый шаг – **вниз**;
- пройденным путем однозначно задается, к какому квадрату нужно выйти в каждый конкретный момент;
- только в самом начале есть выбор одного из двух вариантов.

Но для 3x3 есть еще и такие варианты:

1	6	7
2	5	8
3	4	9

и

1	2	3
6	5	4
7	8	9

Видно, что они совершенно эквивалентны, то есть взаимозаменяемы, поскольку в обоих случаях мы выходим в правый нижний угол. Точно так же эквивалентны и два варианта обхода ими квадрата 9x9:

1					18	19		
		9	10					27
		46	45					28
54					37	36		
55					72	73		
		63	64					81

и

1					54	55		
		9	46					63
		10	45					64
18					37	72		
19					36	73		
		27	28					81

Совершенно аналогично для 5x5, и затем 25x25 и так далее.

Если требуется обойти квадрат размера NxN, сначала определяем, произведением каких простых чисел является N, затем – задаем порядок этих чисел в произведении (а для каждого нечётного множителя – еще и направление обхода). Таким образом будет задан процесс обхода.

Если K, равное числу двоек в произведении, больше одного: K>1, то сторона самого мелкого квадрата должна быть 2^{K-1} или 2^K элементов. Например, если K=3, то обход без разрывов 2x3x2x2 (от самого мелкого 2x2 к 6x6, затем к 12x12 и, наконец, к 24x24) невозможен:

		63	64					9	10		
	59			68			5			14	
55					72	1					18
54					37	36					19
	50			41			32			23	
		46	45					28	27		
…	…	…	…	…	…	73	…	…	…	…	…

Каждая ячейка этой таблицы – квадрат 2x2; нижние 5 строк пропущены: перейти к двум нижним квадратам 12x12 без разрыва не сможем.

Других ограничений при обходе «квадратной змейкой» нет.

Каждый квадрат со стороной L>2 можно обходить обычной змейкой, а не «квадратной». Это выгодно в том случае, если наиболее различающиеся элементы сгруппированы в противоположных углах квадрата.

Упражнение: Нарисуйте порядок обхода квадрата 25x25, а затем – квадрата 12x12. Убедитесь, что разрывов нет, и каждый элемент принадлежит компактно записанной области.

Обход по спирали

Обход по спирали довольно тривиален. Строится квадрат 3x3, затем 5x5, затем 7x7, 9x9 и так далее:

43	44	45	46	47	48	49
42	21	22	23	24	25	26
41	20	7	8	9	10	27
40	19	6	1	2	11	28

39	18	5	4	3	12	29
38	17	16	15	14	13	30
37	36	35	34	33	32	31

Если же строить круги радиуса 2, 3, 4 и т.д., неизбежно будут присутствовать точки разрыва. Спираль может быть и сходящейся. Суть ее можно показать с помощью этой же иллюстрации, только направление движения обратное: от 49 к 1. Кроме того, она может быть с разворотами:

1	2	3	4	5	6	7
24	25	40	39	38	37	8
23	26	41	42	43	36	9
22	27	48	49	44	35	10
21	28	47	46	45	34	11
20	29	30	31	32	33	12
19	18	17	16	15	14	13

Направление изменяется в точках, расположенных на диагонали: в данном примере это «25» и «41».

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.

Вопросы для самоконтроля

1) Какие параметры надо определить, прежде чем сравнивать два алгоритма компрессии?
2) Почему некорректно сравнивать временные параметры реализаций алгоритмов компрессии, оптимально реализованных на разных компьютерах? Приведите примеры ситуаций, когда архитектура компьютера дает преимущества тому или иному алгоритму.
3) Предложите пример своего класса изображений.
4) Какими свойствами изображений мы можем пользоваться, создавая алгоритм компрессии? Приведите примеры.
5) Что такое редактируемость?
6) Назовите основные требования приложений к алгоритмам компрессии.
7) Что такое симметричность?
8) Предложите пример своего класса приложений.
9) Приведите примеры аппаратных реализаций алгоритма сжатия изображений, с которыми вам приходилось сталкиваться (повседневные и достаточно новые).
10) Почему высокая скорость компрессии, высокое качество изображений и высокая степень компрессии взаимно противоречивы? Покажите противоречивость каждой пары условий.

Если вы обнаружите ошибки или неточности в данном тексте, просьба сообщить о них авторам по адресу:
compression@graphicon.ru

Ответы к вопросам и упражнениям и исходные тексты программ вы можете найти на
http://compression.graphicon.ru/

Алгоритмы архивации без потерь

Алгоритм RLE

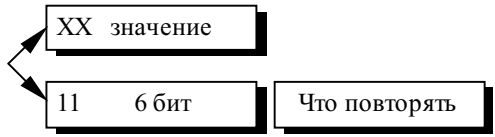
Первый вариант алгоритма

Данный алгоритм необычайно прост в реализации. Групповое кодирование — от английского Run Length Encoding (RLE) — один из самых старых и самых простых алгоритмов архивации графики. 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.

Алгоритм *декомпрессии* при этом выглядит так:

```
Initialization(...);
do {
   byte = ImageFile.ReadNextByte();
   if(является счетчиком(byte)) {
      counter = Low6bits(byte)+1;
      value = ImageFile.ReadNextByte();
      for(i=1 to counter)
         DecompressedFile.WriteByte(value)
   }
   else {
   DecompressedFile.WriteByte(byte)
} while(!ImageFile.EOF());
```

В данном алгоритме признаком счетчика (counter) служат единицы в двух верхних битах считанного файла:



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.

> **Упражнение:** Составьте алгоритм *компрессии* для первого варианта алгоритма RLE.

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.

> **Упражнение:** Предложите два-три примера «плохих» изображений для алгоритма RLE. Объясните, почему размер сжатого файла больше размера исходного файла.

Данный алгоритм реализован в формате PCX. См. пример в приложении.

Второй вариант алгоритма

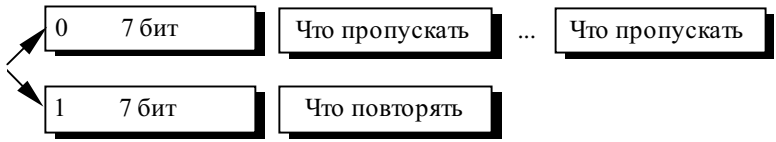
Второй вариант этого алгоритма имеет большую максимальную степень сжатия и меньше увеличивает в размерах исходный файл.

Алгоритм декомпрессии для него выглядит так:

```
Initialization(...);
do {
   byte = ImageFile.ReadNextByte();
   counter = Low7bits(byte)+1;
```

```
   if(если признак повтора(byte)) {
      value = ImageFile.ReadNextByte();
      for (i=1 to counter)
         CompressedFile.WriteByte(value)
   }
   else {
      for(i=1 to counter){
         value = ImageFile.ReadNextByte();
         CompressedFile.WriteByte(value)
         }
} while(!ImageFile.EOF());
```

Признаком повтора в данном алгоритме является единица в старшем разряде соответствующего байта:



Как 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.

Упражнение: Составьте алгоритм компрессии для второго варианта алгоритма RLE.

Похожие схемы компрессии использованы в качестве одного из алгоритмов, поддерживаемых форматом TIFF, а также в формате TGA.

Характеристики алгоритма RLE:

Степени сжатия: Первый вариант: 32, 2, 0,5. Второй вариант: 64, 3, 128/129. (Лучшая, средняя, худшая степени)

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.

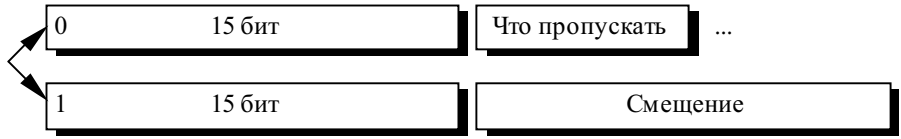
Алгоритм LZW

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.

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. 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.



При этом мы получим увеличение размера файла в худшем случае на 32770/32768 (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.

Упражнение: Предложите другой вариант алгоритма LZ, в котором на пару <счетчик, смещение> будет выделено 3 байта, и подсчитайте основные характеристики своего алгоритма.

Алгоритм LZW

Рассматриваемый нами ниже вариант алгоритма будет использовать дерево для представления и хранения цепочек. Очевидно, что это достаточно сильное ограничение на вид цепочек, и далеко не все одинаковые подцепочки в нашем изображении будут использованы при сжатии. Однако в предлагаемом алгоритме выгодно сжимать даже цепочки, состоящие из 2 байт.

Процесс сжатия выглядит достаточно просто. Мы считываем последовательно символы входного потока и проверяем, есть ли в созданной нами таблице строк такая строка. Если строка есть, то мы считываем следующий символ, а если строки нет, то мы заносим в поток код для предыдущей найденной строки, заносим строку в таблицу и начинаем поиск снова.

Функция InitTable() очищает таблицу и помещает в нее все строки единичной длины.

```
InitTable();
CompressedFile.WriteCode(ClearCode);
CurStr=пустая строка;

while(не ImageFile.EOF()){    //Пока не конец файла
   C=ImageFile.ReadNextByte();
   if(CurStr+C есть в таблице)
           CurStr=CurStr+C;//Приклеить символ к строке
       else {
           code=CodeForString(CurStr);//code-не байт!
           CompressedFile.WriteCode(code);
           AddStringToTable (CurStr+C);
           CurStr=C;     // Строка из одного символа
   }
}
code=CodeForString(CurStr);
CompressedFile.WriteCode(code);
CompressedFile.WriteCode(CodeEndOfInformation);
```

Как говорилось выше, функция `InitTable()` инициализирует таблицу строк так, чтобы она содержала все возможные строки, состоящие из одного символа. Например, если мы сжимаем байтовые данные, то таких строк в таблице будет 256 («0»,

«1», ... , «255»). Для кода очистки (`ClearCode`) и кода конца информации (`CodeEndOfInformation`) зарезервированы значения 256 и 257. В рассматриваемом варианте алгоритма используется 12-битный код, и, соответственно, под коды для строк нам остаются значения от 258 до 4095. Добавляемые строки записываются в таблицу последовательно, при этом индекс строки в таблице становится ее кодом.

Функция `ReadNextByte()` читает символ из файла. Функция `WriteCode()` записывает код (не равный по размеру байту) в выходной файл. Функция `AddStringToTable ()` добавляет новую строку в таблицу, приписывая ей код. Кроме того, в данной функции происходит обработка ситуации переполнения таблицы. В этом случае в поток записывается код предыдущей найденной строки и код очистки, после чего таблица очищается функцией `InitTable()`. Функция `CodeForString()` находит строку в таблице и выдает код этой строки.

Пример:

Пусть мы сжимаем последовательность 45, 55, 55, 151, 55, 55, 55. Тогда, согласно изложенному выше алгоритму, мы поместим в выходной поток сначала код очистки <256>, потом добавим к изначально пустой строке «45» и проверим, есть ли строка «45» в таблице. Поскольку мы при инициализации занесли в таблицу все строки из одного символа, то строка «45» есть в таблице. Далее мы читаем следующий символ 55 из входного потока и проверяем, есть ли строка «45, 55» в таблице. Такой строки в таблице пока нет. Мы заносим в таблицу строку «45, 55» (с первым свободным кодом 258) и записываем в поток код <45>. Можно коротко представить архивацию так:

«45» — есть в таблице;

«45, 55» — нет. Добавляем в таблицу <258>«45, 55». В поток: <45>;

«55, 55» — нет. В таблицу: <259>«55, 55». В поток: <55>;

«55, 151» — нет. В таблицу: <260>«55, 151». В поток: <55>;

«151, 55» — нет. В таблицу: <261>«151, 55». В поток: <151>;

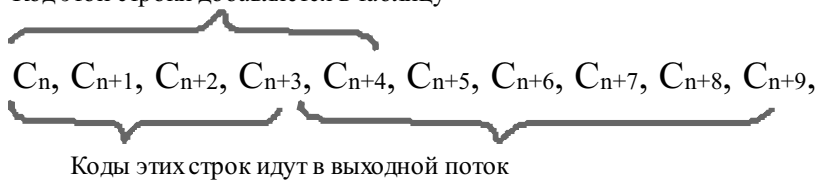
«55, 55» — есть в таблице;

«55, 55, 55» — нет. В таблицу: «55, 55, 55» <262>. В поток: <259>;

Последовательность кодов для данного примера, попадающих в выходной поток: <256>, <45>, <55>, <55>, <151>, <259>.

Особенность LZW заключается в том, что для декомпрессии нам не надо сохранять таблицу строк в файл для распаковки. Алгоритм построен таким образом, что мы в состоянии восстановить таблицу строк, пользуясь только потоком кодов.

Мы знаем, что для каждого кода надо добавлять в таблицу строку, состоящую из уже присутствующей там строки и символа, с которого начинается следующая строка в потоке.



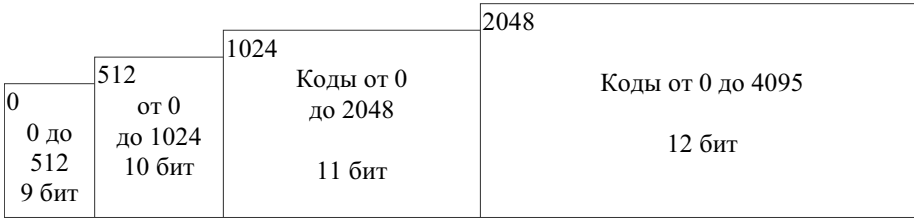
Алгоритм декомпрессии, осуществляющий эту операцию, выглядит следующим образом:

```
code=File.ReadCode();
while(code != CodeEndOfInformation){
   if(code = ClearCode) {
           InitTable();
           code=File.ReadCode();
      if(code = CodeEndOfInformation)
                {закончить работу};
           ImageFile.WriteString(StrFromTable(code));
           old_code=code;
       }
       else {
           if(InTable(code)) {
               ImageFile.WriteString(FromTable(code));
```

```
                AddStringToTable(StrFromTable(old_code)+
                  FirstChar(StrFromTable(code)));
                old_code=code;
            }
            else {
                OutString= StrFromTable(old_code)+
                   FirstChar(StrFromTable(old_code));
                ImageFile.WriteString(OutString);
                AddStringToTable(OutString);
                old_code=code;
            }
        }
}
```

Здесь функция `ReadCode()` читает очередной код из декомпрессируемого файла. Функция `InitTable()` выполняет те же действия, что и при компрессии, т.е. очищает таблицу и заносит в нее все строки из одного символа. Функция `FirstChar()` выдает нам первый символ строки. Функция `StrFromTable()` выдает строку из таблицы по коду. Функция `AddStringToTable()` добавляет новую строку в таблицу (присваивая ей первый свободный код). Функция `WriteString()` записывает строку в файл.

Замечание 1. Как вы могли заметить, записываемые в поток коды постепенно возрастают. До тех пор, пока в таблице не появится, например, в первый раз код 512, все коды будут меньше 512. Кроме того, при компрессии и при декомпрессии коды в таблице добавляются при обработке одного и того же символа, т.е. это происходит «синхронно». Мы можем воспользоваться этим свойством алгоритма для того, чтобы повысить степень компрессии. Пока в таблицу не добавлен 512 символ, мы будем писать в выходной битовый поток коды из 9 бит, а сразу при добавлении 512 — коды из 10 бит. Соответственно декомпрессор также должен будет воспринимать все коды входного потока 9-битными до момента добавления в таблицу кода 512, после чего будет воспринимать все входные коды как 10-битные. Аналогично мы будем поступать при добавлении в таблицу кодов 1024 и 2048. Данный прием позволяет примерно на 15% поднять степень компрессии:



Замечание 2. При сжатии изображения нам важно обеспечить быстроту поиска строк в таблице. Мы можем воспользоваться тем, что каждая следующая подстрока на один символ длиннее предыдущей, кроме того, предыдущая строка уже была нами найдена в таблице. Следовательно, достаточно создать список ссылок на строки, начинающиеся с данной подстроки, как весь процесс поиска в таблице сведется к поиску в строках, содержащихся в списке для предыдущей строки. Понятно, что такая операция может быть проведена очень быстро.

Заметим также, что реально нам достаточно хранить в таблице только пару <код предыдущей подстроки, добавленный символ>. Этой информации вполне достаточно для работы алгоритма. Таким образом, массив от 0 до 4095 с элементами <код предыдущей подстроки; добавленный символ; список ссылок на строки, начинающиеся с этой строки> решает поставленную задачу поиска, хотя и очень медленно.

На практике для хранения таблицы используется такое же быстрое, как в случае списков, но более компактное по памяти решение — хэш-таблица. Таблица состоит из 8192 (2^{13}) элементов. Каждый элемент содержит <код предыдущей подстроки; добавленный символ; код этой строки>. Ключ для поиска длиной в 20 бит формируется с использованием двух первых элементов, хранимых в таблице как одно число (key). Младшие 12 бит этого числа отданы под код, а следующие 8 бит под значение символа.

В качестве хэш-функции при этом используется:

```
Index(key)= ((key >> 12) ^ key) & 8191;
```

Где >> — побитовый сдвиг вправо (key >> 12 — мы получаем значение символа), ^ — логическая операция побитового исключающего ИЛИ, & логическое побитовое И.

Таким образом, за считанное количество сравнений мы получаем искомый код или сообщение, что такого кода в таблице нет.

Подсчитаем лучшую и худшую степень сжатия для данного алгоритма. Лучшее сжатие, очевидно, будет получено для це-

почки одинаковых байт большой длины (т.е. для 8-битного изображения, все точки которого имеют, для определенности, цвет 0). При этом в 258 строку таблицы мы запишем строку «0, 0», в 259 — «0, 0, 0», ... в 4095 — строку из 3839 (=4095-256) нулей. При этом в поток попадет (проверьте по алгоритму!) 3840 кодов, включая код очистки. Следовательно, посчитав сумму арифметической прогрессии от 2 до 3839 (т.е. длину сжатой цепочки) и поделив ее на 3840*12/8 (в поток записываются 12-битные коды), мы получим лучшую степень сжатия.

Упражнение: Вычислить точное значение лучшей степени сжатия. Более сложное задание: вычислить ее с учетом замечания 1.

Худшее сжатие будет получено, если мы ни разу не встретим подстроку, которая уже есть в таблице (в ней не должно встретиться ни одной одинаковой пары символов).

Упражнение: Составить алгоритм генерации таких цепочек. Попробовать сжать полученный таким образом файл стандартными архиваторами (zip, arj, gz). Если вы получите сжатие, значит алгоритм генерации написан неправильно.

В случае, если мы постоянно будем встречать новую подстроку, мы запишем в выходной поток 3840 кодов, которым будет соответствовать строка из 3838 символов. Без учета замечания 1 это составит увеличение файла почти в 1.5 раза.

LZW реализован в форматах GIF и TIFF.

Характеристики алгоритма LZW:
Степени сжатия: Примерно 1000, 4, 5/7 (Лучшее, среднее, худшее сжатие). Сжатие в 1000 раз дости-

гается только на одноцветных изображениях размером кратным примерно 7 Мб.

Класс изображений: Ориентирован LZW на 8-битные изображения, построенные на компьютере. Сжимает за счет одинаковых подцепочек в потоке.

Симметричность: Почти симметричен, при условии оптимальной реализации операции поиска строки в таблице.

Характерные особенности: Ситуация, когда алгоритм увеличивает изображение, встречается крайне редко. LZW универсален — именно его варианты используются в обычных архиваторах.

Алгоритм Хаффмана

Алгоритм Хаффмана с фиксированной таблицей CCITT Group 3

Классический алгоритм Хаффмана был рассмотрен в первой части данной книги. Он практически не применяется к изображениям в чистом виде, а используется как один из этапов компрессии в более сложных схемах.

Близкая модификация алгоритма используется при сжатии черно-белых изображений (один бит на пиксел). Полное название данного алгоритма CCITT Group 3. Это означает, что данный алгоритм был предложен третьей группой по стандартизации Международного Консультационного Комитета по Телеграфии и Телефонии (Consultative Committee International Telegraph and Telephone). Последовательности подряд идущих черных и белых точек в нем заменяются числом, равным их количеству. А этот ряд, уже в свою очередь, сжимается по Хаффману с фиксированной таблицей.

***Определение**:* Набор идущих подряд точек изображения одного цвета называется *серией*. Длина этого набора точек называется *длиной серии*.

В таблице, приведенной ниже, заданы два вида кодов:

• *Коды завершения серий* — заданы с 0 до 63 с шагом 1.

• *Составные (дополнительные) коды* — заданы с 64 до 2560 с шагом 64.

Каждая строка изображения сжимается независимо. Мы считаем, что в нашем изображении существенно преобладает белый цвет, и все строки изображения начинаются с белой точки. Если строка начинается с черной точки, то мы считаем, что строка начинается белой серией с длиной 0. Например, последовательность длин серий 0, 3, 556, 10, ... означает, что в этой строке изображения идут сначала 3 черных точки, затем 556 белых, затем 10 черных и т.д.

На практике в тех случаях, когда в изображении преобладает черный цвет, мы инвертируем изображение перед компрессией и записываем информацию об этом в заголовок файла.

Алгоритм компрессии выглядит так:

```
for(по всем строкам изображения) {
       Преобразуем строку в набор длин серий;
       for(по всем сериям) {
        if(серия белая) {
           L= длина серии;
           while(L > 2623) { // 2623=2560+63
              L=L-2560;
              ЗаписатьБелыйКодДля(2560);
           }
           if(L > 63) {
           L2=МаксимальныйСостКодМеньшеL(L);
              L=L-L2;
              ЗаписатьБелыйКодДля(L2);
           }
           ЗаписатьБелыйКодДля(L);
                     //Это всегда код завершения
        }
        else {
        [Код аналогичный белой серии,
              с той разницей, что записываются
```

```
                черные коды]
    }
}
// Окончание строки изображения
}
```

Поскольку черные и белые серии чередуются, то реально код для белой и код для черной серии будут работать попеременно.

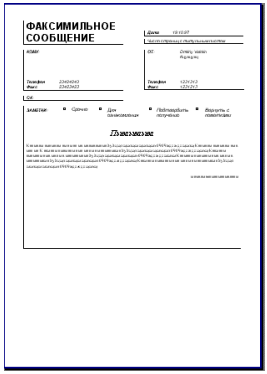
В терминах регулярных выражений мы получим для каждой строки нашего изображения (достаточно длинной, начинающейся с белой точки) выходной битовый поток вида:

((<Б-2560>)*[<Б-сст.>]<Б-зв.>(<Ч-2560>)*[<Ч-сст.>]<Ч-зв.>)$^{+}$
[(<Б-2560>)*[<Б-сст.>]<Б-зв.>]

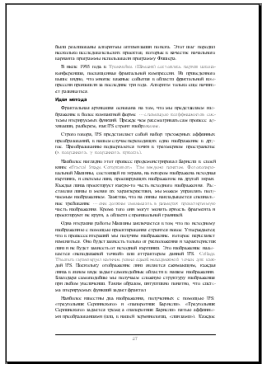
Где ()* — повтор 0 или более раз, ()$^{+}$.— повтор 1 или более раз, [] — включение 1 или 0 раз.

Для приведенного ранее примера: 0, 3, 556, 10... алгоритм сформирует следующий код: <Б-0><Ч-3><Б-512><Б-44><Ч-10>, или, согласно таблице, 00110101**10**0110010*10101101***0000100** (разные коды в потоке выделены для удобства). Этот код обладает свойством префиксных кодов и легко может быть свернут обратно в последовательность длин серий. Легко подсчитать, что для приведенной строки в 569 бит мы получили код длиной в 33 бита, т.е. степень сжатия составляет примерно 17 раз.

> **Упражнение:** Во сколько раз увеличится размер файла в худшем случае? Почему? (Приведенный в характеристиках алгоритма ответ не является полным, поскольку возможны большие значения худшей степени сжатия. Найдите их.)



Изображение, для которого очень выгодно применение алгоритма CCITT-3. (Большие области заполнены одним цветом).



Изображение, для которого менее выгодно применение алгоритма CCITT-3. (Меньше областей, заполненных одним цветом. Много коротких «черных» и «белых» серий).

Заметим, что единственное «сложное» выражение в нашем алгоритме: `L2=МаксимальныйДопКодМеньшеL(L)` — на практике работает очень просто: L2=(L>>6)*64, где >> — побитовый сдвиг L влево на 6 битов (можно сделать то же самое за одну побитовую операцию & — логическое И).

Упражнение: Дана строка изображения, записанная в виде длин серий — 442, 2, 56, 3, 23, 3, 104, 1, 94, 1, 231, размером 120 байт ((442+2+..+231)/8). Подсчитать степень сжатия этой строки алгоритмом CCITT Group 3.

Приведенные ниже таблицы построены с помощью классического алгоритма Хаффмана (отдельно для длин черных и белых серий). Значения вероятностей появления для конкретных длин серий были получены путем анализа большого количества факсимильных изображений.

Таблица кодов завершения:

Длина серии	Код белой подстроки	Код черной подстроки
0	00110101	0000110111
1	00111	010
2	0111	11
3	1000	10
4	1011	011
5	1100	0011
6	1110	0010
7	1111	00011
8	10011	000101
9	10100	000100
10	00111	0000100
11	01000	0000101
12	001000	0000111
13	000011	00000100
14	110100	00000111
15	110101	000011000
16	101010	0000010111
17	101011	0000011000
18	0100111	0000001000
19	0001100	00001100111
20	0001000	00001101000
21	0010111	00001101100
22	0000011	00000110111
23	0000100	00000101000
24	0101000	00000010111
25	0101011	00000011000
26	0010011	000011001010
27	0100100	000011001011
28	0011000	000011001100
29	00000010	000011001101
30	00000011	000001101000
31	00011010	000001101001

Длина серии	Код белой подстроки	Код черной подстроки
32	00011011	000001101010
33	00010010	000001101011
34	00010011	000011010010
35	00010100	000011010011
36	00010101	000011010100
37	00010110	000011010101
38	00010111	000011010110
39	00101000	000011010111
40	00101001	000001101100
41	00101010	000001101101
42	00101011	000011011010
43	00101100	000011011011
44	00101101	000001010100
45	00000100	000001010101
46	00000101	000001010110
47	00001010	000001010111
48	00001011	000001100100
49	01010010	000001100101
50	01010011	000001010010
51	01010100	000001010011
52	01010101	000000100100
53	00100100	000000110111
54	00100101	000000111000
55	01011000	000000100111
56	01011001	000000101000
57	01011010	000001011000
58	01011011	000001011001
59	01001010	000000101011
60	01001011	000000101100
61	00110010	000001011010
62	00110011	000001100110
63	00110100	000001100111

Таблица составных кодов:

Длина серии	Код белой подстроки	Код черной подстроки
64	11011	0000001111
128	10010	000011001000
192	01011	000011001001
256	0110111	000001011011
320	00110110	000000110011
384	00110111	000000110100
448	01100100	000000110101
512	01100101	0000001101100
576	01101000	0000001101101
640	01100111	0000001001010
704	011001100	0000001001011
768	011001101	0000001001100
832	011010010	0000001001101
896	011010011	0000001110010
960	011010100	0000001110011
1024	011010101	0000001110100
1088	011010110	0000001110101
1152	011010111	0000001110110
1216	011011000	0000001110111
1280	011011001	0000001010010

Длина серии	Код белой подстроки	Код черной подстроки
1344	011011010	0000001010011
1408	011011011	0000001010100
1472	010011000	0000001010101
1536	010011001	0000001011010
1600	010011010	0000001011011
1664	011000	0000001100100
1728	010011011	0000001100101
1792	00000001000	совп. с белой
1856	00000001100	— // —
1920	00000001101	— // —
1984	000000010010	— // —
2048	000000010011	— // —
2112	000000010100	— // —
2176	000000010101	— // —
2240	000000010110	— // —
2304	000000010111	— // —
2368	000000011100	— // —
2432	000000011101	— // —
2496	000000011110	— // —
2560	000000011111	— // —

Если в одном столбце встретятся два числа с одинаковым префиксом, то это опечатка.

Этот алгоритм реализован в формате TIFF.

Характеристики алгоритма CCITT Group 3

Степени сжатия: лучшая стремится в пределе к 213.(3), средняя 2, в худшем случае увеличивает файл в 5 раз.

Класс изображений: Двуцветные черно-белые изображения, в которых преобладают большие пространства, заполненные белым цветом.

Симметричность: Близка к 1.

Характерные особенности: Данный алгоритм чрезвычайно прост в реализации, быстр и может быть легко реализован аппаратно.

JBIG

Алгоритм разработан группой экспертов ISO (Joint Bi-level Experts Group) специально для сжатия однобитных черно-белых изображений [5]. Например, факсов или отсканированных документов. В принципе, может применяться и к 2-х, и к 4-х битовым картинкам. При этом алгоритм разбивает их на отдельные битовые плоскости. JBIG позволяет управлять такими параметрами, как порядок разбиения изображения на битовые плоскости, ширина полос в изображении, уровни масштабирования. Последняя возможность позволяет легко ориентироваться в базе больших по размерам изображений, просматривая сначала их уменьшенные копии. Настраивая эти параметры, можно использовать описанный выше эффект «огрубленного изображения» при получении изображения по сети или по любому другому каналу, пропускная способность которого мала по сравнению с возможностями процессора. Распаковываться изображение на экране будет постепенно, как бы медленно «проявляясь». При этом человек начинает анализировать картинку задолго до конца процесса разархивации.

Алгоритм построен на базе Q-кодировщика [6], патентом на который владеет IBM. Q-кодер, так же как и алгоритм Хаффмана, использует для чаще появляющихся символов короткие цепочки, а для реже появляющихся — длинные. Однако, в отличие от него, в алгоритме используются и последовательности символов.

Lossless JPEG

Этот алгоритм разработан группой экспертов в области фотографии (Joint Photographic Expert Group). В отличие от JBIG, Lossless JPEG ориентирован на полноцветные 24-битные или 8-битные в градациях серого изображения без палитры. Он представляет собой специальную реализацию JPEG без потерь. Степени сжатия: 20, 2, 1. Lossless JPEG рекомендуется применять в тех приложениях, где необходимо побитовое соответствие исходного и декомпрессированного изображений. Подробнее об алгоритме сжатия JPEG см. следующий раздел.

Заключение

Попробуем на этом этапе сделать некоторые обобщения. С одной стороны, приведенные выше алгоритмы достаточно универсальны и покрывают все типы изображений, с другой — у них, по сегодняшним меркам, слишком маленькая степень сжатия. Используя один из алгоритмов сжатия без потерь, можно обеспечить архивацию изображения примерно в два раза. В то же время алгоритмы сжатия с потерями оперируют с коэффициентами 10-200 раз. Помимо возможности модификации изображения, одна из основных причин подобной разницы заключается в том, что традиционные алгоритмы ориентированы на работу с цепочкой. Они не учитывают, так называемую, «когерентность областей» в изображениях. Идея когерентности областей заключается в малом изменении цвета и структуры на небольшом участке изображения. Все алгоритмы, о которых речь пойдет ниже, были созданы позднее специально для сжатия графики и используют эту идею.

Справедливости ради следует отметить, что и в классических алгоритмах можно использовать идею когерентности. Существуют алгоритмы обхода изображения по «фрактальной» кривой, при работе которых оно также вытягивается в цепочку; но за счет того, что кривая обегает области изображения по сложной

траектории, участки близких цветов в получающейся цепочке удлиняются.

Вопросы для самоконтроля

1) На какой класс изображений ориентирован алгоритм RLE?
2) Приведите два примера «плохих» изображений для первого варианта алгоритма RLE, для которых файл максимально увеличится в размере.
3) На какой класс изображений ориентирован алгоритм CCITT G-3?
4) Приведите пример «плохого» изображения для алгоритма CCITT G-3, для которого файл максимально увеличится в размере. (Приведенный в характеристиках алгоритма ответ не является полным, поскольку требует более «умной» реализации алгоритма.)
5) Приведите пример «плохого» изображения для алгоритма Хаффмана.
6) Сравните алгоритмы сжатия изображений без потерь.
7) В чем заключается идея когерентности областей?

Если вы обнаружите ошибки или неточности в данном тексте, просьба сообщить о них авторам по адресу:
compression@graphicon.ru

Ответы к вопросам и упражнениям и исходные тексты программ вы можете найти на
http://compression.graphicon.ru/

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 цветопредставления 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):

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

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» (Объясните, почему?). Свои неприятные стороны есть и у других критериев.

Рассмотрим, например, максимальное отклонение:

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

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).

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

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 изо-

бражений, 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.

Алгоритм JPEG

JPEG — один из самых новых и достаточно мощных алгоритмов. Практически он является стандартом де-факто для полноцветных изображений [1]. 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.

Шаг 1.

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:

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

Обратное преобразование осуществляется умножением вектора YUV на обратную матрицу.

$$\begin{vmatrix} R \\ G \\ B \end{vmatrix} = \begin{vmatrix} 1 & 0 & 1.402 \\ 1 & -0.34414 & -0.71414 \\ 1 & 1.772 & 0 \end{vmatrix} * \left(\begin{vmatrix} Y \\ Cb \\ Cr \end{vmatrix} - \begin{vmatrix} 0 \\ 128 \\ 128 \end{vmatrix} \right)$$

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:

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

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

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

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).

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

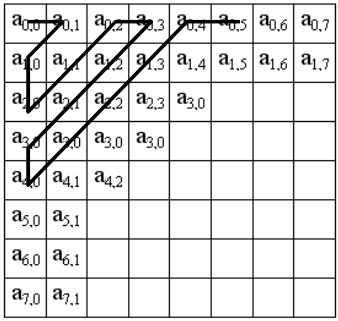
На этом шаге осуществляется управление степенью сжатия, и происходят самые большие потери. Понятно, что, задавая МК с большими коэффициентами, мы получим больше нулей и, следовательно, большую степень сжатия.

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 при помощи «зигзаг»-сканирования, т.е. берем элементы с индексами (0,0), (0,1), (1,0), (2,0)...



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

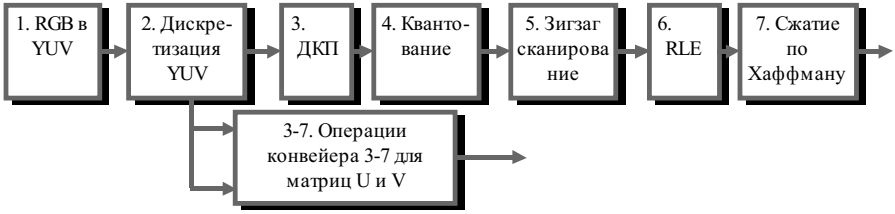
Шаг 6.

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:

1) При повышении степени сжатия изображение распадается на отдельные квадраты (8x8). Это связано с тем, что происходят большие потери в низких частотах при квантовании, и восстановить исходные данные становится невозможно.
2) 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 соот-

ветствующая 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 соб-

ственные. 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.

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». 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.

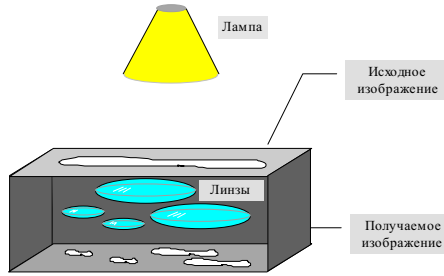
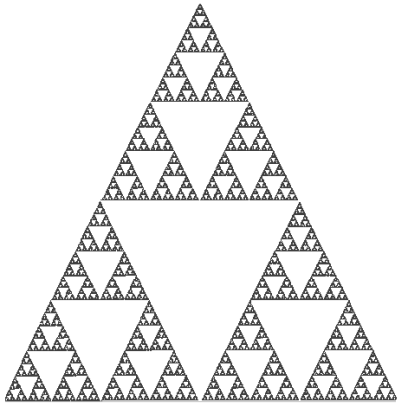


Рисунок. Машина Барнсли

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» и «папоротник Барнсли».



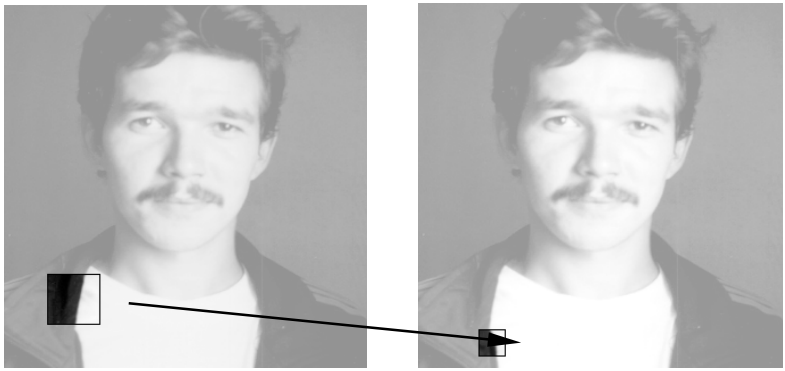
Треугольник Серпинского. Задается 3 преобразованиями.

«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.

> **Упражнение:** Укажите в изображении 4 области, объединение которых покрывало бы все изображение, и каждая из которых была бы подобна всему изображению (не забывайте о стебле папоротника).

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 $\boldsymbol{w}: \boldsymbol{R}^2 \to \boldsymbol{R}^2$, представимое в виде

$$\boldsymbol{w}(\bar{\boldsymbol{\nu}}) = \boldsymbol{w}\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{pmatrix} = \begin{pmatrix}\boldsymbol{a} & \boldsymbol{b}\\ \boldsymbol{c} & \boldsymbol{d}\end{pmatrix} \bullet \begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{pmatrix} + \begin{pmatrix}\boldsymbol{e}\\ \boldsymbol{f}\end{pmatrix}$$

где *a, b, c, d, e, f* действительные числа и $\begin{pmatrix}\boldsymbol{x} & \boldsymbol{y}\end{pmatrix} \in \boldsymbol{R}^2$ называется *двумерным аффинным преобразованием*.

Определение. Преобразование $\boldsymbol{w}: \boldsymbol{R}^3 \to \boldsymbol{R}^3$, представимое в виде

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

где a, b, c, d, e, f, p, q, r, s, t, u действительные числа и $\begin{pmatrix} x & y & z \end{pmatrix} \in R^3$ называется трехмерным аффинным преобразованием.

Определение. Пусть $f : \mathrm{X} \to \mathrm{X}$ — преобразование в пространстве X. Точка $x_f \in \mathrm{X}$ такая, что $f(x_f) = x_f$ называется *неподвижной точкой (аттрактором)* преобразования.

Определение. Преобразование $f : \mathrm{X} \to \mathrm{X}$ в метрическом пространстве (X, d) называется сжимающим, если существует число s: $0 \le s < 1$, такое, что

$$d(f(x), f(y)) \le s \cdot d(x, y) \qquad \forall x, y \in \mathrm{X}$$

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 $f : \mathrm{X} \to \mathrm{X}$ — сжимающее преобразование в полном метрическом пространстве (X, *d*). Тогда существует в точности одна неподвижная точка $x_f \in \mathrm{X}$ этого преобразования, и для любой точки $x \in \mathrm{X}$ последовательность $\{ f^n(x): \ n = 0,1,2... \}$ сходится к x_f.

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 $S(x,y) \in [0...1] \quad \forall x,y \in [0...1]$

Пусть трехмерное аффинное преобразование $w_i : R^3 \to R^3$, записано в виде

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

и определено на компактном подмножестве D_i 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

$$\begin{pmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

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

Определение. Конечная совокупность W сжимающих трехмерных аффинных преобразований w_i, определенных на областях D_i, таких, что $w_i(D_i) = R_i$ и $R_i \cap R_j = \phi \quad \forall i \neq j$, называется *системой итерируемых функций (IFS)*.

Системе итерируемых функций однозначно сопоставляется неподвижная точка — изображение. Таким образом, процесс компрессии заключается в поиске коэффициентов системы, а процесс декомпрессии — в проведении итераций системы до

стабилизации полученного изображения (неподвижной точки IFS). На практике бывает достаточно 7-16 итераций. Области R_i в дальнейшем будут именоваться *ранговыми*, а области D_i — *доменными*.

Построение алгоритма

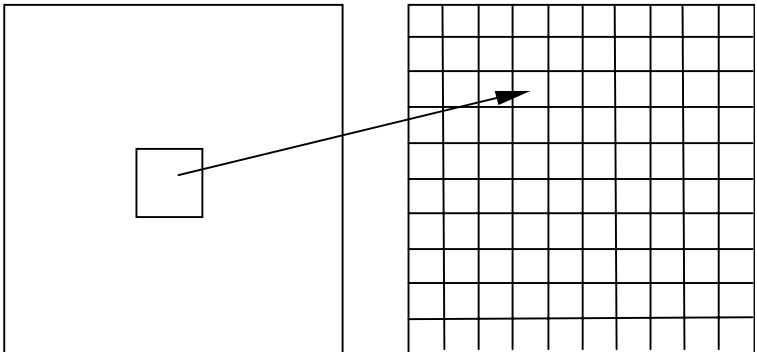
Как уже стало очевидным из изложенного выше, основной задачей при компрессии фрактальным алгоритмом является нахождение соответствующих аффинных преобразований. В самом общем случае мы можем переводить любые по размеру и форме области изображения, однако в этом случае получается астрономическое число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

В ***учебном 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 0^0, 90^0, 180^0 или 270^0*. Также допускает-

ся зеркальное отражение. Общее число возможных преобразований (считая пустое) — 8.

6) Масштабирование (сжатие) по вертикали (яркости) осуществляется *в фиксированное число раз* — в 0,75.



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 переводе.

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 от 2.
3) Алгоритм не сможет воспользоваться подобием объектов в изображении, угол между которыми не кратен 90^0.

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. Ниже приводится схема этого алгоритма.

```
for (all range blocks) {
   min_distance = MaximumDistance;
   Rij = image->CopyBlock(i,j);
   for (all domain blocks) { // С поворотами и отр.
      current=Координаты тек. преобразования;
      D=image->CopyBlock(current);
      current_distance = Rij.L2distance(D);
      if(current_distance < min_distance) {
         // Если коэффициенты best хуже:
```

```
            min_distance = current_distance;
            best = current;
        }
    } // Next range block
    Save_Coefficients_to_file(best);
} // Next domain block
```

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:

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

где r_{ij} — значения пикселов рангового блока (R), а d_{ij} — значения пикселов доменного блока (D). При этом мера считается как:

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

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:

Часть программы	Число операций
`for (all domain blocks)`	4096 (=512/8·512/8)

`for (all range blocks) + symmetry transformation`	492032 (=(512/2-8)* (512/2-8)*8)
Вычисление *q* и *d(R,D)*	> 3*64 операций «+» > 2*64 операций «·»
Итог:	> 3* 128.983.236.608 операций «+» > 2* 128.983.236.608 операций «·»

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=image->CopyBlock(D_coord_for_R);
      For(every pixel(i,j) in the block{
         Rij = 0.75Dij + oR;
      } //Next pixel
   } //Next block
}//Until end
```

Поскольку мы записывали коэффициенты для блоков R_{ij} (которые, как мы оговорили, в нашем частном случае являются квадратами одинакового размера) *последовательно*, то получается, что мы последовательно заполняем изображение по квад-

ратам 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 пере-

станет 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 потребовать, 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 приоритет этих условий, мы будем очень гибко управлять коэффициентом компрессии изображения в диапазоне от побитового соответствия 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, уве-

личивая его в 2-4 раза без появления «лестничного эффекта». При увеличении степени компрессии появляется «блочный» эффект на границах блоков в изображении.

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+1} всегда можно представить в виде $b^1_i=(a_{2i}+a_{2i+1})/2$ и $b^2_i=(a_{2i}-a_{2i+1})/2$. Аналогично последовательность a_i может быть попарно переведена в последовательность $b^{1,2}_i$.

Разберем конкретный пример: пусть мы сжимаем строку из 8 значений яркости пикселов (a_i): (220, 211, 212, 218, 217, 214, 210, 202). Мы получим следующие последовательности b^1_i, и b^2_i: (215.5, 215, 215.5, 206) и (4.5, -3, 1.5, 4). Заметим, что значения b^2_i достаточно близки к 0. Повторим операцию, рассматривая b^1_i как a_i. 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.

Упражнение: Мы восстановили из файла цепочку (215, 211) (0, 5) (5, -3, 2, 4) (см. пример). Постройте строку из восьми значений яркости пикселов, которую воссоздаст алгоритм волнового сжатия.

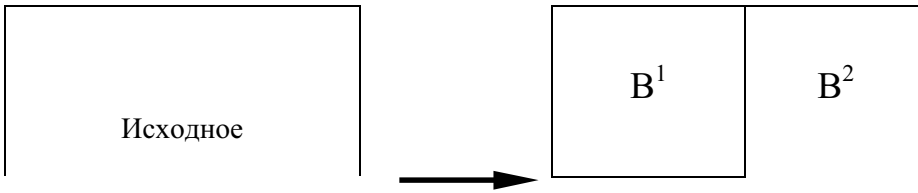
Алгоритм для двумерных данных реализуется аналогично. Если у нас есть квадрат из 4 точек с яркостями $a_{2i,2j}$, $a_{2i+1,\,2j}$, $a_{2i,\,2j+1}$, и $a_{2i+1,\,2j+1}$, то

$$b^1_{i,j} = (a_{2i,2j} + a_{2i+1,2j} + a_{2i,2j+1} + a_{2i+1,2j+1})/4$$

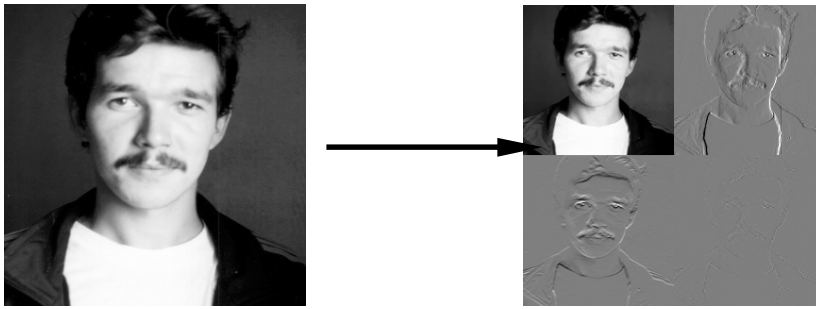
$$b^2_{i,j} = (a_{2i,2j} + a_{2i+1,2j} - a_{2i,2j+1} - a_{2i+1,2j+1})/4$$

$$b^3_{i,j} = (a_{2i,2j} - a_{2i+1,2j} + a_{2i,2j+1} - a_{2i+1,2j+1})/4$$

$$b^4_{i,j} = (a_{2i,2j} - a_{2i+1,2j} - a_{2i,2j+1} + a_{2i+1,2j+1})/4$$



Используя эти формулы, мы для изображения 512x512 пикселов получим после первого преобразования 4 матрицы размером 256x256 элементов:



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 ($b^4_{i,j}$), 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 «прояв–ления» 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.

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:

1. **Лучшее качество изображения при сильной степени сжатия.** Или, что то же самое, б**о**льшая степень сжатия при том же качестве для высоких степеней сжатия. Фактически это

означает заметное уменьшение размеров графики «Web-качества», используемой большинством сайтов.

2. **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.
3. **Основной алгоритм сжатия заменен на wavelet**. Помимо указанного повышения степени сжатия это позволило избавиться от 8-пиксельной блочности, возникающей при повышении степени сжатия. Кроме того, плавное проявление изображения теперь изначально заложено в стандарт (Progressive JPEG, активно применяемый в Интернет, появился много позднее JPEG).
4. **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.
5. **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.

6. **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.
7. **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.

4) 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.

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. Формулу преобразования можно записать как:

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

Значение степени ST для как каждой компоненты R, G и B свое (определяется при сжатии компрессором). При восстановлении изображения выполняется обратное преобразование:

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

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. Его матрица выглядит так:

$$\begin{vmatrix} Y \\ U \\ V \end{vmatrix} = \begin{pmatrix} \left\lfloor \frac{R + 2G + B}{4} \right\rfloor \\ R - G \\ B - G \end{pmatrix}$$

Обратное преобразование осуществляется с помощью обратной матрицы:

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

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, приведенными ниже. Для сжатия с потерями коэффициенты выглядят как:

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

±3	0.01686411844287495	-0.09127176311424948
±4	0.02674875741080976	0
Другие i	0	0

Для сжатия без потерь коэффициенты задаются как:

	При упаковке		При распаковке	
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

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:

$$y_{output}(2n) = \sum_{j=0}^{N-1} x_{input}(j) \cdot h_H(j-2n)$$

$$y_{output}(2n+1) = \sum_{j=0}^{N-1} x_{input}(j) \cdot h_L(j-2n-1)$$

Поскольку большинство $h_L(i)$, кроме окрестности i=0, равны 0, то можно переписать приведенные формулы с меньшим количеством операций. Для простоты рассмотрим случай сжатия без потерь.

$$y_{out}(2n) = \frac{-x_{in}(2n-1) + 2 \cdot x_{in}(2n) + 6 \cdot x_{in}(2n+1) + 2 \cdot x_{in}(2n+2) - x_{in}(2n+3)}{8}$$

$$y_{out}(2n+1) = -\frac{x_{in}(2n)}{2} + x_{in}(2n+1) - \frac{x_{in}(2n+2)}{2}$$

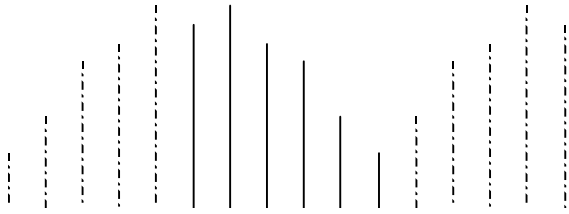
Легко показать, что данную запись можно эквивалентно переписать, уменьшив еще втрое количество операций умножения и деления (однако теперь необходимо будет подсчитать сначала все нечетные *y*). Добавим также операции округления до ближайшего целого, не превышающего заданное число *a*, обозначаемые как $\lfloor a \rfloor$:

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

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

Упражнение: Самостоятельно уменьшите количество операций для случая без потерь.

Рассмотрим на примере, как работает данное преобразование. Для того, чтобы преобразование можно было применять к крайним пикселам изображения, оно симметрично достраивается в обе стороны на несколько пикселов, как показано на рисунке ниже. В худшем случае (сжатие с потерями) нам необходимо достроить изображение на 4 пиксела.



Симметричное расширение изображения (яркости АБ…Е) по строке вправо и влево

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

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:

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

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

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 относитель-

но n=0 и 9. Воспользовавшись этим свойством, расширим нашу строку вправо и влево и применим обратное преобразование:

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	

Как видим, мы получили исходную цепочку (x_{in}= x_{out}).

Упражнение: Примените прямое и обратное DWT-преобразования к цепочке из 10 байт:

121, 107, 98, 102, 145, 182, 169, 174, 157, 155.

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 (высокочастотная составляющая).

y_{out}	1	0	3	1	11	4	13	-2	8	-5
y'_{out}	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 б**о**льшую 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.

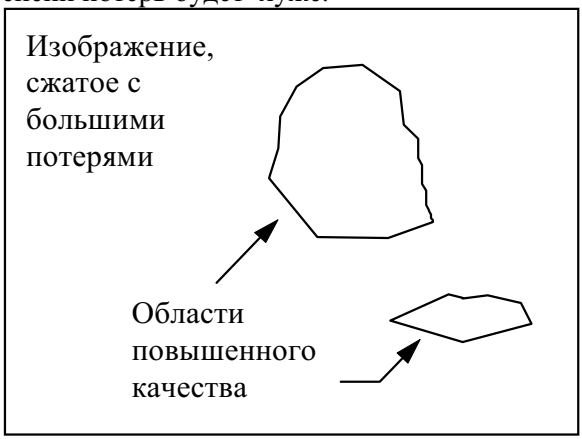
Для сжатия получающихся массивов данных в JPEG 2000 используется вариант арифметического сжатия, называемый MQ-

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, 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.



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:

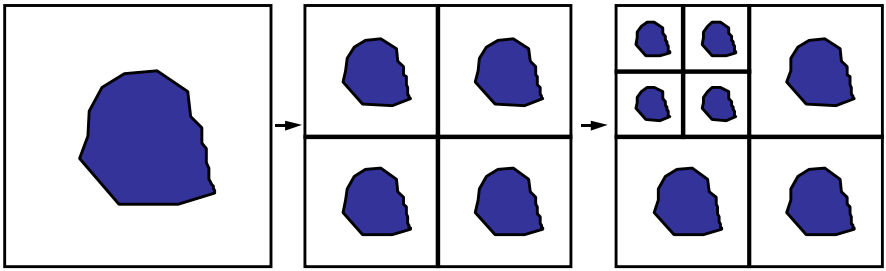
1) Для приложения должна быть критична (максимальна) степень сжатия, причем настолько, что возможен индивидуальный подход к каждому изображению.
2) Изображение сжимается один раз, а разжимается множество раз.

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 умень-

шать время загрузки изображений и степень загруженности каналов связи адаптивно.

В JPEG-2000 используется однобитное изображение-маска, 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:



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.

Заключение

В заключение рассмотрим таблицы, в которых сводятся воедино параметры различных алгоритмов сжатия изображений, рассмотренных нами выше.

Алгоритм	Особенности изображения, за счет которых происходит сжатие
RLE	Подряд идущие одинаковые цвета: 2 2 2 2 2 2 15 15 15
LZW	Одинаковые подцепочки: 2 3 15 40 2 3 15 40
Хаффмана	Разная частота появления цвета: 2 2 3 2 2 4 3 2 2 2 4
CCITT-3	Преобладание белого цвета в изображении, большие области, заполненные одним цветом
Рекурсивный	Плавные переходы цветов и отсутствие резких границ
JPEG	Отсутствие резких границ
Фрактальный	Подобие между элементами изображения

Алгоритм	К-ты сжатия	Симметричность по времени	На что ориентирован	Потери	Размерность
RLE	32, 2, 0.5	1	3,4-х битные	Нет	1D
LZW	1000, 4, 5/7	1.2-3	1-8 битные	Нет	1D
Хаффмана	8, 1.5, 1	1-1.5	8 битные	Нет	1D

CCITT-3	213(3), 5, 0.25	~1	1-битные	Нет	1D
JBIG	2-30 раз	~1	1-битные	Нет	2D
Lossless JPEG	2 раза	~1	24-бит. сер.	Нет	2D
Рекурсивное сжатие	2-200 раз	1.5	24-битные, серые	Да	2D
JPEG	2-200 раз	~1	24-битные, сер.	Да	2D
Фрактальный	2-2000 раз	1000-10000	24-бит. сер.	Да	2.5D

В приведенной таблице отчетливо видны тенденции 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.

Вопросы для самоконтроля

1) В чем разница между алгоритмами с потерей информации и без потери информации?
2) Приведите примеры мер потери информации и опишите их недостатки.
3) За счет чего сжимает изображения алгоритм JPEG?
4) В чем заключается идея фрактального алгоритма компрессии?
5) В чем заключается идея рекурсивного (волнового) сжатия?
6) Можно ли применять прием перевода в другое цветовое пространство алгоритма JPEG в других алгоритмах компрессии?

7) Сравните приведенные в этой главе алгоритмы сжатия изображений.

Если вы обнаружите ошибки или неточности в данном тексте, просьба сообщить о них авторам по адресу:
compression@graphicon.ru

Ответы к вопросам и упражнениям и исходные тексты программ вы можете найти на
http://compression.graphicon.ru/

Различия между форматом и алгоритмом

Дополнительная глава

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. (См. примеры в приложении.)

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 1:** 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.

Литература

Литература по алгоритмам сжатия

[1] *Wallace G.K.* «The JPEG still picture compression standard» // Communication of ACM. Volume 34. Number 4 April 1991.
[2] *Smith B., Rowe L.* «Algorithm for manipulating compressed images.» // Computer Graphics and applications. September 1993.
[3] *Jacquin A.*«Fractal image coding based on a theory of iterated contractive image transformations» // Visual Comm. and Image Processing, vol. SPIE-1360, 1990.
[4] *Fisher Y.* «Fractal image compression» // SigGraph-92.
[5] «Progressive Bi-level Image Compression, Revision 4.1» // ISO/IEC JTC1/SC2/WG9, CD 11544, September 16, 1991.
[6] *Pennebaker W.B., Mitchell J.L., Langdon G.G., Arps R.B.*, «An overview of the basic principles of the Q-coder adaptive binary arithmetic coder» // IBM Journal of research and development, Vol.32, No.6, November 1988, pp. 771-726.
[7] *Huffman D.A.* «A method for the construction of minimum redundancy codes.» // Proc. of IRE vol.40, 1952, pp. 1098-1101.
[8] Standardisation of Group 3 Facsimile apparatus for document transmission. CCITT Recommendations. Fascicle VII.2. T.4. 1980.
[9] *Александров В.В., Горский Н.Д.* «Представление и обработка изображений: рекурсивный подход» // Л-д.: Наука 1985, 190 стр.
[10] *Климов А.С.* «Форматы графических файлов». // С.-Петербург, Изд. «ДиаСофт» 1995.
[11] *Ватолин Д.С.* «MPEG - стандарт ISO на видео в системах мультимедиа» // Открытые системы. Номер 2. Лето 1995
[12] *Ватолин Д.С.* «Тенденции развития алгоритмов архивации графики» // Открытые системы. Номер 4. Зима 1995

[13] *Ватолин Д.С.* «Алгоритмы сжатия изображений» // ISBN 5-89407-041-4 М.: Диалог-МГУ, 1999.
[14] *Добеши И.* «Десять лекций по вейвлетам» // Пер. с анг. Е.В. Мищенко, под ред. А.П.Петухова. М.: Ижевск 2001, 464 стр.
[15] *Яншин В.В.* «Анализ и обработка изображений (принципы и алгоритмы)» // М.: Машиностроение 1995
[16] *Павлидис Т.* «Алгоритмы машинной графики и обработка изображений» // М.: Радио и связь 1986, 400 стр.
[17] *Претт У.* «Цифровая обработка изображений» в двух томах // М.: Мир 1982, 790 стр.
[18] *Розеншельд А.* «Распознавание и обработка изображений» // М.: Мир 1972, 232 стр.
[19] Материалы конференции «Графикон» (статьи по сжатию публиковались практически ежегодно) доступны в научных библиотеках и, частично, на http://www.graphicon.ru
[20] *Ярославский Л.П.* «Введение в цифровую обработку изображений» // М.:Сов. радио 1969, 312 стр.
[21] *Яблонский С.В.* «Введение в дискретную математику». // М. «Наука», 1986. Раздел «Теория кодирования».
[22] Более 150 статей по сжатию изображений можно найти на http://graphics.cs.msu.su/library/.

Литература по форматам изображений

[23] *Климов А.С.* «Форматы графических файлов» // НИПФ «ДиаСофт Лтд», 1995.
[24] *Романов В.Ю.* «Популярные форматы файлов для хранения графических изображений на IBM PC» // Москва «Унитех», 1992
[25] *Сван Том* «Форматы файлов Windows» // М. «Бином», 1995
[26] *Hamilton E.*«JPEG File Interchange Format» // Version 1.2. September 1, 1992, San Jose CA: C-Cube Microsystems, Inc.
[27] Aldus Corporation Developer's Desk. «TIFF - Revision 6.0, Final», June 3, 1992

Указатель терминов