профессиональное сжатие видео

крис касперски ака мыщъх, no-email

эта публикация открывает цикл статей, посвященный вопросам обработки видеоинформации в домашних условиях, и в первую очередь — осмысленной настройке MPEG2/MPEG4 кодеков, увеличивающей степень сжатия (без видимой потери качества) в несколько раз по сравнению с "сертифицированными профилями", установленными по умолчанию.

введение

Научно-технический прогресс развращает, приучая все делать одним взмахом мыши, в результате чего мы имеем DVD-диск от SUPERBIT и EXTRABIT "непревзойденного качества", подготовленные с помощью Ahead Nero (о чем честно свидетельствует поле Source Media Implementation Identifier, хранящееся на DVD-диске вместе с прочей служебной информацией) и записанные с грубыми нарушениями стандарта, в результате чего мы имеем проблемы — от появления "артефактов" и ухудшения качества, до полной невозможности воспроизведения диска на определенных моделях DVD-плееров (особенно стационарных).

Сжатие цифрового видео — очень сложная тема, намного более кромешная и запутанная, чем это кажется до попытки основательно разобраться в ней. Начав исправлять ошибки лицензионных DVD (исключительно в целях домашнего просмотра!), автор зарылся в стандарты, обложился спецификациями, скачал исходные коды всех доступных кодеков и… на несколько месяцев погрузился в пучину бесчисленных экспериментов, в результате чего стал сжимать намного качественнее и плотнее, чем окружающие. "Качественнее" — это значит лучше, чем на _оригинальном_ DVD-носителе (см. рис. 18, 19). Не верите? Говорите: "чудес не бывает"? А разве алгоритмы сжатия звука и видео — это уже само по себе не чудо? Немного людей знает на каких принципах они базируются и еще меньше готовы написать свой собственный кодек или усовершенствовать имеющийся.

Да что там! Подавляющее большинство использует настройки по умолчанию или пытается манипулировать ими вслепую. Журналы и пестрят сравнительными тестами различных кодеков, демонстрирующими один и тот же кадр, рассматриваемый под большим увеличением. И невдомек им, что при популярном двухпроходном сжатии и дробном quantizer'e разные кадры сживаются с разным качеством (поскольку quantizer дробным не бывает и если для достижения заданного объема целевого файла кодер решил, что нужно использовать quantizer == 2.5, в практическом плане это означает, что половина кадров закодируется с quantizer == 2, а половина — с 3 и результаты теста в большой степени зависят от того какой кадр мы возьмем). Так же, при динамическом битрейте кодек стремиться отдать больше битов тем сценам, в которых по ему мнению они сильнее всего нуждаются, отнимая их у всех остальных. Алгоритм перераспределения битов у всех кодеков разный и потому сравнивая _один_ и _тот_ _же_ кадр, сжатый разными кодеками, мы можем получить драматический разрыв в качестве, но… делать глобальные выводы на этом основании нельзя!

Это, так сказать, для затравки. Чтобы пробудить исследовательский интерес. Зная, как работает кодек, мы сможем осмысленно крутить его настройки для достижения максимального сжатия с минимальной потерей качества. Источником видеоматериала может выступать и цифровая камера, и карта видео-захвата, и TV-тюнер, и куча прочего потребительского барахла… но мы (для удобства) остановимся на DVD как на самом удобном и "послушном" носителе информации.

Естественно, законы многих стран запрещают перезапись лицензионных DVD даже в домашних целях (что есть полный абсурд), но в нашей стране Фемида намного более лояльна и до тех пор, пока мы не начнем _распространять_ скопированный (и пережатый) DVD тем или иным образом, можно смело экспериментировать и ничего не опасаться.

Причин же для копирования DVD имеется как минимум две: во-первых, лицензионные DVD (да и не лицензионные тоже) сплошь и рядом записаны с теми или иными ошибками, которые неплохо бы устранить (тем более, что технически это возможно), а во-вторых, DVD-диски занимают слишком много места (геометрического) и каждый раз рыться с завалах коробок — занятие не из приятных. Намного удобнее держать фильмы на жестком диске (кстати, законы — даже Американские — это позволяют, при условии, что диск используется только как кэш и дальше его фильмы никуда не уходят).

Короче, будем считать, что я вас заинтриговал. Правда, первую часть статьи, посвященную основам теории сжатия, придется поскучать и пошевелить мозгами, зато во второй пройдет чистая практика с кучей полезных советов.

требование к читателю

Предполагается, что читатель уже имеет некоторое представление о цифровом видео и хотя бы в общих чертах понимает, чем MPEG отличается от AVI. На всякий случай, вот несколько хороших FAQ по этому поводу:

- **Introduction to MPEG**:
  - http://www.faqs.org/faqs/compression-faq/part2/section-2.html;
- **The Unofficial XviD FAQ**:
  - http://ronald.vslcatena.nl/docs/xvidfaq.html;
- **Современные методы и стандарты экономного кодирования видеоинформации:**
  - http://compression.graphicon.ru/download/articles/video/standards/state-of-the-art-video-compression.pdf;

свет и цвет

Существует множество систем кодирования светоцветовой информации, воспринимаемой человеком. Большинство методов построено на сложении (вычитании) трех или четырех основных цветов. В частности, CRT-телевизоры формируют изображение путем трех лучевых пушек — красной (Red), зеленой (Green) и синей (Blue) и соответствующая ей цветовая схема называется RGB. Она же используется и в большинстве видео-карт. Популярный режим RGB32 (True-Color 32) на кодирование каждого пикселя расходует по 8 бит, плюс еще 8 бит отводится под канал прозрачности. Итого 3*8+8 = 32.

Нетрудно подсчитать, сколько байт потребуется для кодирования "картинки" с любым разумным разрешением. Такое количество информации очень трудно сохранить и еще труднее передать по радиоканалу. А при проектировании цветных телевизоров перед разработчиками поставили задачу — любой ценой вложиться в полосу пропускания отведенную для Ч/Б телевизоров, в которую RGB никак не желала укладываться и пришлось пойти на рад хитростей.

Считается, что нормальный человеческий глаз (художников мы не берем в расчет) способен распознать до 16 миллионов цветовых оттенков, в то время как 32-битный RGB дает 4.294.967.296, что _намного_ больше, то есть явный перебор. Сколько же бит _реально_ необходимо? Прибегнув к логарифмам нетрудно подсчитать, что 24 бита кодируют 16.777.216 оттенков. Но 24 бита это все равно очень и очень много. С другой стороны, далеко не все оттенки равнозначны…

Цветоощущенье — подарок природы, появившийся на поздних стадиях эволюции (многие животные его лишены и до сих пор!), в силу чего, глаз гораздо более чувствителен к изменению ***яркости***, чем к положению в спектре (т. е. ***цветности***). Отталкиваясь от этого факта, перед передачей в эфир исходные RGB-сигналы преобразуются в сигнал яркости Y (он же Luminance или Luma) и два цветоразностных сигнала U и V (Chroma), вычисляемых по следующей формуле:

```
Y = 0.299R + 0.587G + 0.114B, U = R - Y, V = B - Y
```

Формула 1 перевод RGB-сигналов в YUV форму

Коэффициенты подобраны с учетом особенной человеческого восприятия: максимум воспринимаемой глазом световой энергии сосредоточено в зеленой (точнее, желто-зеленой) области спектра, поэтому ему выделяется наибольшее количество бит. Как видно, кодирование сигнала происходит с большими потерями, но… это неизбежная плата за узкие полосы пропускания. Радиодиапазон ведь не резиновый, а вещать хотят все…

Естественно, перед выводом на экран приходится осуществлять обратное преобразование:

```
R = Y + U, B = Y + V, G = Y - 0.509U - 0.194V
```

Формула 2 перевод YUV сигналов в RGB

Подробнее об этом можно прочитать в любой книжке по ремонту и настройке цветных телевизоров или в следующей статье: http://www.videoton.ru/Articles/Article2.html. Какое отношение имеют телевизоры к компьютерам и (тем более!) к сжатию информации? Самое прямое!!! Поскольку, сжимать в основном приходится не RGB32, сграбленный с экрана, а видеоматериал, предназначенный для эфирной трансляции, то в нем _уже_ отсутствует _вся_ информация о 2^32 оттенках и потому MPEG-кодеры работают с "телевизионными" цветовыми схемами. В MPEG1 это YUV420 (она же YUV12 и YV12), в которой значение яркости (Y) сохраняется для каждого пикселя, а цветоразностные компоненты (U&V) получаются путем усреднения значений 4'х пикселей, образующий матрицу 2x2. На все компоненты отводится по 8 бит, в результате чего на одни пиксель приходится 12 бит (отсюда и название). Главным недостатком подобной схемы становится не только низкое цветовое разрешение, но и невозможность работы с черезстрочечным (interlaced) видеоматериалом, т. к. из-за объединения соседних вертикальных линий возникают сильные искажения.

В MPEG2 используется более продвинутая цветовая схема YUV422 (она же YUY2), которая, так же, как и предыдущая сохраняет яркостной сигнал (Y) для всех точек, а вот сигнал цветности усредняется у двух соседних пикселей по горизонтали, в результате чего появляется возможность работать с черезстрочечным видеоматериалом, цветовое разрешение возрастает вдвое, а на один пиксель уже приходится 16 бит.

В MPEG4-кодеках может использоваться как та, так и другая схема. YV12 встречается гораздо чаще, поскольку обладает более высоким сжатием, экономящим 35% бит по сравнению с YUY2. Кстати, именно по этой причине многие MPEG-4 кодеки первого поколения (такие, как DivX, например) не могли работать с черезстрочечным видео и перед его сжатием приходилось выполнять операцию de-interlaced, что не только требовало время, приводило к появлению "артефактов", но и ухудшало сжимаемость, но постепенно разработчики кодеков решили эту проблему и теперь при включении одноименной опции, черные промежуточные полосы _не_ сжимаются вообще, существенно сокращая размер выходного файла.

Ладно, это все была сухая теория. Переходим к суровой практике. При сжатии MPEG4-кодеком видеоматериала, представленного в формате YUY2 (например, DVD), искажения будут происходить не только из-за потери информации о цветности, но и… ошибок преобразования YUY2 в YV12 и последующей конвертации YV12 в RGB при выводе на экран монитора. Искажения цветопередачи зачастую оказываются весьма значительными и кристально чистая небесная голубизна превращается в унылую серую грязь. Чтобы понять почему так происходит (и что сделать, чтобы этого не происходило) необходимо разобраться в этом вопросе более подробно и основательно.

Начнем, как водится с канонов. Группа **MSSG** (**MPEG S**oftware **S**imulation **G**roup) прилагает к стандарту референсную (reference – эталон) версию библиотеки mpeg2, последнюю версию которой можно скачать по адресу: ftp://ftp.mpegtv.com/pub/mpeg/mssg/mpeg2v12.zip (для доступа требуется логин и пароль) или утянуть ее прямо из института Беркли без всяких логинов и паролей: http://bmrc.berkeley.edu/ftp/pub/multimedia/mpeg/mpeg2/software/mpeg2v12.zip

В файле \src\mpeg2enc\readpic.c содержится исходный код функции readpic.c, преобразующий RGB в YUV2, ключевой фрагмент которой приведен ниже:

```
for (i=0; i<vertical_size; i++)
{
        yp = frame[0] + i*width;
        up = u444 + i*width;
        vp = v444 + i*width;

        for (j=0; j<horizontal_size; j++)
        {
                r=getc(fd); g=getc(fd); b=getc(fd);
                /* convert to YUV */
                y = cr*r + cg*g + cb*b;
                u = cu*(b-y);
                v = cv*(r-y);
                yp[j] = (219.0/256.0)*y + 16.5;  /* nominal range: 16..235 */
                up[j] = (224.0/256.0)*u + 128.5; /* 16..240 */
                vp[j] = (224.0/256.0)*v + 128.5; /* 16..240 */
        }
}
```

Листинг 1 фрагмент функции перевода RGB в YUV2, выданный из референсной библиотеки mpeg2

Первое, что бросается в глаза, это, конечно же, вещественная арифметика, которая проигрывает целочисленной и пожирает процессорные такты со страшной силой. Но это еще не самое страшное (в конце концов, оптимизация — вопрос реализации). Хуже всего то, что отображение цветового диапазона RGB на цветовой диапазон YUV2 выполняется _неправильно_ с грубыми ошибки, допущенными _умышленно_ и даже отмеченными в комментарии. Для увеличения степени сжатия, уровень каждой из компонент сужается с 0..255 до 16..235 и обратное преобразование (естественно), приходится выполнять в том же порядке. Для телевизора (и CRT-монитора времен ранней молодости MS-DOS с подсевшей трубкой), эта схема работает на ура, поскольку по краям диапазона оттенки яркости практически не различимы, а вот для современного LCD монитора с S-IPS матрицей (каким, в частности, является мой NEC 1970NX) — уровень 16 это уже не черный, а серый (или, точнее говоря, слегка белесоватый). В результате, фильмы, действие которых разворачивается во мрачных замках, куда не проникает дневной свет, смотреть становится практически невозможно. Все тени приобретают грязноватый оттенок.

Умные декодеры (к которым, в частности, относиться мной любимый FFDShow) позволяют исправить ситуацию, либо коррекцией уровней (levels), либо модификацией функции конвертации (это совсем несложно сделать, исходные тексты доступны и хорошо структурированы). Только не берите версию FFShow, лежащую на www.sourceforge.net. Она безнадежно устарела (поскольку по непонятным причинам разработчикам отрубили доступ к проекту). В настоящее время скачать самую последнюю версию FFDShow можно с сервера www.free-codecs.com (более точная ссылка не приводится, поскольку она постоянно мигрирует в широких пределах).

Естественно, референсная библиотека — это совсем не Коран и LCD мониторы появились не вчера и даже не позавчера. Другими словами, "правильный" DVD должен записываться с использованием "правильной" схемы конвертации, но… даже среди абсолютно лицензионных дисков за $15-$20 встречается куча откровенного барахла… Вот только один пример (см. рис. 1). Видите? Нижний порог яркости начинается не с нуля, а верхний — заканчивается задолго до достижения максимального насыщения (естественно, для снятия этого screenshot'a была специально выбрана соответствующая сцена из фильма, содержащая и предельно черные — в ее понимании — и предельно белые — опять-таки, в ее понимании — уровни яркости).

Рисунок 1 наблюдение за уровнями яркости в FFDShow

Таким образом, мнение, что "DVD-качество" это — эталон, выше которого не запрыгнуть, глубоко неверно. И часто случается так, что при _правильном_ "перегоне" DVD на 1/2CD мы _выигрываем_ в размере за счет потери несущественных деталей (которые все равно не видны) и в качестве — за счет устранения грубых ошибок мастеринга, сразу же бросающихся в глаза при просмотре оригинала.

Ладно, уровни это ерунда. С ними и ребенок справится (естественно, с той оговоркой, что при растяжке уровня с 16..235 до 0..255 происходит неизбежная потеря информации о яркости — отрезанные биты ведь не из воздуха берутся — хотя, в целом картинка смотрится намного естественнее). Большинство коммерческих (да и некоммерческих) кодеков для достижения максимального быстродействия используют целочисленную арифметику. Вот фрагмент функции преобразования YUV в RGB, выдранный из XviD'a, и находящийся в файле \src\image\colorspace.c. Сами же исходные тексты можно найти на официальном сайте: http://downloads.xvid.org/downloads/xvidcore-1.1.2.zip

```
#define WRITE_RGB16(ROW,UV_ROW,C1)                                          \
        rgb_y = RGB_Y_tab[ y_ptr[y_stride*(ROW) + 0] ];                      \
        b[ROW] = (b[ROW] & 0x7) + ((rgb_y + b_u##UV_ROW) >> SCALEBITS_OUT);  \
        g[ROW] = (g[ROW] & 0x7) + ((rgb_y - g_uv##UV_ROW) >> SCALEBITS_OUT); \
        r[ROW] = (r[ROW] & 0x7) + ((rgb_y + r_v##UV_ROW) >> SCALEBITS_OUT);  \
        *(uint16_t *) (x_ptr+((ROW)*x_stride)+0) = C1(r[ROW], g[ROW], b[ROW]); \
        rgb_y = RGB_Y_tab[ y_ptr[y_stride*(ROW) + 1] ];                      \
        b[ROW] = (b[ROW] & 0x7) + ((rgb_y + b_u##UV_ROW) >> SCALEBITS_OUT);  \
        g[ROW] = (g[ROW] & 0x7) + ((rgb_y - g_uv##UV_ROW) >> SCALEBITS_OUT); \
        r[ROW] = (r[ROW] & 0x7) + ((rgb_y + r_v##UV_ROW) >> SCALEBITS_OUT);  \
        *(uint16_t *) (x_ptr+((ROW)*x_stride)+2) = C1(r[ROW], g[ROW], b[ROW]);
```

Листинг 2 макрос из XviD'a, отвечающий за преобразование YUV в RGB

Чем плоха целочисленная арифметика? А тем, что ошибки округления приводят к искажению цветопередачи, которое в ряде случаев оказывается просто драматическим (особенно, если сравнивать сконвертированное изображение с оригиналом).

Для борьбы с этим явлением разработчики видео-карт придумали такую штуку как **оверлей** (***overlay mode***), позволяющую части экрана иметь цветовую схему, отличную от текущей. В overlay-mode видеоинформация проходит "транзитом" сквозь карту и поступает на монитор, не затрагивая остальные узлы, минуя в видеопамять и… освобождая центральный процессор от необходимости выполнения преобразования одной цветовой схемы в другую. Вот только… монитор, он ведь на входе совсем не YUV ожидает увидеть и потому оверлей не снимает проблему конвертации, а просто перекладывает ее на плечи карты. Причем, если программный алгоритм преобразования цветовых пространств легко поменять (выбирав из всех самый удачный), то с железом не поспоришь и карта выдает то, что в нее заложено.

Часть карт вообще не поддерживает overlay-mode, часть — поддерживает, но выдает изображение такого скверного качества, что на него практически невозможно смотреть (в особенности это относится к картам NVIDA). У почитаемого мной Matrox'a G450 с цветопередачей в режиме оверлея все нормально, но вот драйвера… при попытке перехода в overlay-mode они частенько обрушивают систему в голубой экран смерти, что совсем не добавляет оптимизма.

Стандартный Microsoft Media Player всегда стремится использовать оверлей и никой возможности отключить эту функцию у него нет (может быть и есть, где-нибудь в глубине реестра, да и то навряд ли). Плееры сторонних производителей (и в частности **BSPlayer** – http://www.bsplayer.org/) позволяют отключать оверлей установкой галочки "Force RGB mode" (см. рис. 2).

Рисунок 2 форсирование RGB-режима в BSPlayer'e

Ниже приводятся два кадра из одного и того же фильма. Первый получен в форсированном RGB-режиме, второй — в overlay-mode на карте NVIDA GeForce 6600. Как говориться, почувствуйте разницу!

Рисунок 3 кадр, полученный в RGB режиме (с кодеком FFDShow)

Рисунок 4 кадр, полученный в режиме оверлея на карте NVIDIA GeForce 6600

А разница такова, что в RGB-mode у мужика вид сильно поддатого грузчика :-) Зато в overlay-mode море и сгустившийся над ним туман приобретают грязно-серый цвет, а низ каната (если присмотреться повнимательнее!) окашивается в зеленый цвет! Уж не водорослями он успел обрасти за это время?!

Уже упомянутый кодек FFDShow имеет функцию высококачественного преобразования YV12 в RGB, активируемую установкой флажка "High quality VY12 to RGB conversion". Она отнимает дополнительные такты, но дает значительно лучший результат (см. рис. 5):

Рисунок 5 активирование функции высококачественного преобразования YV12 в RGB в кодеке FFDShow

Лицо приобретает нормальный телесный оттенок (см. рис. 6), но вот океан… Впрочем, не видя оригинала, трудно сказать насколько (не)качественно выполнено преобразование. Но по крайней мере, мы можем выбирать из всех вариантов преобразования тот, что наиболее приятен нашему глазу.

Рисунок 6 форсированный RGB-режим, кодек FFDShow (функция высококачественного цветового преобразования)

На этой ноте разборки с цветовыми схемами будем считать законченными и приступим к по-настоящему серьезным вещам.

дискретное косинусное преобразование

Фундаментом всех систем сжатия с потерями (JPEG, MP3, MPEG1/2/4) является ***дискретное косинусное преобразование*** (***D**iscrete **C**osine **T**ransform* или сокращенно **DCT**), представляющего собой разновидность косинусного ортогонального преобразования для вектора действительных чисел. Понять, что такое DTC можно на примере широко известного дискретного преобразования Фурье — ***D**iscrete **F**ourier **T**ransform* или **DFT** (дискретное косинусное преобразование представляет собой гомоморфизм его векторного пространства).

В общем случае DTC-преобразование осуществляется умножением вектора на матрицу преобразования:

$$[Y] = [C]x[X]x[C^T]$$

Формула 3 DTC преобразование, записанное в матричной форме, здесь: [X] – начальная матрица, [C] и [C^T] – матрицы с коэффициентами преобразования, где C^T означает транспонированную матрицу

Алгоритмы MPEG1, MPEG2 и MPEG4 разбивают каждый кадр (фрейм) на блоки по 8x8 пикселей, над которыми осуществляется DCT-преобразование — сначала для каждой строки, а затем для каждого столбца матрицы, поэтому такое преобразование часто обозначается как DCT8 (число "восемь" означает 8 векторов).

После выполнения дискретного косинусного преобразования в Y-матрице уже нет пикселей и она превращается в совокупность волн с различными частотами и амплитудами. Низкие частоты, сосредоточенные в левом вернем углу матрицы, соответствуют крупным деталям исходного изображения. Высокие частоты, соответствующие мелким деталям, оккупируют правый нижний угол (см. рис. 7).

Рисунок 7 наглядная демонстрация DFT- и DCT-преобразований

Обходя матрицу в зигзагообразном порядке от левого верхнего угла к правому нижнему, кодируем ее содержимое методом Хаффмана или Арифметическим кодированием или любым другим методом энтропийного кодирования, использующего коды переменной длинны. Как уже несложно догадаться, наиболее короткие коды достаются коэффициентам, расположенным в левом верхнем углу. Уже за счет одно лишь DCT-преобразования, мы добиваемся некоторого сжатия, причем это сжатие происходит ***без потерь***, что, впрочем, не дает никаких оснований для радости, поскольку степень сжатия невелика и сопоставима с обычными архиваторами.

Чтобы увеличить сжатие, необходимо отсечь наименее значимые детали, практически не воспринимаемые человеческим глазом, но кодируемые наиболее длинными кодами. После выполнения DCT-преобразование эта задача выполняется элементарно, поскольку делали уже разделены на значимые и незначимые. Как говориться отсекай и властвуй. Операция отсечения осуществляется при помощи матрицы квантования, подробно описанной в одноименном разделе, там же где описан процесс квантования, определяющий качество сжатия в целом.

Закодированная матрица (или, точнее, все, что от нее осталось, поскольку после квантования большинство коэффициентов правого нижнего угла обращаются в нули) помещается в сжатый MPEG-файл и кодировщик переходит к обработке следующего 8x8 блока.

Для вывода картинки на экран декодер, естественно, должен превратить совокупность волн в привычные нам пиксели, выполняя операцию обратного дискретного косинусного преобразования — Inverse Discrete Cosine Transformation или, сокращено, IDCT. Понятное дело, что она уже не будет соответствовать оригиналу и часть деталей окажется безвозвратно утеряна.

матрица квантования

После выполнения DCT-преобразования мы получаем матрицу, наиболее значимые детали которой сосредоточены в левом вернем углу, а наименее значимые — в левом нижнем. Возникает естественное желание — отбросить незначимые детали, сократив размер целевого файла. Это достигается путем квантования.

Квантованием *(применительно к обработке сигналов) называется деление величины сигнала на некоторое* ***целое*** *число, называемое* ***квантом*** *(**quant**)*, обычно обозначаемый буквой Q, а обратный ему процесс — **деквантированием**. С математической точки зрения X/Q = X*Q, но целочисленная арифметика "огрубляет" сигнал (см рис 8, 9), причем это "огрубление" тем сильнее, чем больше квант. При Q = 1 мы не теряем никаких деталей, при Q превышающим половину амплитуды сигнала — теряем все.

Рисунок 8 сигнал, переведенный в цифровую форму

Рисунок 9 оцифрованный сигнал после квантования

Применительно к MPEG сжатию операция квантования выполняется дважды — первый раз при наложении на DCT-матрицу специальной Матрицы Квантования (Quantization Matrix или QM) и второй раз — при квантовании коэффициентов матрицы, образующейся после наложения.

Стандартная MPEG-2 Матрица Квантования выглядит так (см. листинг 3):

```
08 16 19 22 26 27 29 34
16 16 22 24 27 29 34 37
19 22 26 27 29 34 34 38
22 22 26 27 29 34 37 40
22 26 27 29 32 35 40 48
26 27 29 32 35 40 48 58
26 27 29 34 38 46 56 69
27 29 35 38 46 56 69 83
```

Листинг 3 стандартная MPEG-2 Матрица Квантования

В литературе, посвященной сжатию, процесс наложения матрицы квантования обычно описывается слегка неточно. Утверждается, что коэффициенты исходной DCT-матрицы попарно сравниваются с коэффициентами матрицы квантования и если Y[i] < QM[i], то данный коэффициент отбрасывается и вообще не кодируется ([Y] – матрица, полученная при DCT-преобразовании). То есть, в грубом приближении, коэффициенты QM-матрицы как бы задают порог отсечения.

На самом деле, квантование происходит так: **X[i] = Y[i]/QM[i]**, где [X] – матрица, полученная после квантования (***примечание****: тот факт, что фреймы разбиваются на inter- и intra-блоки, квантование которых выполняется слегка различно, мы пока опускаем, т. к. для его объяснения требуется иметь представление о типах фреймов I, P, B и S разговор о которых еще впереди, на всякий случай отметим, что сейчас мы разбирает кодирование intra-блоков*).

Теперь становится ясно, почему левом вернем углу QM-матрицы находится небольшие числа, планомерно растущие во всех трех направлениях. Крупные детали квантуются минимально, мелкие — после квантования исчезают совсем. Увеличение коэффициентов матрицы квантования ведет к росту потерь деталей, а градиент (т. е. скорость нарастания коэффициентов по мере продвижения к трем остальным углам) задает "весовую" ценность деталей различного размера, что в конечном счете определяет резкость изображения.

Большинство MPEG-4 кодеков работают как минимум с двумя матрицами — MPEG-2 (резкое изображение, среднее сжатие) и H.263 (сглаженное изображение, высокое сжатие). Некоторые кодеки (и, в частности, XviD) поддерживают пользовательские матрицы квантования, что открывает практически неограниченные перспективы для качественного сжатия, но об этом чуть позже, а пока разберемся с одним весьма щекотливым моментом, вызывающим путаницу и недопонимая.

Пример QM-матрицы, обеспечивающий большее сжатие за счет намного более агрессивного выбрасывания деталей, приведен ниже:

```
 20  20  20  20  31  63 127 255
 20  40  40  40  63 127 255 255
 20  20  31  63 127 255 255 255
 20  31  63 127 255 255 255 255
 31  63 127 255 255 255 255 255
 63 127 255 255 255 255 255 255
127 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255
```

Листинг 4 нестандартная QM-матрица, обеспечивая очень высокое сжатие, но вместе с ним намного более значительную потерю деталей

Достоинство алгоритмов семейства MPEG в том, что они позволяют ужать видео-файл до любого требуемого размера. За счет чего этого достигается? А все за счет того же квантования. Матрица, образовавшаяся после наложения QM-матрицы, подвергается

повторному квантованию, то есть делению на некоторое целое (!) число, называемое **quantizer'ом** и обозначаемое буквой Q.

В упрощенном виде процесс квантования (включающий в себя наложение QM-матрицы выглядит так):

```
#define DIV_DIV(a,b)   (((a)>0) ? ((a)+((b)>>1))/(b) : ((a)-((b)>>1))/(b))
#define VM18P 3
#define VM18Q 4

/* divide-by-multiply table
 * needs 17 bit shift (16 causes slight errors when q > 19) */
#define SCALEBITS 17
#define FIX(X) ((1UL << SCALEBITS) / (X) + 1)
static const uint32_t multipliers[32] =
{
        0,       FIX(2),  FIX(4),  FIX(6),
        FIX(8),  FIX(10), FIX(12), FIX(14),
        FIX(16), FIX(18), FIX(20), FIX(22),
        FIX(24), FIX(26), FIX(28), FIX(30),
        FIX(32), FIX(34), FIX(36), FIX(38),
        FIX(40), FIX(42), FIX(44), FIX(46),
        FIX(48), FIX(50), FIX(52), FIX(54),
        FIX(56), FIX(58), FIX(60), FIX(62)
};

/* quantize intra-block
 *
 * const int32_t quantd = DIV_DIV(VM18P*quant, VM18Q);
 * const uint32_t  mult = multipliers[quant];
 * uint32_t level = DIV_DIV(16 * data[i], default_intra_matrix[i]);
 * coeff[i] = ((level + quantd) * mult) >> SCALEBITS;
 */
```

Листинг 5 псевдокод, иллюстрирующий наложение QM-матрицы и квантование заданным quantizer'ом (позаимствован из исходных текстов XviD'a)

Quantizer представляет собой целое число от 1 до 31 включительно. При Q = 1 мы сохраняем максимум деталей, а при Q = 31 – теряем все ("теряем все" — в пределах одного 8x8 блока, который заливается одним цветом, и мы получаем мозаику из 8x8 квадратов, из которой еще кое-что можно разобрать).

На самом деле, 31 — это _очень_ большое число и уже при Q > 6 на изображение без содрогания смотреть становится невозможно. С другой стороны, учитывая, что DVD диски обычно записываются с Q = 2, становится ясно, что для большинства видеоматериалов имеет смысл использовать только Q от 2 до 6.

Ниже представлен ряд изображений, сжатых с разными quantizer'ами для сравнения:

Рисунок 10 изображение, сжатое с Q = 1

Рисунок 11 левая половина — Q = 1, правая — Q = 6

Рисунок 12 левая половина — Q = 1, правая — Q = 13

Рисунок 13 левая половина —Q = 1, правая — Q = 31

Откуда взялись эти числа — 1 и 31? Ну, с "1" все понятно. Поскольку, на ноль делить нельзя, то 1 — это наименьший возможный quantizer, кстати говоря, не выполняющий никакого квантования, поскольку от деления на единицу коэффициенты матрицы не меняются. Но вот почему максимальный quantizer равен 31?

Ответ хранится в исходных текстах кодека MPEG-2, а точнее — в функции квантования quant_intra(), находящейся в файле \src\mpeg2enc\quantize.c, ключевой фрагмент которой приведен ниже.

```
for (i=1; i<64; i++)
{
```

```
        x = src[i];
        d = quant_mat[i];
        y = (32*(x>=0 ? x : -x) + (d>>1))/d;  /* round(32*x/quant_mat) */
        d = (3*mquant+2)>>2;
        y = (y+d)/(2*mquant); /* (y+0.75*mquant) / (2*mquant) */

        /* clip to syntax limits */
        if (y > 255) if (mpeg1) y = 255; else if (y > 2047) y = 2047;

        dst[i] = (x>=0) ? y : -y;
}
```

Листинг 6 quant_intra(\src\mpeg2enc\quantize.c

Немного подумаем. Максимальное значение индекса матрицы равно 255 (в MPEG-2 – 2047 или 7FFh). Минимальный коэффициент квантования в левом верхнем углу матрицы — 8. Тогда при quantizer >= 31 весь блок 8x8 _гарантированно_ обращается в ноль, т. е. теряется _вся_ информация, хотя, как уже говорилось выше, использование таких высоких Q бессмысленно, разве, что в экспериментальных целях, но не будем углубляться в теорию, а лучше вернемся обратно к практике.

Чем больше Q, тем выше степень сжимаемости видеоматериала, но и тем ниже его качество. Варьируя значение Q можно получить файл заданного размера, например, ужать DVD до размеров одного CD. Но часто бывает так, что при Q = 4 файл на диск никак не влезает, а при Q = 5 влезает с большим запасом.

Практически во всех кодеках, которые мне только доводилось видеть quantizer представляется дробным числом с несколькими знаками после запятой (см. рис. 14).

Рисунок 14 задание дробного quantizer'а в настойках XviD'а

Непосвященному в тонкости кодирования это кажется вполне нормальным, но после разбора приведенных выше фрагментов исходных текстов кодеков MPEG-2 и XviD возникает резонный вопрос: *как же quantizer может быть дробным, если он по жизни целый*?! **Дробных quantizer не бывает!!!** И ведь правда — не бывает. Просто не встречается в живой природе. Выбор нецелого quantizer'а приводит к тому, что часть фреймов кодируются с одним Q, а часть — с другим (см. рис. 15). Усредненное значение и даст нужное нам (псевдодробное) значение Q.

Рисунок 15 результат использования дробного quantizer'а — различные фреймы сжимаются с различным Q, и все вместе они дают усреднение (по горизонтали откладывается Q, по вертикали — количество фреймов, сжатых с данным Q, типы фреймов [I/P/B] мы рассмотрим в следующий раз)

Психофизическая модель, используемая кодеками, отталкивается от того факта, что при быстром мелькании качественных (т. е. детализированных) и некачественных (т. е. с потерей деталей) кадров, человеческий глаз, а точнее мозг, улавливает детали и игнорирует размытость, в результате чего при чередовании кадров с Q и Q+1, субъективное качество приближается к Q.

А вот это для кого как! Лично у меня субъективное качество в первую очередь обуславливается наименее качественными кадрами, вплотную приближаясь к Q+1 и даже становясь хуже его! Чередование Q и Q+1 создает некоторое, трудное передаваемое словами, ощущения "дрожания" изображения и фильм сжатый с Q+1 субъективно (опять-таки субъективно!) воспринимается более приятно, чем с Q/Q+1. Но это — сугубо индивидуальные ощущения. Тем не менее, они подтверждаются большим количеством моих знакомых. К тому же тут еще вот какое обстоятельство прослеживается: при просмотре фильма с потерянными деталями (о существовании которых мы даже не подозреваем), субъективное качество остается вполне приемлемым даже при завышенном quantizer'е, но… стоит только глазу показать несколько кадров, где эти детали сохранены, как он моментально перестраивается и меняет оценку с "приемлемой" до "неудовлетворительной". Это можно сравнить с тем, что газетный лист на кажется белым до тех пор пока рядом с ним не окажется кусок мелованной бумаги.

Отсюда **вывод**: *выбор целого значения quantizer'а обеспечит лучшее качество изображения, чем дробное.* А как же в этом случае "подгонять" сжатый файл под требуемый размер? Целочисленный quantizer — слишком уж грубое средство… Действительно грубое, ведь он квантует сигнал, уже подвергшийся квантованию!

Редактирование матрицы квантования — это, конечно, высший пилотаж, требующий опыта и глубоких теоретических познаний, иначе качество можно только ухудшить, но правильно составленная матрица квантования позволяет ужимать файл до любого (разумного) размера с максимальным качеством и с минимальным потерей деталей.

Кстати говоря, в XviD'e имеется экспериментальный "**Modulated QM**", стремящийся выбрать наиболее адекватную матрицу квантования для данного Q. Увы, алгоритм его работы постоянно меняется. Сначала он использовал H.263 матрицы (обеспечивающие лучшее сжатие, но дающие более высокую размытость) для фреймов с Q <= 3 и MPEG-2 матрицы (выше резкость, хуже сжимаемость) для фреймов с Q >= 4, что вызывало массу негативных откликов. Действительно, зачем _портить_ качество "мыльной" матрицей на низкий Q и зачем подчеркивать резкость потерянных деталей при высоких Q?! В следующий версиях стратегия модулятора изменилась на прямо противоположную и теперь он стал выбирать MPEG-2 матрицы для фреймов с Q <= 3 и H.263 матрицы для фреймов с Q >= 4, что обеспечивает лучше качество при большей степени сжатия (примечание: quantizer выбирается для каждого 8x8 блока индивидуально, хотя в подавляющем большинстве случаев весь блок кодируется с одним Q).

Подробнее о матрицах квантования мы поговорим в следующей статье, а пока вернемся с заоблачных небес на грешную землю, на которой творятся такие дела, что даже язык не поворачивается, чтобы их описать… в ***двухпроходном режиме сжатия*** (самом популярном на данный момент) кодек не позволяет задавать quantizer, требуя от нас указать средний ***битрейт*** (***average bitrate***) или же сразу желаемый размер файла.

Битрейт — это просто количество битов в секунду. Зная размер (в битах) и частоту (в fps) кадров, не составит труда рассчитать размер конечного файла, равно как и наоборот — размер файла однозначно обуславливает его битрейт. Величина самого битрейта, естественно, зависит от степени сжимаемости видеоматериала, и поскольку кодек не пророк он сжимает исходный видеоматериал за два прохода. В первом проходе никакого сжатия не выполняется, а лишь оценивается степень сжимаемости каждого фрейма, затем на основе полученной информации вычисляется необходимый quantizer, с которым видеоматериал реально сжимается во втором проходе.

Для достижения максимального качества, кодек стремится отдать больше битов тем сценам, которые сильнее всего в них нуждаются, отобрав их у остальных. Алгоритмы перераспределения битов по фильму непрерывно совершенствуются, но… до сих пор находятся в зачаточном состоянии. Наиболее типичная ошибка — выделив некоторым сценам низкий Q (соответствующий, как мы помним, высокому качеству), кодек вынужден сжимать остальные сцены с высоким Q (соответствующему низкому качеству).

Пример такого поведения продемонстрирован на рис 16. Из-за того, что кодек выделил некоторым сценам Q < 8, для достижения заданного размера, он оказался вынужден закодировать подавляющее большинство фреймов с Q = 10. Большое количество фреймов оказалось сжато с Q от 11 до 16 включительно, а некоторые фреймы кодировались с… Q = 22! И хотя средний Q = 12.15, при постоянном quantizer'e тот же самый фильм с теми же самыми настройками сжимается тем же самым кодеком за один проход с Q = 7, занимая даже чуть-чуть меньший размер и _чудовищно_ выигрывая в качестве (и это _без_ редактирования матрицы квантования и прочих шаманских ритуалов).

Попутно — чем уже "гора", образующаяся на диаграмме XviD'a — тем выше качество сжатия. Наличие "выхлестав" (т. е. нескольких несмежных пиков) указывает на то, что в первом проходе кодек неверно выполнил расчет сжимаемости видеоматериала и на определенных сценах ему катастрофически не хватило битрейта, вот он и сжал их с неоправданно завышенным Q (другая возможная причина — выбор профиля, ограничивающего максимальный пиковый битрейт). Обычно увеличение битрейта позволяет удалить лишние пики, сузив ширину "горы" и увеличив тем самым качество.

Замечено, что XviD страдает хроническим оптимизмом — к концу фильма ему постоянно недостает битрейта и он вынужден увеличивать Q, сжимая финальные сцены с гораздо худшим качеством.

Рисунок 16 диаграмма квантования фильма, сжатого с переменным битрейтом

После многочисленных экспериментов, сжав в общей сложности свыше _тысячи_ DVD-дисков, автор этой статьи полностью отказался от двухпроходного сжатия и теперь жмем фильмы в один проход с фиксированным Q.

Кодек FFDShow позволяет отображать в реальном времени quantizer для каждого из 8x8 блоков и строить диаграмму размеров сжатых фреймов, что невероятно полезно для

объективной оценки качества видеоматериала (см. рис. 17). Чтобы задействовать этот режим, откройте FFDShow (Программы → FDShow → Video decoder configuration), найдите вкладку "Visualization" и взведите галочки "Quantizers" и "Graph", не забыв отметить и саму "Visualization" (или сделайте тоже самое во время просмотра фильма, кликнув по иконке в системном трее, которую FFDShow будет отображать, если в графе "Try, Dialogs & Paths" взвести галочку напротив "Show tray icon".

Рисунок 17 визуализация quantizer'а и построение диаграммы размера фреймов в реальном времени кодеком FFDShow

Для покадрового просмотра фильма удобно использовать бесплатный видео-редактор AviDemux (http://fixounet.free.fr/avidemux). Качество соседних кадров, закодированных с различным Q меняется весьма значительно. Особенно это хорошо заметно на крупных планах, когда отчетливо видны волосы, глаза, ресницы и прочие тонкие черты лица. То есть, предполагается, что они должны быть видны, но высокий Q все "замыливает"…

>>> врезка лучше чем DVD

Quantizer равный 3 обладает одним очень интересным свойством — убирая наименее значимые детали он действует как отличный "шумодав", что весьма актуально для очистки "зашумленных" DVD (а таких среди них — большинство). Ниже приведены два кадра. Один — с оригинального DVD, другой после сжатия XviD'ом с Q = 3 и H.263 матрицей.

Рисунок 18 кадр с оригинального DVD (обратите внимание на шум)

Рисунок 19 тот же самый кадр после сжатия (куда подевался шум?!)