**Алгоритм сжатия видео: рецепторы как кодировщики**

Дмитрий Сахань

С момента опубликования алгоритма сжатия видео Pixel Behaviour Check прошло уже больше полугода (по состоянию на январь 2003 года). Поначалу я даже написал кое-какой код для декодера этого формата. Но вскоре проблемы искусственного интеллекта поглотили мое внимание полностью. Жизнь закружила, заставляя решать совершенно другие вопросы, и увела в сторону от видеосжатия. К сожалению, чтобы довести идею до работающего коммерческого продукта, требуется уделить ей очень много времени. На это времени уже не хватало, а идея осталась ждать лучших времен.

Тогда я как-то упустил из виду, что надо было сделать доступными мои исходники декодера. По себе знаю: желание поработать с любой идеей убивает необходимость писать пробную программу, без которой невозможно проверить идею на деле. Как правило, приходится потратить уйму времени на написание взаимодействия блоков программы, прежде чем дело дойдет до проверки самой идеи. Поэтому я выкладываю исходники декодера. Можете скачать их здесь в виде RAR-архива (205,5 Кбайт). Исходники нормально документированы, так что не составит большого труда в них разобраться. В исходниках не ставилась как таковая цель: написать оптимальный по быстродействию код. Как вы вывернете код декодера, что из этого получится - это уже ваш личный вопрос. Дальнейший же разговор об этом алгоритме пойдет совсем в другом ракурсе.

Погружаясь в пучину искусственного разума, я все более поражался тем обстоятельствам, что работе живого организма вообще чужды те принципы, которыми мы руководствуемся в компьютерном мире. Пытаясь найти связующие нити между необычным устройством частей живого организма и их функциями, мне удалось сделать очень необычную находку. Вот о ней как раз и пойдет речь, так как она позволяет по-другому взглянуть на сжатие видео. Кого интересует более детальная информация об этом, читайте статью вот здесь, правда, по понятным причинам, там больше уклон в сторону искусственного интеллекта, чем в сторону сжатия видео. А мы возвращаемся к нашей теме.

**Необычное решение обычной проблемы**

Для начала хочу вкратце ознакомить вас с сутью вышеупомянутой статьи. Во время разглядывания изображения наши глаза совершают микродвижения, не заметные ни стороннему наблюдателю, ни нам лично. В среднем частота таких движений - около 100 раз в секунду. Величина смещения изображения по сетчатке глаза во время микродвижения ничтожно мала - в пределах 1-2 соседних рецепторов. То есть видимое изображение как бы дрожит на сетчатке в пределах соседних пикселей (рецепторов). Все эти микродвижения обеспечивают удержание на рецепторах сетчатки контуров всех объектов в видимом изображении. Причем рецепторы очень удачно преобразовывают изображение, и в мозг поступает уже закодированное видео.

Если вы интересовались особенностью современных алгоритмов кодирования видео, то могли обратить внимание, что MPEG и ему подобные модификации стараются выделить контуры двигающихся объектов в сцене, чтобы закодировать только их и тем самым снизить объем видеопотока. Вообще в истории сжимающих алгоритмов прослеживается изживание математических методов сжатия. Постепенно математические алгоритмы достигли того предела, после которого блок данных нельзя больше сжать, не потеряв информации. На смену им появились JPEG и MPEG. Здесь уже сжатие достигалось за счет потерь избыточной информации. В конечном итоге в MPEG добавилась еще и компенсация движения, когда отслеживаются контуры смещающихся объектов, чтобы в следующем кадре передвинуть объекты в новое положение без повторного кодирования их содержимого (которое внутри их контуров). Но беда MPEG заключалась в том, что контуры отслеживались в виде целых блоков (квадратиков), ибо отслеживать реальные контуры слишком сложно. Должно быть, вы видели, как при неуверенном приеме кабельного канала на экране телевизора изображение начинает выводиться скачущими квадратиками.

Над проблемой выделения контуров работало много специалистов. Где-то около полугода назад в Интернете появилось сообщение, что специалисты одной американской компании разработали более качественный алгоритм сжатия, основанный на весьма продвинутой математике выделения контуров. Как тогда заявлялось, контуры выделяются очень близко к реальным, так что у нового алгоритма нет особых проблем с квадратиками. Но серьезная математика, естественно, требует серьезной производительности компьютера.

И вот когда соизмеряешь все это с принципами устройства живого организма, оказывается, что достаточно лишь следовать этим принципам и никакой сложной математики не нужно вообще. Имитируем микродвижение глаза, и вот получили контурное изображение.

Забавное получается изображение. Это не негатив, хотя очень похоже. Можно заметить, как контуры объектов (лепестки, пестик) окрашены разными цветами, а внутри контуров находится черный цвет. Вот по такому изображению можно восстановить все, что было в оригинале, включая тональную окраску объектов. Для этого лишь нужно знать, в какую сторону имитировалось микродвижение глаза.

А вот на этом изображении посмотрите на цвет неба за статуей. Он не однородный, а переходит от светлого тона у плеч статуи в темный тон над ее головой. На контурном изображении вы не заметите в этих местах никаких тональных переходов, хотя они там есть. Разумеется, на темном фоне легко принять слабый переход за его полное отсутствие. В дальнейшем я объясню, что имел в виду, говоря: "не заметите в этих местах никаких тональных переходов".

Любой нацеленный на удаление избыточной информации алгоритм иногда вынужден удалять важную информацию. Вопрос только в том, насколько сильны потери. Обязательно есть места, где алгоритм справляется удачно, и есть места, где происходят большие потери. Очень проблематично найти приемлемое решение. Например, возьмем тот же пример с цветом неба за статуей. При уменьшении объема видеопотока MPEG-алгоритм начинает срезать частотный диапазон тональных переходов неба, в результате чего небо имитируется выложенными рядом квадратиками с близкими цветами.

**Выкинуть максимум, ничего не выкидывая**

Если мы посмотрим на решение природы, которое она реализовала в рецепторах сетчатки глаза, то вообще не увидим никакого удаления информации. В контурном изображении статуи действительно нет никакого тонального перехода неба, потому что переход в таком изображении закодирован, и просто так разглядеть его не удастся. Объясню это на примере. Допустим, у нас есть изображение с возрастающим цветом.

Возьмем из изображения одну горизонтальную линию. В ней 256 зеленых пикселей. Байтовые значения пикселей различаются на 1. Самый левый пиксель имеет значение 0, самый правый - 255. В итоге имеем 256 возрастающих по значениям байт.

Так вот после микродвижения глаза и последующего преобразования этой линии рецепторами сетчатки, линия превратится в набор 256 байт из одних единичек (или -1, если микродвижение глаза было в другую сторону). То есть из линии исчезнет наглядный для нас тональный переход. А происходит это потому, что рецепторы держат на своих выходах разницу между предыдущим и вновь виденным изображением. В глаз попадает изображение линии, затем происходит микродвижение, теперь в глаз попадает смещенная линия, и только после этого рецепторы выдают в мозг информацию. И каждый рецептор сигнализирует о том, что его "пиксель" отличается на 1 по значению от рядом стоящего "пикселя".

Как можно догадаться, линия из возрастающего зеленого цвета превратилась в линию со сплошным цветом со значением 1. И только зная о микродвижении, можно сказать, что эта сплошная линия обозначает линию с возрастающим цветом. По этим же причинам в контурном изображении статуи мы не увидим никаких тональных переходов неба, а лишь одни единички (двойки, тройки или насколько там сильно изменялся переход). Черный фон изображения, напоминающий о негативе, на самом деле обозначает, что в этих местах находится цвет, принадлежащий некоторому объекту в сцене. Цвет объекта закодирован разницей тонального перехода в контуре объекта. Кстати, не стоит забывать, что у объекта может быть множество внутренних подобъектов, имеющих свои контуры.

**Новая последовательность операций**

Теперь возвратимся к MPEG-алгоритму. Все его удаление избыточной информации сводится к тому, чтобы привести блок (квадратик) изображения к максимальному содержанию одинаковых по значению байт. Конечно, 256 нулей, единичек, двоек или троек сжать проще всего. На этом основаны все алгоритмы сжатия видео, только каждый из них прокладывает свой путь, чтобы превратить блок данных в набор одинаковых байт. Понятное дело, не каждый путь приводит к хорошим результатам.

В связи с этим работа алгоритмов сжатия видео следующего поколения, как мне кажется, должна выглядеть следующим образом. Сначала кадр изображения преобразуется вот таким "рецепторным" образом. Это сразу снимает избыточную информацию (посмотрите на размеры и количество черных областей в контурных изображениях). Существенный плюс - снятие избыточной информации вообще не обозначает ее потерю. Природа не зря придумывала столь изощренные и не всегда понятные "фокусы" с живыми организмами. Почему же нам не воспользоваться ее принципами. Второй плюс - "рецепторное" преобразование настолько просто реализуется, что имитация рецепторного поля не представляет сложностей на программном уровне.

Но есть и свой минус. А он заключается в том, что нам сейчас известна только маленькая находка с рецепторами. Всего лишь капля в море секретов природы. Возможно, в скором времени мы сможем проникнуть в ее секреты поглубже (имеется в виду способ обработки видеоряда), но пока остальные преобразования придется выполнять стандартными математическими средствами. Поэтому за "рецепторным" преобразованием вероятнее всего должны выполняться самые обычные операции из уже известных алгоритмов сжатия. В принципе, очищенное от избыточной информации изображение можно еще раз "очистить" уже операциями MPEG-сжатия, хотя, по моему мнению, это уже лишнее. Если уж удалось избавиться от избытка, не потеряв при этом информацию, тогда зачем терять ее дальше. Зато компенсация движения из MPEG подойдет сюда как раз кстати. Правда, ее бы немного модифицировать, чтобы уже четко выделенные контуры объектов не испортить.

**Что касается алгоритма Pixel Behaviour Check**

Дальше разговор пойдет для тех, кто захочет повозиться с доводкой идеи. У меня совсем нет времени заниматься и вопросами искусственного разума, и алгоритмом сжатия. "Рецепторное" преобразование открывает новые возможности в сжатии, поэтому его просто необходимо использовать в PBC-алгоритме. Основная фишка алгоритма - сжатие за счет контроля поведения пикселей видеокадров. Направление контроля - вдоль кадров, а не вдоль линий одиночного кадра, как в обычных алгоритмах. Думаю, такая мысль уже давно обсуждалась, просто никто не предлагал более-менее подходящее ее воплощение. В общем-то, компенсация движения из MPEG немного похожа по сути, но там ведется наблюдение за двигающимися объектами. Здесь же контролируется поведение каждого пикселя кадра, не обращая внимания на двигающиеся объекты. Вполне реально объединить компенсацию движения с контролем поведения остальных пикселей.

Конечно, структуры данных в алгоритме может быть придется слегка подправить. Во-первых, в числе первых шагов должно идти "рецепторное" преобразование видеокадра. По этим причинам могут отпасть за ненадобностью YCbCr-преобразование и перевод в процентные отношения, хотя структуры спроектированы так, что к ним можно просто "нарастить" новое преобразование, вообще не трогая структуры.

Во-вторых, коды поведений пикселей можно перестроить так, чтобы стало возможным кодировать поведение пикселей не только через массив поведений, но и с помощью дополнительных алгоритмов. Если подключать еще и компенсацию движений, тогда для нее также выделяется отдельный код. При желании коды поведений можно перестроить, не прибегая к изменению структур данных. Например, в моих исходных кодах декодера вы можете посмотреть, как я выделил определенные коды для перезагрузки массива поведений, для титров и тому подобного. Вы можете поступить точно таким же образом. Ну, тут не мне вас учить.

В-третьих, в декодере нужно будет написать обратное "рецепторное" преобразование. Оно не сложное, главное - заведомо знать, в какую сторону выполнялась имитация микродвижения глаза. Очень даже возможно, чтобы направление микродвижения могло меняться по ходу кодирования видео. Но это уже частности.

Касательно исходников декодера могу сказать, что я использовал самый простой вариант чтения байт из видеофайла. По этой причине скорость извлечения информации из видеопотока невысока. Лучше всего создать буфер в памяти, куда изначально читается большой блок файла, затем извлекая байты прямо из памяти. По мере опустошения буфера выполняется подгрузка блоков файла в память. Для этого в исходниках придется переписать процедуру чтения из файла.

Когда я экспериментировал с декодером, то заметил, что сам алгоритм декодирования очень прост и не требует серьезной производительности, хотя поначалу мне казалось, что алгоритм сильно нагрузит процессор. Оперировать многопиксельным видеопотоком при декодировании оказалось делом несложным. Совсем другое дело было при кодировании. Я набросал каркас кодировщика на скорую руку (сразу предупрежу: исходники кодировщика в связи с большим периодом времени хранения были утеряны, так что рабочие исходники, к сожалению, нет возможности предоставить). Из-за этого он кодировал видео просто с черепашьей скоростью, к тому же из него как из дырявой бочки начали сыпаться ошибки. А затем жизнь закружила: проблемы, вопросы, решения и так далее. И кодировщик выпал из моего поля зрения, так и оставшись недоработанным.

Надеюсь, вы будете более удачливы. А со своей стороны мне остается только пожелать вам удачи и достаточного времени.