Цель сжатия данных - обеспечить компактное представление данных, вырабатываемых источником, для их более экономного сохранения и передачи по каналам связи.

## СЖАТИЕ БЕЗ ПОТЕРЬ

Основано на устранении избыточности —

- *Оформление*(прописные и строчные буквы)
- *Неэффективное кодирование* (наиболее часто встречающиеся символы – меньшим числом бит)
  приписывать к сжатому файлу некую таблицу (словарь)
  *Алгоритмы, основанные на перекодировании информации − алгоритмы Хафмана.*
- *Наличие повторяющихся фрагментов*
  алгоритмы, основанные на выявлении повторов
  *- методы RLE (Run Length Encoding) – часто – в графических иллюстрациях (на фото – нет из-за шумов)*

## С ПОТЕРЯМИ

Неприменимы для программ, н вполне походит для фото/видео. Потеря информации при сжатии и последующей распаковке - как появление дополнительного «шума». Но поскольку при создании определенный «шум» уже присутствует, его небольшое увеличение не выглядит критичным, а выигрыш в размерах дает огромный (в 10-15 раз на музыке, в 20-30 раз на фото- и видеоматериалах).

JPEG – изображения (.jpg)

MPEG – видео и музыка (.MPG и .MP3)

ТОЛЬКО ДЛЯ ПОТРЕБИТЕЛЬСКИХ ЗАДАЧ. Это значит, например, что если фотография передается для просмотра, а музыка для воспроизведения, то подобные алгоритмы применять можно. Если же они передаются для дальнейшей обработки, например для редактирования, то никакая потеря информации в исходном материале недопустима.

На фотографических иллюстрациях, предназначенных для воспроизведения на экране, потеря 5% информации обычно некритична, а в некоторых случаях можно допустить и 20-25%.

**RLE (без потерь)**

Групповое кодирование - Run Length Encoding (RLE) - один из самых старых и самых простых алгоритмов архивации. Сжатие в RLE происходит за счет замены цепочек одинаковых байт на пары "счетчик, значение". («красный, красный, ..., красный» записывается как «N красных»).

К положительным сторонам алгоритма можно отнести то, что он не требует дополнительной памяти при работе, и быстро выполняется. Алгоритм применяется в форматах PCX, TIFF, BMP. Интересная особенность группового кодирования в PCX заключается в том, что степень архивации для некоторых изображений может быть существенно повышена всего лишь за счет изменения порядка цветов в палитре изображения.

# 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*. Рассматриваемый нами ниже вариант алгоритма будет использовать дерево для представления и хранения цепочек. 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.

**Deflate** — это алгоритм сжатия без потерь, использующий комбинацию алгоритмов LZ77 и Хаффмана. Deflate считается свободным от всех существующих патентов, и пока оставался в силе патент на LZW (он применяется в формате GIF), это привело к использованию Deflate не только в формате ZIP, для которого Кац изначально его спроектировал, но также в компрессоре/декомпрессоре gzip и в PNG-изображениях.

Deflate-поток содержит серии блоков. Перед каждым блоком находится трёхбитовый заголовок:

- Один бит: флаг последнего блока.
    - 1: блок последний.
    - 0: блок не последний.
- Два бита: метод, с помощью которого были закодированы данные.
    - 00: данные не закодированы (в блоке находятся непосредственно выходные данные).
    - 01: данные закодированы по методу статического Хаффмана.
    - 10: данные закодированы по методу динамического Хаффмана.
    - 11: зарезервированное значение (ошибка).

    Большая часть блоков кодируется с помощью метода 10 (динамический Хаффман), который предоставляет оптимизированное дерево кодов Хаффмана для каждого нового блока. Инструкции для создания дерева кодов Хаффмана следуют непосредственно за заголовком блока.

    Компрессия выполняется в два этапа:

    - замена повторяющихся строк указателями (алгоритм LZ77);
    - замена символов новыми символами, основываясь на частоте их использования (алгоритм Хаффмана).

**JPEG (с потерями)**

Алгоритм JPEG был разработан группой фирм под названием Joint Photographic Experts Group. Целью проекта являлось создание высокоэффективного стандарта сжатия как черно-белых, так и цветных изображений, эта цель и была достигнута разработчиками. В настоящее время JPEG находит широчайшее применение там, где требуется высокая степень сжатия - например, в Internet.

В отличие от LZW-алгоритма JPEG-кодирование является кодированием с потерями. Сам алгоритм кодирования базируется на очень сложной математике, но в общих чертах его можно описать так: изображение разбивается на квадраты 8*8 пикселов, а затем каждый квадрат преобразуется в последовательную цепочку из 64 пикселов. Далее каждая такая цепочка подвергается так называемому DCT-преобразованию, являющемуся одной из разновидностей

дискретного преобразования Фурье. Оно заключается в том, что входную последовательность пикселов можно представить в виде суммы синусоидальных и косинусоидальных составляющих с кратными частотами (так называемых гармоник). В этом случае нам необходимо знать лишь амплитуды этих составляющих для того, чтобы восстановить входную последовательность с достаточной степенью точности. Чем большее количество гармонических составляющих нам известно, тем меньше будет расхождение между оригиналом и сжатым изображением. Большинство JPEG-кодеров позволяют регулировать степень сжатия. Достигается это очень простым путем: чем выше степень сжатия установлена, тем меньшим количеством гармоник будет представлен каждый 64-пиксельный блок.

Безусловно, сильной стороной данного вида кодирования является большой коэффициент сжатия при сохранении исходной цветовой глубины. Именно это свойство обусловило его широкое применение в Internet, где уменьшение размера файлов имеет первостепенное значение, в мультимедийных энциклопедиях, где требуется хранение возможно большего количества графики в ограниченном объеме.

Отрицательным свойством этого формата является неустранимое никакими средствами, внутренне ему присущее ухудшение качества изображения. Именно этот печальный факт не позволяет применять его в полиграфии, где качество ставится во главу угла.

Однако формат JPEG не является пределом совершенства в стремлении уменьшить размер конечного файла. В последнее время ведутся интенсивные исследования в области так называемого вейвлет-преобразования (или всплеск-преобразования). Основанные на сложнейших математических принципах вейвлет-кодеры позволяют получить большее сжатие, чем JPEG, при меньших потерях информации. Несмотря на сложность математики вейвлет-преобразования, в программной реализации оно проще, чем JPEG. Хотя алгоритмы вейвлет-сжатия пока находятся в начальной стадии развития, им уготовано большое будущее.

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.

2. АОН

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.

3.

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 - высокочастотной.

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).

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".

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)...` (рис. 6.1)

**Рис. 6.1.**

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.

5. Свертываем получившиеся пары кодированием по Хаффману с фиксированной таблицей.

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*

# Область применения[править | править код]

Алгоритм JPEG наиболее эффективен для сжатия фотографий и картин, содержащих реалистичные сцены с плавными переходами яркости и цвета. Наибольшее распространение JPEG получил в цифровой фотографии и для хранения и передачи изображений с использованием Интернета.

Формат JPEG в режиме сжатия с потерями малопригоден для сжатия чертежей, текстовой и знаковой графики, где резкий контраст между соседними пикселами приводит к появлению заметных артефактов. Такие изображения целесообразно сохранять в форматах без потерь, таких как JPEG-LS, TIFF, GIF, PNG, либо использовать режим сжатия Lossless JPEG.

JPEG (как и другие форматы сжатия с потерями) не подходит для сжатия изображений при многоэтапной обработке, так как искажения в изображения будут вноситься каждый раз при сохранении промежуточных результатов обработки.

JPEG не должен использоваться и в тех случаях, когда недопустимы даже минимальные потери, например при сжатии астрономических или медицинских изображений. В таких случаях может быть рекомендован предусмотренный стандартом JPEG режим сжатия Lossless JPEG (который, однако, не поддерживается большинством популярных кодеков) или стандарт сжатия JPEG-LS.

## ВИДЕО

В результате подавляющее большинство сегодняшних алгоритмов сжатия видео являются алгоритмами с потерей данных. При сжатии используется несколько типов избыточности:

- **Когерентность областей изображения** - малое изменение цвета изображения в соседних пикселах (свойство, которое эксплуатируют все алгоритмы сжатия изображений с потерями).
- **Избыточность в цветовых плоскостях** - используется большая важность яркости изображения для восприятия.
- **Подобие между кадрами** - использование того факта, что на скорости 25 *кадров* в секунду, как правило, соседние *кадры* изменяются незначительно.

Первые два пункта знакомы вам по алгоритмам сжатия графики. Использование подобия между кадрами в самом простом и наиболее часто используемом случае означает *кодирование* не самого нового кадра, а его *разности* с предыдущим кадром. Для видео типа "говорящая голова" (передача новостей, видеотелефоны), большая часть кадра остается неизменной, и даже такой простой метод позволяет значительно уменьшить *поток данных*. Более сложный метод заключается в нахождении для каждого блока в сжимаемом *кадре* наименее отличающегося от него блока в *кадре*, используемом в качестве базового. Далее кодируется разница между этими блоками. Этот метод существенно более ресурсоемкий.

## MPEG

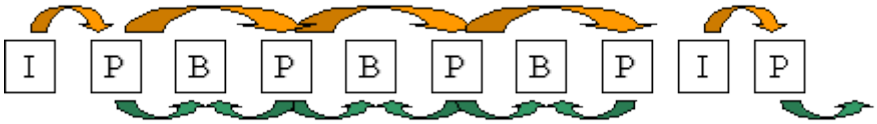
Технология сжатия видео в *MPEG* распадается на две части: уменьшение избыточности видеоинформации во временном измерении, основанное на том, что соседние *кадры*, как правило, отличаются не сильно, и сжатие отдельных изображений.

Для того чтобы удовлетворить *противоречивым требованиям* и увеличить гибкость алгоритма, рассматривается четыре типа *кадров*:

- `I-кадры` - *кадры* сжатые независимо от других *кадров* ( `I-Intra pictures` ),
- `P-кадры` - сжатые с использованием ссылки на одно изображение ( `P-Predicted` ),
- `B-кадры` - сжатые с использованием ссылки на два изображения ( `B-Bidirection` ),

- DC-*кадры* - независимо сжатые с большой потерей качества (используются только при быстром поиске).

I-*кадры* обеспечивают возможность произвольного доступа к любому кадру, являясь своеобразными входными точками в *поток данных* для декодера. P-*кадры* используют при *архивации* ссылку на один I- или P-*кадр*, повышая тем самым степень сжатия фильма в целом. B-*кадры*, используя ссылки на два кадра, находящихся впереди и позади, обеспечивают наивысшую степень сжатия. Сами в качестве ссылки использоваться не могут. Последовательность *кадров* в фильме может быть, например, такой: IBBPBBPBBPBBIBBPBB... Или, если мы не экономим на степени сжатия, такой (рис. 8.1):




Частота I-*кадров* выбирается в зависимости от требований на время произвольного доступа и надежности потока при передаче через канал с ошибками. Соотношение P- и B-*кадров* подбирается, исходя из требований к величине компрессии и ограничений декодера. Как правило, *декодирование* B-*кадров* требует больше вычислительных *мощностей*, однако позволяет повысить степень сжатия. Именно варьирование частоты *кадров* разных типов обеспечивает алгоритму необходимую гибкость и возможность расширения. Понятно, что для того, чтобы распаковать B-*кадр*, мы должны уже распаковать те *кадры*, на которые он ссылается.

Алгоритм сжатия отдельных *кадров* в *MPEG* похож на соответствующий алгоритм для статических изображений - *JPEG*. Если говорить коротко, то сам алгоритм сжатия представляет собой конвейер преобразований. Это дискретное косинусное преобразование исходной матрицы 8x8, *квантование* матрицы и вытягивание ее в вектор v11,v12,v21,v31,v22,...,v88 (зигзаг-сканирование), сжатие вектора групповым кодированием и, наконец, сжатие по алгоритму Хаффмана.

## Motion-JPEG

Motion-*JPEG* (или M-*JPEG* ) является наиболее простым алгоритмом сжатия видео. В нем каждый *кадр* сжимается независимо алгоритмом *JPEG*. Этот прием дает высокую скорость доступа к произвольным *кадрам*, как в прямом, так и в обратном порядке следования. Соответственно легко реализуются плавные "перемотки" в обоих направлениях, *аудио*-визуальная синхронизация и, что самое главное - редактирование. Типичные *операции JPEG* сейчас поддерживаются на аппаратном уровне большинством видеокарт и данный формат позволяет легко оперировать большими объемами данных при монтаже фильмов. Независимое сжатие отдельных *кадров* позволяет накладывать различные эффекты, не опасаясь, что взаимное влияние соседних *кадров* внесет дополнительные искажения в фильм.

### *Использование векторов смещений блоков*

Простейший способ учитывать подобие соседних *кадров* - это вычитать каждый блок сжимаемого кадра из соответствующего блока предыдущего. Однако более гибким является алгоритм поиска *векторов*, на которые сдвинулись блоки текущего кадра по отношению к предыдущему.

Для каждого блока в изображении мы находим блок близкий по некоторой *метрике* (например, по сумме *квадратов разности* пикселей) в предыдущем *кадре* в некоторой *окрестности* текущего положения блока. Если минимальное расстояние по выбранной *метрике* с блоками в предыдущем *кадре* больше выбранного порога - блок сжимается независимо (рис. 8.2).

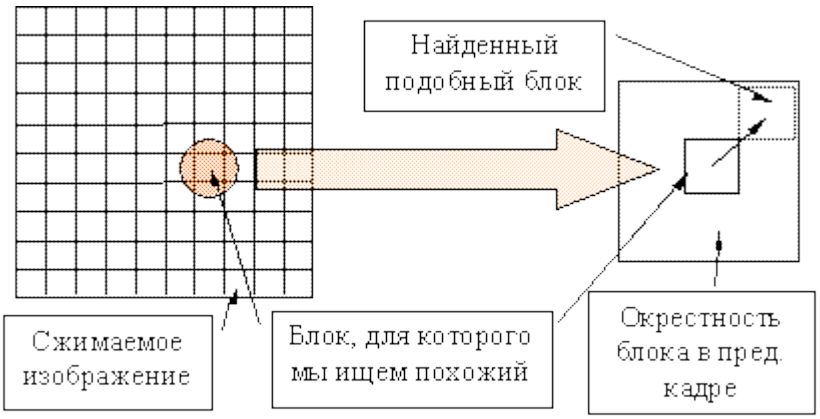



**Рис. 8.2.**

Таким образом, вместе с каждым блоком в поток теперь сохраняются координаты смещения максимально похожего блока в предыдущем `I-` или `P-кадре`, либо признак того, что данные сжаты независимо. Эти координаты задают вектор смещения блока ( `motion vector` ). В ситуациях, когда камера наезжает на объект или дает *панораму*, использование *векторов* смещений блоков позволяет значительно уменьшить амплитуду *разности кадров*, и как следствие - значительно поднять степень сжатия.

**GIF** (англ. *Graphics Interchange Format* «формат для обмена изображениями») — растровый формат графических изображений. Способен хранить сжатые данные без потери качества в формате не более 256 цветов. Не зависящий от аппаратного обеспечения формат, поддержка прозрачности и анимации. GIF использует LZW-компрессию, что позволяет сжимать файлы, в которых много однородных заливок (логотипы, надписи, схемы).

Изображение в формате GIF хранится построчно, поддерживается только формат с индексированной палитрой цветов. Стандарт разрабатывался только для поддержки 256-цветовой палитры.

Один из цветов в палитре может быть объявлен «прозрачным». В этом случае в программах, которые поддерживают прозрачность GIF (например, большинство современных браузеров) сквозь пиксели, окрашенные «прозрачным» цветом, будет виден фон. «Полупрозрачность» пикселей (технология альфа-канала) не поддерживается.

Формат GIF поддерживает анимационные изображения. Они представляют собой последовательность из нескольких статичных кадров, а также информацию о том, сколько времени каждый кадр должен быть показан на экране. Анимацию можно сделать цикличной (англ. *loop*), тогда вслед за последним кадром начнётся воспроизведение первого кадра и т. д.

GIF-анимация может использовать прозрачность для того, чтобы не сохранять очередной кадр целиком, а только изменения относительно предыдущего.

GIF использует формат сжатия LZW. Таким образом хорошо сжимаются изображения, строки которых имеют повторяющиеся участки. В особенности изображения, в которых много пикселей одного цвета по горизонтали[5].

Алгоритм сжатия LZW относится к форматам сжатия без потерь. Это означает, что восстановленные из GIF данные будут в точности соответствовать упакованным. Следует отметить, что это верно только для 8-битных изображений с палитрой, для цветной фотографии потери будут обусловлены переводом её к 256 цветам.

Формат GIF допускает чересстрочное хранение данных. При этом строки разбиваются на группы, и меняется порядок хранения строк в файле. При загрузке изображение проявляется

постепенно, в несколько проходов. Благодаря этому, имея только часть файла, можно увидеть изображение целиком, но с меньшим разрешением.

**BMP** (от англ. *Bitmap Picture*) — формат хранения растровых изображений, разработанный компанией Microsoft. Файлы формата BMP могут иметь расширения `.bmp`, `.dib` и `.rle`.