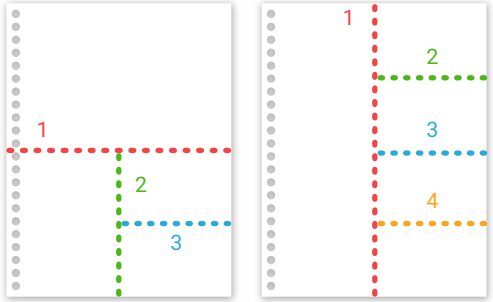
Модуль 3. Урок 1.
Сжатие графической информации.

Начнем с определения общих понятий: есть **алгоритмы** и **методы**, есть **форматы**, есть **расширения**. В видео есть ещё **кодеки**, **контейнеры**, **стандарты**. А ещё мы говорим о **сжатии** и о **кодировании**. Когда мы говорим о **компрессии** – мы все равно говорим о сжатии, но просто используем английский термин вместо русского.

Алгоритм сжатия — это набор инструкций, который в конечное число шагов приводит к преобразованию исходного кода цифрового изображения в код меньшего объёма за счёт устранения избыточности.

Алгоритм сжатия — это совсем не синоним **формату** графического файла. Алгоритм JPEG может применяться в каком-нибудь другом формате, например, вы можете сохранить файл TIFF, выбрав сжатие JPEG для его содержимого.



Например, лист бумаги можно сложить в восемь раз как минимум двумя разными способами. Мы получаем одинаковый результат, используя разные последовательности действий, т.е. разные **алгоритмы.**

Метод — это более общее понятие, чем алгоритм. Например, метод сжатия RLE может быть реализован различными алгоритмами, которые мы, конечно же, назовём RLE. Только в разных случаях они реализуют одну и ту же идею, содержащуюся в методе, немного по-разному, чтобы добиться наилучших результатов в тех случаях, для которых он разрабатывается.

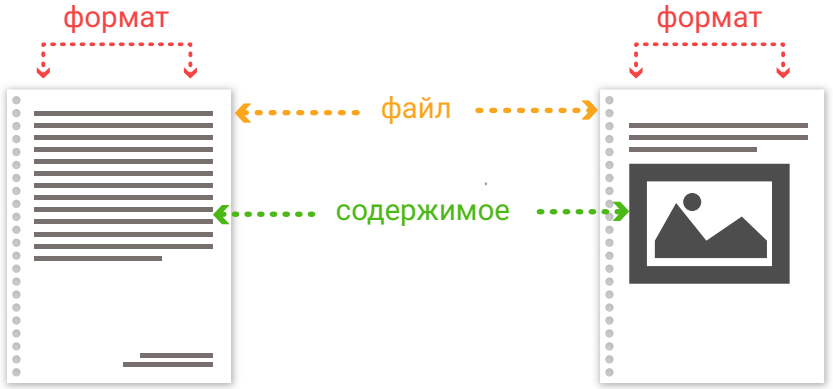
Уменьшить размер листа бумаги можно, просто скомкав его. Это пример применения другого **метода**, в результате которого мы получаем другой результат.

Код – это конечная последовательность битов, соответственно, **длина кода** – это количество битов в нём.

Кодировать можно с разной целью. Начнём с того, что после преобразования изображения из аналоговой формы в цифровую, сама по себе запись подразумевает какое-то кодирование – ведь нужно же как-то обозначить, в какой точке какой был цвет. Можно кодировать информацию с целью шифрования или, скажем, для резервирования, тогда она, как правило, прибавляет в объёме. Зато частичная потеря такой информации компенсируется за счет восстановления из резерва. Так работают серверные хранилища, использующие RAID-массивы.

Форматом файла называют структуру данных, записанных в этом файле. Собственно файл – это просто ограниченная область памяти с определенным именем и атрибутами.

А вот как внутри этой области расположены данные – этот порядок и называют форматом. Собственно файл – это просто ограниченная область памяти с определенным именем и атрибутами. А вот как внутри этой области расположены данные – этот порядок и называют форматом.



Одни и те же данные можно записать по-разному. А ещё их можно их как-нибудь упаковать. В частности, применяя те самые алгоритмы сжатия.

Вот здесь важно не путать: формат и алгоритм -- это разные понятия. Например, информация в файле PDF может записываться с использованием сразу нескольких алгоритмов сжатия для разных видов содержащихся в нем данных.

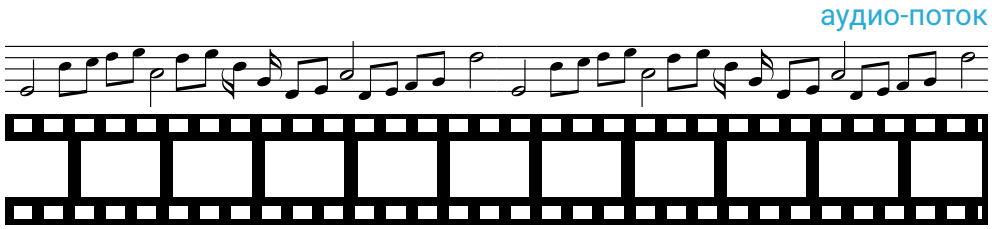
Для того, чтобы файлы разных форматов не путались между собой, в некоторых системах, например, в Windows, используют расширение – это дополнительные символы к имени файла, записываются через точку. Это простой способ объяснить всем программам, какого формата данный файл.

Если вы возьмете обычную фотографию JPG и переименуете расширение в, скажем, EXE, то Фотошоп откажется её открывать, даже не разобравшись, что там внутри. Если же сделаете наоборот, какой-нибудь неподобающий файл переименуете в JPG, то Фотошоп попытается его открыть, но удивится и выплюнет ошибку.

Если говорить о видео, то там путаница усугубляется. Например, есть **кодеки**, **контейнеры** и **стандарты**, а заодно и **потоки**. **Алгоритмы** и **форматы** при этом никуда не деваются, как и **расширения**.

По сути, кодек – это просто программа, реализующая алгоритм кодирования. Основная задача кодека — сформировать нужные потоки аудио и видео информации.

Причём, потоки будут использоваться как в файловом, так и в потоковом видео. Это уже способ хранения и передачи. От этого способа зависит стратегия сжатия.



Теперь, когда у нас есть потоки, их нужно как-то уместить все вместе в **контейнер**, чтобы записать в файл для воспроизведения или передачи по сети. Собственно, **контейнер** и определяет структуру файла, возможные кодеки и их параметры, состав потоков и дополнительных данных, например, субтитры.И **кодеки** и **контейнеры** в разных случаях могут называть форматом видео. В случае с кодеками – это формат сжатия, в случае с контейнерами – формат упаковки в файл или поток.

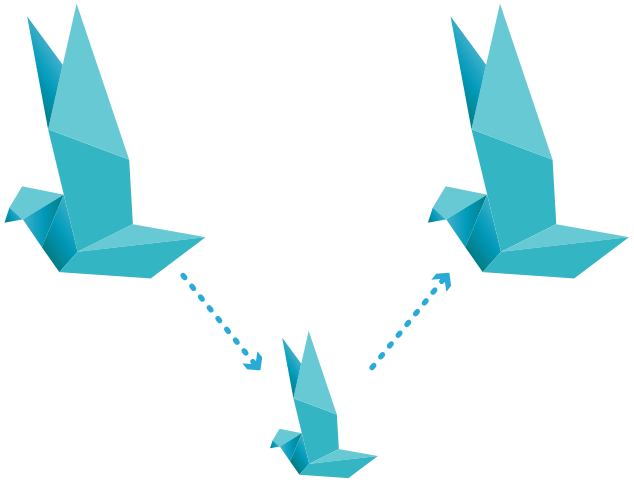
По умолчанию будем считать форматом видео комбинацию **контейнера**, а так же **кодеков**, которыми сжаты те **потоки**, которые содержатся в этом **контейнере**, и ключевые параметры сжатия – хотя бы разрешение и битрейт.

Основные алгоритмы сжатия.

Для доставки конечного изображения зрителю применяется сжатие. Начнём с классификации: бывает сжатие **с потерями** и **без потерь**.

Сжатие без потерь подразумевает, что результирующее изображение будет в точности соответствовать оригинальному.

То есть, не выглядеть, как оригинальное, а абсолютно точно передавать значение каждого пикселя, при этом предполагается, что размер файла или величина потока уменьшится.

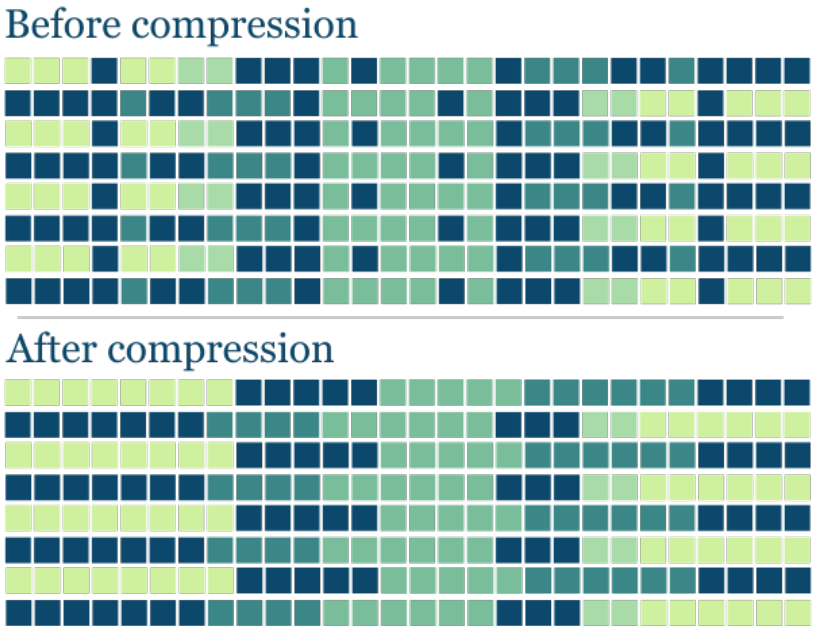


При всех достоинствах, эти методы неэффективны, у них очень низкий коэффициент сжатия. Строго говоря, они сжимают в разы.

Сжатие с потерями более эффективно, оно сжимает не в разы, а в десятки, сотни и даже тысячи раз, однако от этого страдает точность передачи изображения и скорость кодирования-декодирования.

Основная суть сжатия без потерь – запись повторяющейся информации в более компактном виде . Так работает алгоритм RLE, он рассматривает изображение по строкам и компактно записывает одинаковые элементы.

Это хорошо работает, если в изображении есть большие области равномерной заливки. В полноцветном изображении однородная область такой только кажется. Таким образом, RLE ориентирован скорее на однобитные изображения.

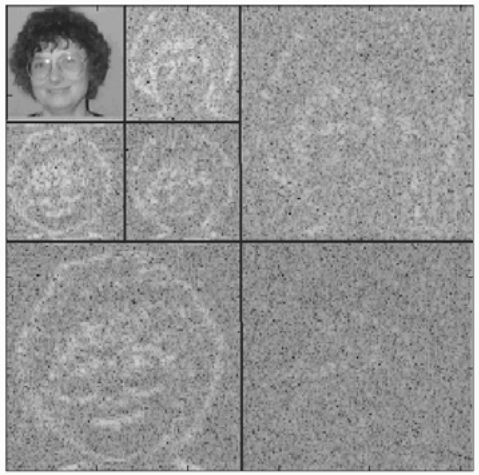


Другой алгоритм, LZW, работает уже с последовательностями байтов, а не со значением пикселей, и может применяться не только в графике.

Там он ориентирован на работу с индексированным цветом, а для работы с полноцветными изображениями хорошо подходит алгоритм Losless Jpeg.

Вейвлет и фрактальное сжатие

Вейвлет преобразование называют ещё волновым или рекурсивным. Идея его заключается в том, что в файл сохраняется разница – число между средними значениями соседних блоков изображения.



Для всего изображения создаётся четыре матрицы половинного размера по вертикали и горизонтали. Первая хранит уменьшенное изображение, вторая и третья – значения разности пар пикселей по вертикали и горизонтали, а четвёртая – усреднённую разницу значений пикселей по диагонали.

Далее операция повторяется для первой матрицы несколько раз, в итоге мы имеем миниатюру исходного изображения и множество матриц, описывающих разницу. Алгоритм вейвлет используется в качестве основного алгоритма для JPEG 2000

Плюсы этого сжатия:

- Нет дробления на блоки (эффекта Гиббса)
- Постепенное декодирование (удобно передавать по сети)
- Наличие миниатюры изображения
- Регулируемая степень сжатия (от двух до двухсот раз)

Идея фрактального сжатия состоит в поиске самоподобных элементов изображения, поэтому он прекрасно работает с изображениями, состоящими из подобий себя (например, с фрактальными изображениями).

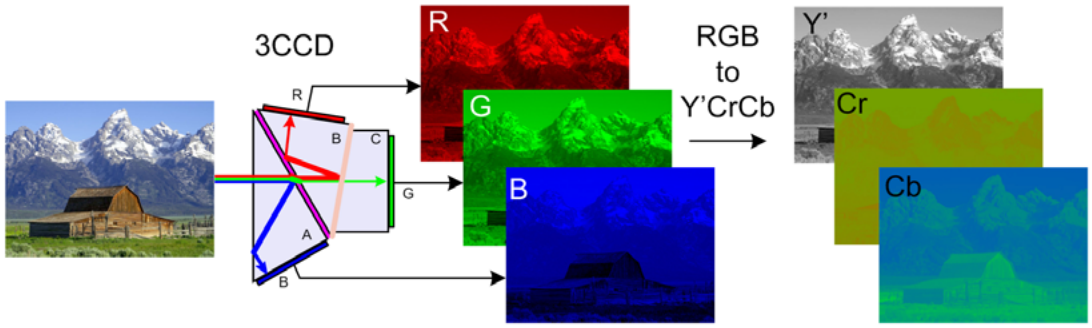
Коэффициент сжатия получается очень высоким – до 20 000. Но и на обычных изображениях этот алгоритм сжимает лучше других алгоритмов. Однако время поиска себеподобных областей очень велико, поэтому алгоритм работает очень медленно.

Урок 2-3. Форматы графики. JPEG

Любое сжатие информации возможно только при наличии избыточности. Для рассмотренных выше алгоритмов избыточностью считаются однородные области.
Рассмотрим подробнее то, как кодируется JPEG. Кодирование состоит из следующих этапов:

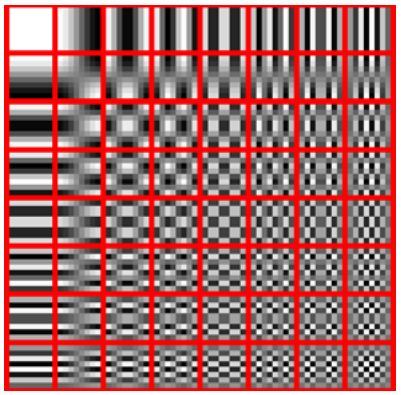
1. Перевод RGB в YCbCr

На первом шаге яркость записывается отдельно от цветности. Яркость для нашего глаза имеет большее значение, чем цветность, а это значит, что отделив яркость от цветности мы можем манипулировать с цветовыми каналами. При этом, если мы посмотрим на цветовые каналы в таком представлении, то увидим, что они значительно менее контрастные, а значит, обнаружить однородные области будет проще.



2. Субдискретизация

На этом этапе мы разбиваем исходное изображение на матрицы 8x8 пикселей.



Здесь происходит потеря, которая называется субдискретизацией. Если для канала яркости считываются все пиксели, то в цветовых каналах пиксели выбираются через одну строку и через один ряд.

Таким образом, мы теряем ¾ полезной информации о цветовых составляющих. Но для всего файла получили сжатие в два раза.

3. Дискретное косинусное преобразование.

Итак, мы имеем матрицу 8 на 8 значений, они будут варьироваться от 0 до 255 и любой пиксель может принять любое значение из этого диапазона.

	1	2	3	4	5	6	7	8
1	1603	203	11	45	-30	-14	-14	-7
2	108	-93	10	49	27	6	8	2
3	-42	-20	-6	16	17	9	3	2
4	56	69	7	-25	-10	-5	-2	-2
5	-33	-21	17	8	3	-4	-5	-3
6	-16	-14	8	2	-4	-2	1	1
7	0	-5	-6	-1	2	3	0	1
8	9	5	-6	-9	0	3	3	1

- НЧ компоненты;
- СЧ компоненты;
- ВЧ компоненты

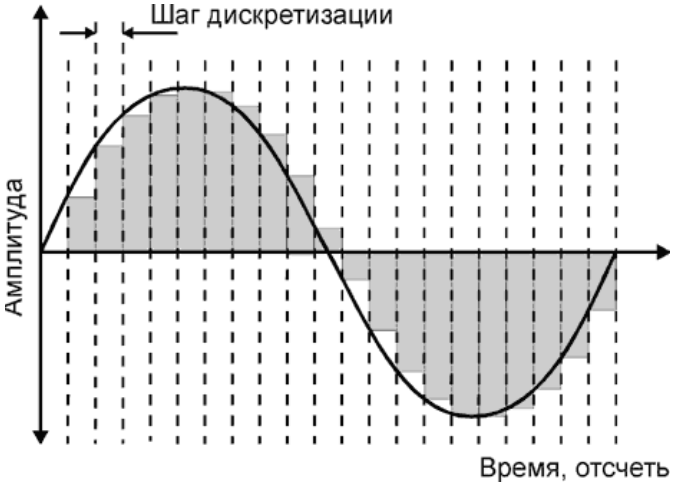
Сжимать информацию можно только в том случае, если есть избыточность, но искать повторы и закономерности среди большого числа значений может быть затруднительно.

При ДКП мы получаем матрицу, в которой коэффициенты в левом верхнем углу соответствуют низкочастотной составляющей изображения, а в нижнем правом углу – высокочастотной.

Понятие частоты здесь следует из рассмотрения изображения как двумерного сигнала. Плавные изменения значений соответствуют низкой частоте, а резкие скачки – высокой. Основная часть изображения содержится в низкочастотной области, а различные шумы – в высокочастотной. Таким образом, мы отделяем значимую часть изображения от мелочей.

4. Квантование

Если в предыдущем шаге мы преобразовывали дискретный сигнал в непрерывный, то на этом этапе мы поступаем наоборот. Аналогичным методом оцифровывают звук, поэтому мы рассмотрим квантование на примере звукового сигнала.

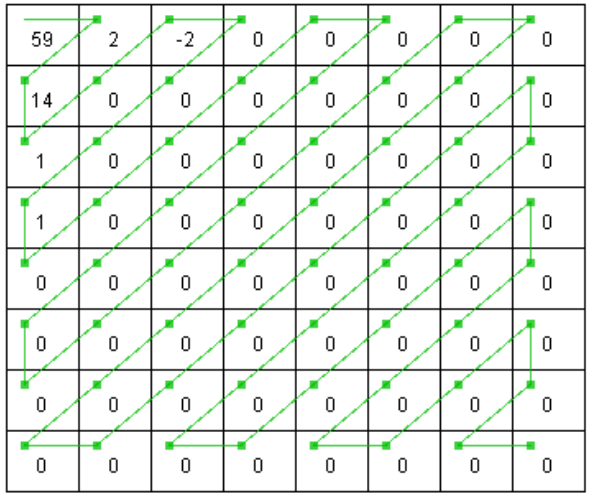


На сетке шаг по горизонтали называется дискретизацией – для звука это отсчёт времени, а для изображения – координаты пикселей. Шаг по вертикали это квантование (для звука – уровень сигнала, и для изображения тоже, в нашем примере уровень сигнала выражен яркостью).

Здесь квантование – это деление матрицы коэффициентов из предыдущего шага на матрицу квантования, именно на этом этапе можно управлять степенью сжатия и определять, что именно будет сжато.

5. Линейный вид

На этом шаге происходит считывание матрицы для представления значений в строчку. Причём, считывание проводится зигзагом – от левого верхнего угла к нижнему правому, чтобы получить значения, выстроенные по мере роста частоты коэффициентов. Обычно ненулевые значения в этой матрице сконцентрированы в левом верхнем углу.



При считывании таким зигзагом мы получаем строку из 64 значений, из которых в начале некоторое количество ненулевых, а дальше – нули.

6. RLE

Применяется алгоритм сжатия без потерь, то есть, всю информацию мы записываем в более компактном виде, указывая, сколько бит для значения нам нужно пропустить, сколько предоставить, и само значение.

7. Алгоритм Хаффмана

Весь полученный набор мы сжимаем алгоритмом Хаффмана. Это тоже метод сжатия без потерь, суть которого заключается в том, что часто используемым значениям присваиваются короткие коды, а редко встречающимся – длинные.

При этом сами коды устроены так, что их можно записать подряд без разделителей, а декодирующая программа самостоятельно сможет отделить один от другого.

Ключевой момент – алфавит (всё допустимое количество кодируемых значений) должен быть ограниченным, и чем меньше он будет, тем короче будут присвоенные коды, и, соответственно, итоговая запись будет компактнее.

Чтобы восстановить закодированное изображение, нужно пройти весь путь в обратную сторону, однако следует помнить, что абсролютной идентичности мы не получим, так как на этапе квантования не только делили значения в матрицах, но и округляли до целых, безвозвратно теряя подробности.

Тем не менее, алгоритм позволяет сжать изображение в десятки раз, может работать с большой глубиной цвета и позволяет регулировать уровень сжатия.

Альтернативы JPEG: BPG

Формат BPG был создан Фабрисом Белларом. Технологически этот формат, который так и позиционируется, как замена JPEG, родом из видеокодека HEVC, известного так же как H.265. Рассмотрим его возможности:

- Высокий уровень сжатия. Итоговые файлы получаются заметно меньшего размера, по сравнению с файлами JPEG аналогичного качества.

- Поддержка в большинстве браузеров, благодаря наличию декодировщика на языке JavaScript. Размер сжатого кода JavaScript-библиотеки декодирования 76 Кб.
- Методы кодирования основаны на подмножестве стандарта сжатия видео HEVC/H.265;
- Использование идентичных с JPEG форматов цветовой субдискретизации: оттенки серого, YCbCr 4:2:0, 4:2:2, 4:4:4, что позволяет исключить потери при преобразовании из JPEG.
- Поддержка слоя прозрачности;

- Поддержка схем формирования цвета CMYK, RGB, YCgCo. Заметьте, тут присутствует и такая экзотическая схема: цветовые каналы отсчитываются не от синего и красного, а от зелёного и оранжевого.
- Поддержка от 8 до 14 битов на цветовой канал;
- Наличие режима сжатия без потерь.
- Возможность интеграции в файл различных метаданных, включая блоки EXIF. В общем-то, она есть и в JPEG.
- Поддерживается анимация, но подразумевается, что ролики будут короткие, чтобы их можно было уместить в кеше. Если декодер не поддерживает воспроизведение анимации, то будет показано статическое изображение. Зато уровень сжатия здесь высокий за счёт использования межкадрового кодирования, которое применяется в кодеке HEVC.

WebP

Помимо недавно появившегося формата BPG есть формат, продвигаемый Google – WebP. И он так же основан на видеокодеке VP8, и права и компанию, которая разрабатывала этот кодек, Google купил.

Параллельно с WebP для статики разработан ещё и WebM для видео. Собственно, WebP кодируется так же, как опорные кадры из WebM.

Описание Google говорит нам, что версия для сжатия с потерями использует следующие улучшения, если сравнивать с JPEG:

Первое – это предсказание блоков. Здесь суть в том, что кодирование осуществляется так же по блокам, но блоки эти не независимы, а кодируются, исходя из уже известной информации о соседних блоках. Поскольку часто бывает, что соседний блок очень похож, или один из них повторяет в целом картину данного блока, то это используется для повышения уровня сжатия. Очевидный, в общем-то ход. Раз больше возможностей сказать кодировщику «смотри вон туда и сделай так же», то это приводит к тому, что файл заполняется большим, чем в JPEG, количеством нулевых значений, отсюда – лучшие показатели сжатия.

JPEG file size: 120.78 KB

WebP file size: 80.76 KB

Второе улучшение — это адаптивное распределение уровней квантования, то есть, «делителя качества», если совсем на пальцах. Обычно в изображении есть фрагменты попроще и фрагменты посложнее. В случае с JPEG уровень квантования устанавливается для всех блоков одинаковым. Но кодировщик WebP оценивает изображение и распределяет коэффициенты квантования индивидуально каждому блоку.

Так же, вместо сжатия полученных после всех преобразований данных методом Хаффмана, используется более гибкое арифметическое энтропийное кодирование.

Выделим значимые характеристики формата:

Сжатие – Google заявляет, что их формат на 39% более эффективен.
Контейнер — RIFF.
Сжатие – как с потерями, так и без потерь.
Прозрачность – поддерживается, может быть сохранена без потерь даже если изображение сжималось с потерями.
Анимация — поддерживается.
Метаданные — не фиксированы.

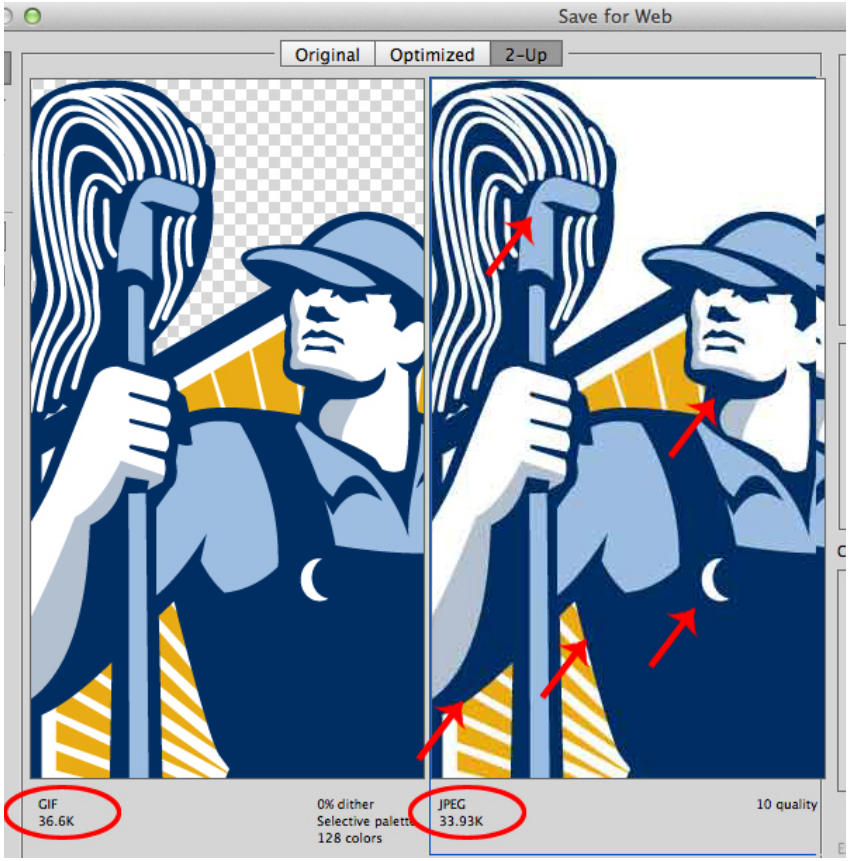
На чём забывают заострить внимание создатели WebP, так это на цветовой субдискретизации. WebP даже при сжатии без потерь будет сохранять цвет по схеме 4:2:0, то есть, лишь четверть цветовой информации от изначально имевшейся.
Так же, доступна только глубина цвета 8 бит.
Проще говоря, это формат для конечного просмотра, никак не для промежуточного сохранения, затем он и создавался.

GIF

Этот формат использует сжатие по алгоритму LZW.
Что в этом алгоритме хорошо:

- Сжатие без потерь
- Кодирование и декодирование производятся довольно быстро.

Недостатки:
Работает по строкам, что существенно ограничивает его способности находить избыточность. Такой же принцип был в RLE, но он ищет не последовательности одинаковых значений цвета, а одинаковые последовательности, «фразы». Такой подход более применим к полутоновым изображениям, хотя количество цветов, то есть, алфавит входящей последовательности, всё равно должен быть ограниченным. Из вышесказанного вытекает главный недостаток: индексированная палитра цвета, это 256 цветов в пределе, можно делать и меньше.



После модификации в 1989 году формат обрёл два важных свойства, за которые GIF любят: прозрачность и анимацию. Прозрачность весьма урезанная – только один цвет из 256 может быть объявлен прозрачным. Полноценного альфа-канала не предусмотрено.

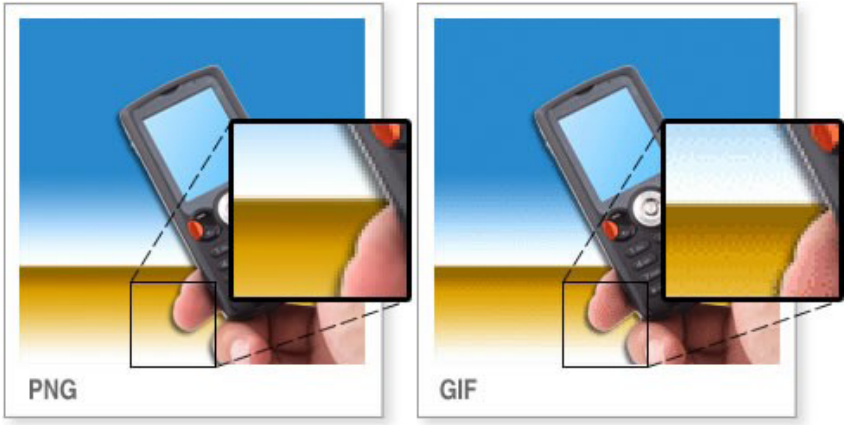
К свойствам формата относится и анимация. Возможность анимации тесно связана с прозрачностью. Анимированное изображение записывается разницей: только изменения относительно предыдущего. При этом, каждый кадр может иметь разную продолжительность.

PNG

PNG – это очень любопытный формат. Мало того, что он лучше, чем GIF сжимает изображения, он поддерживает полноцветные изображения. Изначально он использовался в программе Macromedia Fireworks в качестве оригинального, то есть внутреннего формата. Программа эта была для подготовки графики для веб-страниц. И прямо в этот файл можно было записать разметку на слайсы, ссылки и прочие подробности.

Рассмотрим возможности PNG:

- Открытый и бесплатный.
- Глубина цвета – 16 бит. То есть, можно хранить не только изображения для просмотра, но и исходники фотографий.
- При этом, существует три режима: greyscale, indexed 8 bit (это как в GIF, так называемый PNG-8) и полноцветный PNG-24.



- Цветовые пространства. Здесь всплывает происхождение формата. Про CMYK авторы забыли. В полиграфии, увы, неприменим.
- Анимация поддерживается в расширении формата aPNG, но плохо поддерживается браузерами.

- Прозрачность поддерживается, для этого выделяется отдельный канал, называется альфа-канал.
- Небольшие файлы с индексированными цветами компактнее укладываются в GIF, во всех других случаях файлы PNG будут меньше.
- Формат хранит метаданные, чего так не хватало GIF-у. Например, попробуйте напечатать GIF – он же не знает ничего даже о себе, не то что о запечатленном на нём изображении. И принтер будет вам честно выводить изображение в 72 точки на дюйм, потому что это экранное разрешение по умолчанию.

Теперь об алгоритме: как происходит сжатие. Алгоритмы, основанные на устранении повторяющейся информации бессильны, если в строках повторяющейся информации нет.

Выйти из этого положения можно двумя путями:
1. Посмотреть вверх и сказать кодировщику «повтори».
2. И так же можно сказать: да это же градиент! Просто прибавляй по единичке в каждой следующей ячейке.

В обоих случаях слева указывается фильтр – то самое указание кодировщику, что делать со строкой. У каждой строки фильтр может быть свой и это большое поле для творчества при оптимизации кодирования.

Всего фильтров, которые можно применить к строке, пять:

None – Без фильтра
Sub – Вычесть значение слева
Up – Вычесть значение сверху
Average – Вычесть среднее значение пикселей сверху и слева
Paeth – Подставить значение левого, верхнего или левого верхнего пикселя.

Lossless версии lossy форматов

Завершая тему сжатия статической графики, заметим, что многие вроде бы деструктивные форматы, то есть, сжимающие изображения с потерями, имеют версию для сжатия без потерь.

Например:

JPEG, JPEG2000, BPG, WebP – все они поддерживают сжатие без потерь, используя недеструктивные алгоритмы вместо обычных для них деструктивных.

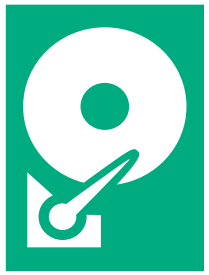
Есть целых два варианта недеструктивного JPEG сжатия: Lossless JPEG и JPEG-LS. Вот с JPEG-LS связано ещё одно любопытное понятие: мы уже упоминали сжатие с потерями, без потерь, а тут есть ещё и «почти без потерь», near lossless. В этом случае потери ограничены и регулируются пользователем. Здесь используется алгоритм LOCO-I, который расшифровывается как Low Complexity Lossless Compression for Images.

Lossless JPEG – это совсем другой алгоритм, фактически, дополнение к классическому JPEG. Использует схему предсказания значения пикселя по трём ближайшим соседям — верхнему, левому и верхнему левому пикселям, а для сжатия разницы между истинным и предсказанным значением пикселя использует энтропийное кодирование. В народ этот формат, как и прочие варианты на тему JPEG, не пошел.

Урок 4. Сжатие видео.

Видео в цифровом формате это последовательность изображений по аналогии с кинематографом. Графическая информация требует большого количества памяти, и хранить её без сжатия может быть затруднительно.

Несжатое видео



- Пропускная способность всего тракта
- Скорость системы хранения данных
- Объём памяти и системы хранения

Сжатое видео

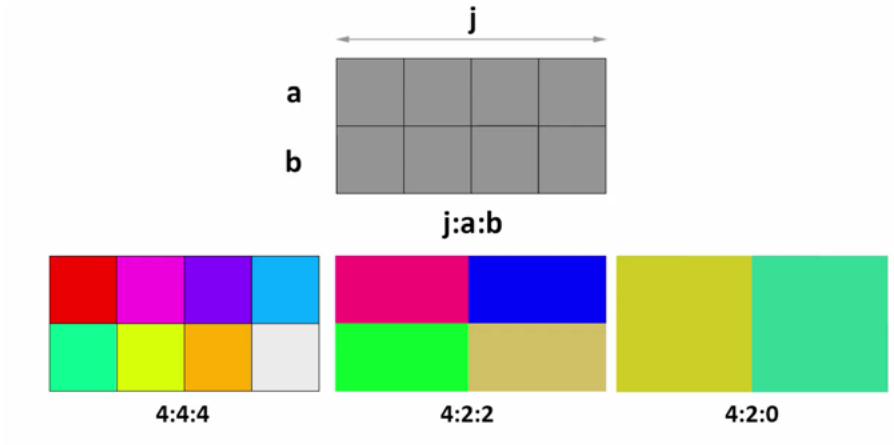
- Процессорная мощность
- Поддержка кодеков

Изучим, как кодируется видео. Чтобы нам было, от чего отталкиваться, посчитаем объём одного кадра несжатого видео. По сути, это объём одного несжатого изображения.
В стандарном разрешении это будет 1215 Кб, в высоком разрешении – 6075 Кб. Тогда при скорости 25 кадров в секунду, одна минута такого видео будет занимать 1,74 Гб для SD, и 8,69 Гб для HD, что чудовищно много.

И видеокамера, и монитор, используют для представления цвета модель RGB, но хранится видеозапись, как правило, в YCbCr. Мы уже знаем, что человеческий глаз более чувствителен к яркости, поэтому при сжатии страдать будет цвет.

Экономят на цвете простым путём – понижаю разрешение цветовых каналов. То есть, информация о цвете будет одинаковой сразу для нескольких пикселей.

Здесь нам следует подробнее объяснить, что означают цифры 4:4:4, 4:2:2, 4:2:0 при описании цвета.



Пусть j – количество пикселей, используемых в шаблоне, a – количество цветовых составляющих, используемых в первом ряду, а b, соответственно, во втором.

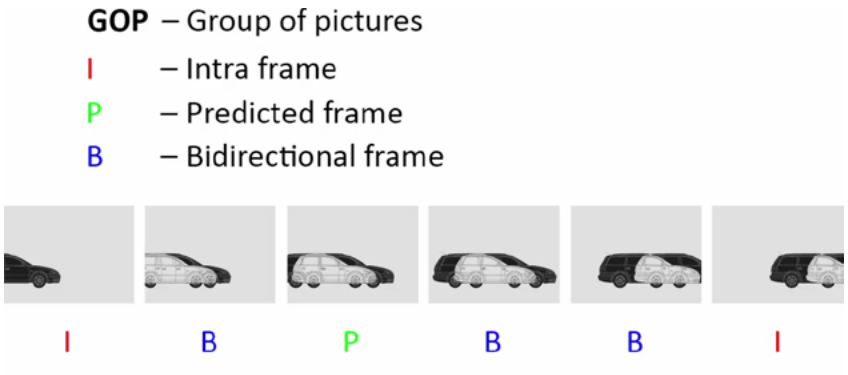
Тогда запись 4:4:4 означает, что для матрицы в 4 пикселя по ширине и два по высоте, количество данных о яркости совпадает с количеством данных о цветности. Так кодируется графика в Lossless форматах.

Запись 4:2:2 означает, что в каждой строке из 4 пикселей есть только два значения цвета. Так кодируют профессиональные видеокамеры.

В схеме 4:2:0 вторая строка повторяет первую, а в первой если лишь два значения цвета. Такая схема используется в JPEG.

Субдискретизация по схеме 4:2:0 может уменьшить изображение вдвое, но и этого недостаточно. Применив к каждому кадру видео сжатие JPEG, мы получим значительно большее сжатие. Оно называется внутрикадровым и используется в старых кодеках, а конкретно в DV.

1 час видеозаписи, состоящей из последовательных кадров в DV весит 13 Гб, то есть, примерно по 152 Кб на кадр, что тоже трудно назвать хорошим сжатием. Для дальнейшего сжатия используется предположение, что соседние кадры имеют много общего, то есть, изменения в них будут минимальны, если, конечно не меняется точка съёмки и т.д. Этим можно воспользоваться и записать полный кадр один раз, а далее только дописывать изменения. Этот полный кадр называется опорным, и с него начинается группа изображений (Group Of Pictures – GOP).



Между опорными кадрами присутствует предсказанные кадры (predicted), а между предсказанными – двунаправленные (bidirectional). При кодировании и декодировании видео в памяти кодера должна помещаться вся сцена в пределах одной группы, разумеется, предсказанные и двунаправленные кадры должны занимать значительно меньше места.

Кодеки.

Рассмотрим некоторые основные кодеки, которые используются для обработки видео.

Первый кодек H.261 был разработан в 1988 году и имел многие функции современных кодеков: опорные кадры, предсказанные кадры, а также сжатие кадров по макроблокам и с дискретным косинусным преобразованием.

Позже в 1993 году появился MPEG 1, отличительными чертами которого стали двунаправленные кадры, а также высокое разрешение.

В 1996 году появился MPEG 2, и на его основе было построенно телевещание, также он использовался в DVD.

В 1998 году появился в MPEG 4, сам по себе он не был популярен, но на его основе в 2003 году появился стандарт MPEG 4 Part 10, он же H.264, он требовал высокой мощности для кодирования и декодирования, но давал прекрасные результаты и быстро стал популярен.

Естественно, что это далеко не все используемые кодеки, а лишь их малая часть, существуют также LossLess кодеки, сжимающие без потерь (HuffYUV) и кодеки, разработанные для конкретных платформ (WindowsMedia, QuickTime).