

СЖАТИЕ ВИДЕО БЕЗ ПОТЕРЬ НА ОСНОВЕ LZW

Скробов А.Л.¹

e-mail: tyomitch@gmail.com

При разработке системы электронного сопровождения заседания перед нами встала задача предоставить возможность участникам заседания показывать на табло зала документы, хранящиеся на их рабочем месте (в неизвестном заранее формате). Как наиболее общее решение задачи, нами было предложено производить захват экрана несколько раз в секунду и передавать снимки экрана по сети в виде изображений. Тут же возник вопрос о сжатии таким образом полученного видеопотока. Понятно, что стандартные алгоритмы видеосжатия (JPEG и т.п.) в данном случае не подходят, т.к.

- а) они рассчитаны на полноцветное видео с миллионами оттенков, тогда как в типичном снимке экрана их на порядки меньше;
- б) они неизбежно вызывают искажения изображения, незаметные в динамичных полноцветных видеофрагментах, но заметные в мало меняющихся снимках экрана с множеством мелких деталей.

Поэтому мной было принято решение использовать какой-нибудь алгоритм видеосжатия *без потерь*, рассчитанный на применение к малоцветным видео, таким как последовательности снимков экрана. Не найдя готовых реализаций таких видеокодеков, я решил написать собственную, взяв за основу LZW.

Сжатие видео состоит из трёх этапов:

- 1) сначала пиксели, в которых изображение осталось неизменным, заменяются "псевдоцветом" 0x80000000;
- 2) затем выполняется RLE-сжатие, поскольку в снимках экрана достаточно распространёнными являются случаи больших последовательностей точек одинакового цвета — например, псевдоцвета, соответствующего неизменившимся точкам кадра;
- 3) наконец, выполняется LZW-сжатие полученного материала.

¹Работа выполнена при поддержке ЗАО НПО "Уралсистем"

Дополнительно запланирован был ещё один этап между 1 и 2 — сброс нескольких нижних битов цвета для дополнительного уменьшения размера сжатых данных. Оказалось, однако, что это практически не влияет на степень сжатия малоцветных изображений, тогда как заметно ухудшает их качество. Поэтому я решил ограничиться тремя перечисленными стадиями. Сжатие, тем не менее, выполняется в один проход: перечисленные этапы применяются по очереди к каждой точке, а не к изображению целиком. Таким образом, возможно (хотя на данный момент не реализовано) сжатие кадров, не помещающихся в память целиком.

Используемое в моём кодеке LZW-сжатие не вполне совпадает с описанным в [1]: поскольку возможно 2^{24} различных значений для цвета точки, при использовании “чистого” LZW мне бы пришлось инициализировать таблицу префиксов всеми возможными значениями цвета, и начинать сжатие с кода ширины 25 бит. При этом едва ли бы мне удалось достичь заметного уменьшения размера данных. (Сжатие, используемое в GIF-файлах, именно потому ограничено 256-цветными изображениями, что предполагает инициализацию таблицы префиксов всеми возможными значениями цвета — это внутреннее ограничение “чистого” LZW. Одной из основных моих целей было снятие этого ограничения с моего алгоритма.) Поэтому я определяю 2 “спецкода”: 0 — “следующие 24 бита — значение цвета” и 1 — “следующие 16 бит — длина последовательности одинаковых пикселей”. В первый раз для элемента выводится один из спецкодов, соответствующий его типу (значение цвета либо длина последовательности) и затем его значение; в последующие разы выводится его индекс в таблице. Когда декомпрессор встречается спецкод, он читает следующие 24 или 16 бит и добавляет их в конец текущей таблицы, устанавливая, если нужно, флаг “длина последовательности”. Кроме того, таблица (первый элемент в ней имеет индекс 2) инициализируется псевдоцветом 0x80000000. Коды, большие 2, имеют обычное для LZW значение — индекс соответствующего элемента таблицы.

Хочется дополнительно подробнее рассказать об используемом варианте RLE-сжатия. Его использование позволяет получить выгоду в трёх областях:

- 1) во-первых, последовательности любой длины N сжимаются до 2 кодов (“чистый” LZW сжал бы их до $\log_2 N$ кодов);

- 2) во-вторых, сами коды получаются короче: т.к. каждый выводимый код добавляется в таблицу, и таким образом RLE экономит ячейки таблицы, реже происходит увеличение ширины кода;
- 3) наконец, в предположении обычной для снимка экрана структуры (набор прямоугольных областей-окон), даже для длины последовательности чаще всего не нужен будет новый код: он уже был добавлен при обработке крайнего ряда пикселей данной прямоугольной области несколькими строками ниже.

Дополнительное преимущество в том, что поскольку для длины последовательности выделен особый спецкод, то невозможны коллизии с теми кодами, которые соответствуют значению цвета.

Компрессия начинается с кода ширины 2 бита (в 3-ую ячейку таблицы, очевидно, попадёт цвет первого встретившегося в изображении пиксела, т.е. левого нижнего, и т.д.) Ширина кода, как и в GIF-файлах, может увеличиваться по ходу сжатия, до 32 бит (в GIF — до 12 бит). Никаких спецкодов для сброса таблицы в процессе декомпрессии не предусмотрено; предполагается, что для кодирования любого изображения будет достаточно 32-битных кодов (между кадрами таблица может сбрасываться). Кроме того, поскольку максимальный размер таблицы в моей реализации “прошит” в кодеке (т.е. она не может расти динамически) и составляет на текущий момент 655360 кодов, то уже 20-битными кодами пользоваться нельзя. Понятно, что этот максимальный размер можно задать любым в пределах доступной памяти; мои оценки, однако, показывали, что для кодирования одного кадра имеющейся таблицы достаточно.

До сих пор мой кодек рассматривался применительно к статичным изображениям. Про обработку видео хочется сделать следующие замечания:

- а) благодаря первому этапу сжатия кодируются только отличия между соседними кадрами;
- б) поскольку соседние кадры предположительно имеют сходную структуру, то таблица, полученная при кодировании предыдущего кадра, используется и для следующего, а не сбрасывается. Сброс таблицы можно выполнить явно; кроме того, она сбросится автоматически после заполнения больше чем наполовину, чтобы предотвратить её переполнение.

Так как для декомпрессии кадра нужно иметь таблицу, уже заполненную при декомпрессии предшествовавших ему кадров, то при повреждении закодированного видеопотока его восстановление будет возможно, только начиная со следующего “ключевого” кадра, т.е. закодированного со сброшенной таблицей. Поэтому было бы разумно принудительно сбрасывать таблицу, например, каждые несколько секунд. Используемая здесь концепция “ключевых” кадров - та же, что и в существующих алгоритмах видеосжатия.

Детали реализации кодека подсказаны идеями из [1]: таблица хранится в виде массива пар “номер префикса-значение цвета”, для декомпрессии используется статическая хеш-таблица (в текущей реализации — размера 1000003 с шагом 256). Коды упаковываются в байты не так, как в GIF-файлах, а наоборот — от старшего бита к младшему, для удобства чтения результата в двоичном виде (при отладке кодека мне это, действительно, помогло). Так, если в GIF-файлах упаковка ведётся следующим образом ([1]):

For example, if you're starting with 5-bit compression codes, and your first three codes are, say, <abcde>, <fghij>, <klmno>, where e, j, and o are bit#0, then your codestream will start off like:

byte#0: hijabcde

byte#1: .klmnofg

то в моей реализации они упакуются, как мне кажется, более естественным образом:

byte#0: abcdefgh

byte#1: ijklmno.

Про сравнение моего кодека с существующими хочется отметить, что с известными мне видеокодеками (вроде того же MPEG) его сравнивать некорректно, т.к. он предназначен совсем для других целей. Для тех видеоклипов, где MPEG обеспечивает наилучшее качество (миллионы уникальных цветов, нерезкие границы, соседние кадры имеют мало общих пикселей), мой кодек неприменим вовсе, и скорее всего, не обеспечит никакого сжатия. Поэтому более корректно было бы его сравнивать с GIF и PNG, которые обеспечивают сжатие без потерь. В нескольких экспериментах было проверено, что сжатие моего кодека примерно такого же уровня, что у PNG; преимущество перед GIF в том, что допускается сжатие без потерь изображений с любым количеством цветов. Фактически, в процес-

се сжатия строится минимальная “палитра”, способная вместить все цвета изображения, и ширина кода подбирается под эту палитру: другими словами, если в изображении 64 уникальных цвета, то компрессор в процессе работы дойдёт до точно такой же ширины кода, как GIF с 6-битным цветом (при этом для использования GIF количество уникальных цветов необходимо знать заранее, а для моего кодека это необязательно). Ещё одно преимущество перед GIF — способность обрабатывать изображения с более чем 256 цветами; например, для 260- или 512-цветного изображения возможно сжатие без потерь, причём в первом случае — практически с тем же результатом, что и для аналогичного (по количеству повторяющихся участков) 256-цветного изображения. Таким образом я заключаю, что мой кодек превосходит GIF по всем показателям.

24-битные снимки экрана в формате BMP сжимаются примерно до 5–15%%

И ещё о подробностях реализации. Кодек состоит из двух взаимосвязанных кусков, написанных на VB и MASM32 соответственно (поскольку VB не поддерживает ассемблерных вставок и не может создавать DLL с экспортируемыми функциями, я был вынужден оформить оба куска в виде отдельных файлов). Это VIDS-совместимый кодек, т.е. он устанавливается при помощи inf-файла в список Панель управления→Звук и мультимедиа→Оборудование→Видео кодеки→Свойства. В этом списке можно проверить, что кодек установлен, или удалить его. Также для удаления можно воспользоваться диалогом Панель управления→Установка и удаление программ. Вместе с кодеком идёт демонстрационная программа, позволяющая сжимать и разжимать отдельные изображения.

Файлы кодека можно загрузить с <ftp://cs.usu.edu.ru/util/MGIF/>

Список литературы

- [1]. *Steve Blackstock*, LZW and GIF explained.
www.cis.udel.edu/~amer/CISC651/lzw.and.gif.explained.html