УДК 004.627:004.932.2

АЛГОРИТМ ОЦЕНКИ ДВИЖЕНИЯ, АДАПТИРОВАННЫЙ ДЛЯ ОБРАБОТКИ ЭКРАННОГО ВИДЕО

Д. В. Дружинин[1]

Представлена модификация алгоритма оценки движения, разработанного для обработки экранного видео. Эта модификация отличается от аналогичных по назначению разработок большей скоростью выполнения. Приведены результаты сравнительного тестирования реализаций различных модификаций алгоритма оценки движения. Рассмотрены главные отличия экранного видео от камерного видео.

Ключевые слова: оценка движения, экранное видео, быстрые алгоритмы сжатия.

Введение. Определим понятия камерного и экранного видео. *Камерное видео* — это видео, снятое видеокамерой. *Экранное видео* — это видео происходящего на экране пользователя. Экранное видео возникает в результате фиксации активности пользователя: движение курсора мыши, скроллинг, сворачивание и открытие свернутого окна, перемещение окна, ввод текста и т.п. Зачастую о полезности программ, осуществляющих сжатие и запись на жесткий диск этого типа видеоданных, можно говорить только в том случае, когда их можно запустить на счет не только в режиме реального времени, но и в фоновом режиме, так как иначе программа, предназначенная для фиксации активности пользователя, будет мешать выполнению ресурсоемких приложений, с которыми взаимодействует пользователь.

Экранное видео — это видео высокого разрешения. Тем самым требуется применять эффективные алгоритмы сжатия такого типа видеоданных, поэтому актуальной является задача оптимизации по времени алгоритмов сжатия экранного видео. Использование алгоритма оценки движения позволяет существенно увеличить коэффициент сжатия, хотя и требует значительных вычислительных ресурсов.

Поскольку экранное видео в значительной степени отличается от камерного видео (см. раздел 1), то для использования алгоритма оценки движения при обработке экранного видео требуется его адаптация. Проблема оптимизации по времени алгоритма оценки движения для обработки экранного видео является малоизученной. Удалось найти только одну работу [1], где представлен алгоритм оценки движения, адаптированный для обработки экранного видео. Однако этот алгоритм выполняется слишком медленно (см. раздел 3), что не позволяет использовать его для сжатия экранного видео в режиме реального времени.

В настоящей статье представлена модификация алгоритма оценки движения, рассмотренного в [1], позволяющая значительно повысить скорость кодирования видеоданных. Ускорение процесса кодирования достигается за счет выявления не всех типов движений, а только некоторых из них.

1. Сравнение экранного и камерного видео. Экранное видео в значительной степени отличается от видео, снятого видеокамерой. Камерное видео запечатлевает объекты реального мира с определенной точки обзора при определенном освещении. Для этого типа видеоданных характерны плавные (непрерывно-тоновые) цветовые переходы между соседними пикселями. Это свойство камерного видео используется во многих алгоритмах сжатия графических данных (например, JPEG [7, с. 182]). Экранное видео запечатлевает искусственным образом созданные объекты, для которых характерны резкие (дискретно-тоновые) цветовые переходы между соседними пикселями.

Разумно использовать информацию об особенностях экранного видео, чтобы адаптировать алгоритм оценки движения для более эффективного выявления объектов, перемещенных на некоторое расстояние относительно предыдущего кадра.

В [1] представлен алгоритм оценки движения, адаптированный для обработки экранного видео.

Рассмотрим более подробно отличия экранного видео от камерного видео, которые наиболее ярко проявляются при использовании алгоритма оценки движения, а также особенности алгоритма [1], использующие эти свойства.

1. Для экранного видео характерны низкая степень совпадения для большинства векторов движения и полное (или почти полное) совпадение для единственного вектора, в то время как для камерного видео часто существуют несколько векторов движения, которые обеспечивают высокую степень сходства

[1] Томский государственный университет, факультет информатики, просп. Ленина, 36, корпус 2, 634050, Томск; аспирант, e-mail: dendru@rambler.ru

соответствующих блоков по заданному критерию. Поэтому в [1] используется расстояние Хэмминга для измерения степени сходства блоков, т.е. рассчитывается количество пар соответствующих пикселей одного цвета.

2. В экранном видео объекты могут перемещаться на значительное расстояние за временной отрезок, разделяющий два соседних кадра, в то время как в видео, снятом видеокамерой, обычно происходит плавное движение объектов от кадра к кадру. Поэтому в [1] используется более обширная область поиска (вплоть до поиска по всему кадру), чем при сжатии камерного видео. Например, ограничение интервала поиска вводится в [8], где представлена архитектура, позволяющая оптимизировать по времени выполнение оценки движения в кодеке MPEG. Трехшаговый алгоритм поиска соответствующего блока (Three Step Search) [4], разработанный для ускорения оценки движения в MPEG-4 кодеке, также предполагает ограничение области поиска.

3. Градиентный метод поиска оптимального вектора движения активно применяется при сжатии камерного видео [2, 5, 6]. Однако в случае экранного видео в силу дискретно-тоновой природы таких видеоданных выбор из нескольких ближайших векторов движения в приоритетном порядке в качестве текущего вектора v_1, который обеспечивает минимальное расхождение (если оно не равно 0), зачастую неэффективен. Это обусловлено тем обстоятельством, что вероятность того, что вектор движения, обеспечивающий наилучшее соответствие блоков, находится в непосредственной близости от вектора v_1, не выше, чем при проверке любого другого вектора движения. Поэтому в [1] не используется градиентный метод.

4. В экранном видео обычно значительная часть соответствующих пикселей соседних кадров совпадают. Поэтому в [1] предлагается сначала провести попиксельное сравнение текущего и предыдущего кадров на равенство. Затем оценка движения проводится только для блоков текущего кадра, изменившихся относительно предыдущего кадра.

5. В экранном видео велика вероятность соседства нескольких блоков, имеющих один и тот же вектор движения (например, при движении окна), в отличие от камерного видео, где одинаковый вектор движения для соседних блоков встречается несколько реже за счет изменения формы объекта и угла обзора при движении этого объекта.

2. Описание алгоритма оценки движения, адаптированного для экранного видео. Выявление всех движений в экранном видео оказалось слишком трудоемкой задачей для выполнения в режиме реального времени (раздел 3). В такой ситуации представляется логичной идея провести некоторую классификацию типов движений в экранном видео и разработать алгоритмы, выявляющие некоторые из этих типов движений и работающие значительно быстрее алгоритма, выявляющего все типы движений.

Можно выделить три основных вида движений в экранном видео.

1) Движения по вертикали и горизонтали (потенциально на большие расстояния). Это движения, осуществляемые вследствие вертикального и горизонтального скроллинга, нажатия пользователем на клавиши вниз, вверх, вправо, влево и пр.

2) Движения в произвольном направлении на небольшие расстояния. Это движения, осуществляемые вследствие, например, достаточно плавного перетаскивания пользователем окна.

3) Движения, осуществляемые в произвольном направлении на большие расстояния.

Как правило, подавляющее большинство движений различных объектов на экране, возникающих в ходе работы пользователя, относятся к первой либо ко второй категории. Например, такая закономерность наблюдается в обучающем видео [11].

Замечание 1. В данном случае оценка типов движения производилась визуально. В дальнейшем предполагается разработать и реализовать автоматические средства подсчета движений различных типов.

Замечание 2. Источников, где бы рассматривались подобные техники применительно к экранному видео, найти не удалось.

Использовав приведенную классификацию движений в экранном видео, удалось разработать следующую схему кодирования на основе алгоритма оценки движения, описанного в [1].

Последовательно выполняются две модификации алгоритма оценки движения. Модификация 1 осуществляет поиск блока, соответствующего текущему блоку, только по вертикали и по горизонтали, т.е. эта модификация не рассматривает векторы движения с одновременно ненулевыми сдвигами по оси x и y. Модификация 2 осуществляет поиск блока, соответствующего текущему блоку, во всех направлениях, но только в ближайшей окрестности текущего блока. Таким образом, выполнение алгоритма, осуществляющего полный перебор возможных векторов движения (предложенный в [1]), заменяется на выполнение двух менее трудоемких алгоритмов. Естественно, что при этом будут выявлены только движения первого

и второго типов.

Было принято решение кодировать блоки, для которых не найдено точное соответствие, независимо от других кадров (как блоки ключевых кадров — I-кадров). При обработке камерного видео алгоритмом оценки движения в случае нахождения вектора движения, обеспечивающего частичное совпадение блоков, для отличающихся пикселей текущего кадра вычисляется разность их цветов с цветами соответствующих пикселей предыдущего кадра [9, с. 346]. Для обеспечения эффективного сжатия на дальнейших этапах кодирования используется тот факт, что эти разности чаще всего близки к нулю (в силу непрерывно-тоновой природы таких видеоданных). Например, когда плавно происходят следующие изменения на экране: меняется освещение объекта, объект поворачивается в каком-то направлении, объект приближается или отдаляется, то блоки текущего и предыдущего кадров, содержащие этот объект, не будут идентичны, но соответствующие пиксели этих блоков будут иметь близкие значения.

В случае экранного видео это предположение относительно разностей цветов отличающихся пикселей неверно, так как для экранного видео нетипичны перечисленные выше изменения на экране; различия цветов соответствующих пикселей отличающихся частей сравниваемых блоков могут принимать произвольные значения (в силу дискретно-тоновой природы таких видеоданных). Возможно, в дальнейшем будет найден способ эффективно сжимать цвета отличающихся пикселей при частичном совпадении блоков для экранного видео.

Для ускорения работы алгоритма оценки движения при обработке экранного видео предлагается использовать следующую технику.

При сравнении двух блоков (текущего блока текущего кадра и блока предыдущего кадра, соответствующего текущему пробному вектору движения) сначала происходит сравнение диагональных элементов этих блоков. Только в случае, когда все диагональные элементы попарно равны между собой, происходит сравнение всех элементов этих блоков. Такая техника приводит к ускорению выполнения алгоритма оценки движения, так как она одновременно учитывает и горизонтальную, и вертикальную корреляцию пикселей движущегося объекта, а не горизонтальную (вертикальную) и затем вертикальную (горизонтальную) корреляцию, как это делается при полном попиксельном сравнении двух блоков. Эта техника используется в обеих предложенных модификациях алгоритма оценки движения.

Рассмотрим шаги модификаций 1 и 2 (они отличаются только областью поиска вектора движения).

Первым шагом является инициализация используемых структур данных, состоящая в следующем.

а) Начальное заполнение массива foundArray (0 — вектор движения для блока с этим индексом еще не найден, 1 — найден). При инициализации элементы массива foundArray устанавливаются в 1 для тех блоков, которые не изменились по сравнению с предыдущим кадром.

б) Вычисление минимального прямоугольника minimalRectangle, охватывающего все изменившиеся относительно предыдущего кадра области текущего кадра. Именно в пределах этого прямоугольника в дальнейшем будет осуществляться поиск соответствия для текущего блока.

в) Заполнение массива initialShifts значением (0; 1). В этом массиве хранятся векторы движения, с которых начинается проверка для конкретного блока.

На втором шаге для каждого блока, для которого соответствующий элемент массива foundArray равен 0, выполняется следующая последовательность действий.

а) Текущий пробный вектор движения принимает значение начального вектора движения для этого блока. Это может быть начальный вектор движения по умолчанию (1; 0) или некоторое другое значение, установленное уже в ходе выполнения алгоритма. Во втором случае после проверки начального вектора движения текущий пробный вектор устанавливается в начальное значение по умолчанию.

б) Сравнение диагональных элементов текущего блока текущего кадра и выбранного для проверки блока предыдущего кадра.

в) В случае равенства всех пар соответствующих диагональных элементов выполняется полное попиксельное сравнение двух блоков. Если блоки оказались равными, то соответствующий элемент массива foundArray устанавливается в 1. Начальные векторы движения для всех блоков, соседствующих с данным, устанавливаются равными текущему вектору движения. Таким образом происходит предсказание вероятного вектора движения для соседних блоков. Затем происходит переход к следующему блоку. Если же блоки не равны, то происходит изменение значения текущего вектора движения и проводится новая проверка на равенство блоков. Указанные действия выполняются до тех пор, пока не будет найден блок предыдущего кадра, равный текущему блоку, или не произойдет выход за границы minimalRectangle, или не произойдет выход за пределы ближайшей окрестности текущего блока (для модификации 2).

Замечание 3. Проверка векторов движения осуществляется в следующем порядке. Алгоритм поиска блока, соответствующего текущему блоку, только по вертикали и по горизонтали на каждом шаге

проверяет векторы движения $(0; i)$, $(i; 0)$, $(0; -i)$, $(-i; 0)$ и увеличивает счетчик i на 1. Начальное значение i равно 1. Модификация 2 алгоритма оценки движения перебирает возможные векторы движения построчно в диапазоне $\big[(i-\text{maxXShift}; j-\text{maxYShift}); (i+\text{maxXShift}; j+\text{maxYShift})\big]$, где $(i; j)$ — текущая позиция в текущем кадре, а (maxXShift; maxYShift) — максимальные отклонения от текущей позиции по оси x и y соответственно. При увеличении значений maxXShift и maxYShift алгоритм оценки движения способен выявить больший процент движений, но при этом возрастает время выполнения алгоритма, и наоборот. Один из способов вычисления maxXShift и maxYShift — по формуле $n \times \text{blockSize}$, где n — некоторая константа, а blockSize — размер "стороны" блока пикселей (4 для блока размером 4×4, 8 для блока размером 8×8 и т.д.). При тестировании использовался именно этот способ вычисления максимальных отклонений.

Замечание 4. Техники предсказания вероятного вектора движения для блоков, соседствующих с блоком, для которого найдено соответствие, применяются и для сжатия камерного видео. Например, подобная техника применяется в [10], где описана методика построения промежуточных кадров видеопоследовательности.

Таблица 1

Движения первого типа по классификации, приведенной в разделе 1 (движения по вертикали и горизонтали потенциально на большие расстояния). Для возникновения движения такого типа использовался скроллинг

№	Алгоритм / параметр	Время выполнения, мс	Количество блоков, шт.
1	Алгоритм оценки движения, представленный в [1]	10049	1436
2	Алгоритм оценки движения, представленный в [1], с добавленной начальной проверкой диагональных элементов	4012	1436
3	Модификация 1 алгоритма оценки движения, представленная в этой работе (поиск блока, соответствующего текущему блоку, осуществляется только по вертикали и по горизонтали)	24	1436
4	Модификация 1 алгоритма оценки движения, представленная в этой работе, без начальной проверки диагональных элементов	65	1436
5	Модификация 2 алгоритма оценки движения, представленная в этой работе (поиск блока, соответствующего текущему блоку, осуществляется во всех направлениях, но только в ближайшей окрестности текущего блока)	510	697
6	Модификация 2 алгоритма оценки движения, представленная в этой работе, без начальной проверки диагональных элементов	1094	697
	$3 \to 5$	$24 \to 10$	$1436 \to 0$
	$5 \to 3$	$510 \to 15$	$396 \to 830$

3. Результаты тестирования. Поскольку не удалось найти реализацию алгоритма оценки движения, представленного в [1], то в тестировании принимала участие реализация, созданная автором этой работы. При тестировании был выбран размер блока 16×16 пикселей. Максимальные отклонения по оси x и y для алгоритмов-участников тестирования 5 и 6 рассчитывались по формуле $n \times \text{blockSize}$ (см. предыдущий раздел), при этом было выбрано значение n, равное 3.

При тестировании каждый кадр имел разрешение 1024×768 и глубину цвета в 32 бита.

Тестирование проводилось на платформе со следующими характеристиками: процессор Intel Core 2 Duo E6750 2,66 ГГц, оперативная память DDR2 2Гб, операционная система Windows XP.

Замечание 5. Замеряемый параметр "Количество блоков" означает количество блоков, для которых было найдено соответствие данной реализацией. Суммарное количество блоков равно $(1024 \times 768)/(16 \times 16) = 3072$, при этом надо учитывать, что часть блоков не изменились по сравнению с предыдущим кадром.

Замечание 6. Запись $3 \to 5$ означает, что текущий кадр сначала обрабатывается реализацией 3

алгоритма оценки движения, а затем реализацией 5.

В табл. 1 и 2 представлены результаты тестирования при двух типах движений.

Таблица 2

Движения второго типа по классификации, приведенной в разделе 1 (движения в произвольном направлении на небольшие расстояния). Для возникновения движения этого типа использовалось плавное перетаскивание окна

№	Алгоритм / параметр	Время выполнения, мс	Количество блоков, шт.
1	Алгоритм оценки движения, представленный в [1]	3561	1384
2	Алгоритм оценки движения, представленный в [1], с добавленной начальной проверкой диагональных элементов	2422	1384
3	Модификация 1 алгоритма оценки движения, представленная в этой работе (поиск блока, соответствующего текущему блоку, осуществляется только по вертикали и по горизонтали)	63	450
4	Модификация 1 алгоритма оценки движения, представленная в этой работе, без начальной проверки диагональных элементов	140	450
5	Модификация 2 алгоритма оценки движения, представленная в этой работе (поиск блока, соответствующего текущему блоку, осуществляется во всех направлениях, но только в ближайшей окрестности текущего блока)	47	1384
6	Модификация 2 алгоритма оценки движения, представленная в этой работе, без начальной проверки диагональных элементов	62	1384
	$3 \to 5$	$62 \to 47$	$463 \to 962$
	$5 \to 3$	$47 \to 16$	$1379 \to 38$

Обычно экранное видео требуется сжимать в режиме реального времени. Очевидно, что алгоритм оценки движения, представленный в [1], не применим для таких целей, так как один кадр обрабатывается от 3,5 секунды до 10 секунд. Время выполнения реализации 5 в значительной степени варьируется в зависимости от типа движения и находится в интервале от 47 до 510 мс. Быстрее всего выполняется реализация 3, что позволяет использовать ее при сжатии экранного видео в режиме реального времени. Однако реализация 3 не в состоянии выявить движения второго типа, поэтому имеет смысл рассмотреть комбинации реализаций 3 и 5.

Время последовательного выполнения реализаций 3 и 5 (в любом порядке) при обоих типах движения оказывается меньше, чем сумма времени выполнения этих реализаций. Это объясняется тем, что соответствие для части блоков находит первая из выполняемых реализаций, уменьшая тем самым размер входных данных для второй реализации. Такой подход дает наилучшие результаты, когда первой выполняется реализация, созданная для данного типа движения. Например, при скроллинге время выполнения последовательности реализаций $3 \to 5$ значительно меньше, чем последовательности $5 \to 3$. Если учитывать время выполнения при обоих типах движения, то наилучшей по этому критерию является последовательность $3 \to 5$. Время работы этой последовательности более стабильно при различных типах движения и не превышает 109 мс. Таким образом, при записи 10 кадров в секунду это время близко к показателю, обеспечивающему сжатие в режиме реального времени. Вполне возможно, что при использовании более мощной аппаратной платформы время выполнения этой последовательности не превысит 100 мс, что позволит использовать такую схему сжатия в режиме реального времени. При этом последовательно выполняемые реализации 3 и 5 находят соответствие для практически такого же количества блоков, что и реализация 1.

Как следует из результатов тестирования, начальная проверка диагональных элементов уменьшает время выполнения оценки движения для всех рассмотренных реализаций.

Замечание 7. Алгоритм компенсации движения, представленный в [1], выполняется очень быстро,

порядка нескольких миллисекунд для одного кадра (время может варьироваться в зависимости от количества блоков, для которых необходимо провести компенсацию движения), поэтому он не нуждается в оптимизации.

4. Заключение. Проведение классификации движений в экранном видео сделало возможным поиск заданных типов движений. Разработан алгоритм оценки движения, использующий информацию о том, что одни типы движений встречаются в экранном видео чаще, чем другие. Просматриваются лишь области, соответствующие типам движений, которые встречаются чаще всего. Такое сужение области поиска позволяет, как показали результаты тестирования, значительно ускорить выполнение оценки движения по сравнению с алгоритмом, выявляющим все типы движений, что в свою очередь позволит в перспективе применить представленный алгоритм для сжатия экранного видео в режиме реального времени.

Предложенная схема сжатия позволяет ускорить выполнение оценки движения десятикратно при незначительных потерях в количестве распознанных движений, так как бóльшая часть движений в экранном видео относятся именно к первому или ко второму типам. Поэтому такой алгоритм оценки движения может быть использован на практике.

Хотя и удалось в значительной степени повысить скорость выполнения алгоритма оценки движения, представленного в [1], загрузка центрального процессора при этом остается высокой. Наилучшим вариантом был бы запуск процесса сжатия экранного видео в фоновом режиме, что позволило бы пользователю продолжать привычную деятельность на своем компьютере. Для этого планируется в дальнейшем перенести выполнение представленной в этой работе схемы сжатия на видеокарту с использованием технологии NVidia CUDA [3].

СПИСОК ЛИТЕРАТУРЫ

1. Motion estimation/compensation for screen capture video (http://www.freepatentsonline.com/7224731.html).
2. Movement estimation system for video signals using a recursive gradient method (http://www.freepatentsonline.com/4695882.html).
3. NVIDIA CUDA Compute Unified Device Architecture. Programming Guide (http://developer.download.nvidia.com/compute/cuda/1_1/ NVIDIA_CUDA_Programming_Guide_1.1.pdf).
4. *Agha S.* Algorithms and VLSI architectures for MPEG-4 motion estimation (http://www.lboro.ac.uk/departments/el/research/esc-miniconference/papers/agha.pdf).
5. *Timoner J.* Multi-image gradient-based algorithms for motion estimation (http://people.csail.mit.edu/samson/papers/Timoner_Freeman_MultiImageGradient.pdf).
6. *Keller Y.* Fast motion estimation using bidirectional gradient methods (http://www.eng.biu.ac.il/~kellery1/publications/pdf/optical_flow_ieee_final.pdf).
7. *Сэломон Д.* Сжатие данных, изображений и звука. М.: Техносфера, 2006.
8. *Zandonai D.* An architecture for MPEG motion estimation (http://www.iberchip.org/VII/cdnav/pdf/38.pdf).
9. *Ватолин Д.* Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео. М.: Диалог-МИФИ, 2003.
10. *Дворкович А.* Методика построения промежуточных кадров видеопоследовательности (http://www.autex.spb.ru/cgi-bin/download.cgi?dspa2004_1_48).
11. Резервное копирование данных (http://screentube.ru/2009/02/10/резервное-копирование-данных).

Поступила в редакцию
25.04.2009