УДК 004.93'1; 004.932

А.И. Кравчонок

АЛГОРИТМ БЫСТРОГО ВЫЧИСЛЕНИЯ ОПТИЧЕСКОГО ПОТОКА ПРИ ПОМОЩИ SSE2-ИНСТРУКЦИЙ ПРОЦЕССОРОВ СЕМЕЙСТВА x86

Представлен алгоритм быстрого вычисления оптического потока при помощи SSE2-инструкций на персональном компьютере. Алгоритм имеет константную сложность в зависимости от радиуса окна оптического потока, применяет SSE2 SIMD-инструкции на всех этапах вычислений, при работе на многоядерных процессорах использует параллельный режим работы. Алгоритм позволяет значительно ускорить вычисление оптического потока, что делает возможным его применение в режиме реального времени на персональных компьютерах.

Введение

Оптический поток – это смещение каждой точки данного изображения по отношению к предыдущему изображению на видеопоследовательности. Вычисление оптического потока – одна из центральных проблем в компьютерном зрении. Различные методы решения данной задачи находят применение во многих приложениях, таких как навигация роботов [1], отслеживание объектов (людей, транспортных средств) [2–4], видеонаблюдение и контроль доступа [5–7], сжатие видепоследовательностей [8–12], стереовосстановление [1, 13–22] и др. Многие приложения нуждаются в вычислении оптического потока в режиме реального времени, однако достичь такой производительности на персональном компьютере (ПК) все еще сложно. Существующие методы быстрого вычисления плотного поля оптического потока [1, 13, 23, 24] не позволяют использовать режим реального времени на персональных компьютерах при достаточно больших размерах изображения и достаточно больших возможных смещениях объектов на изображениях, не жертвуя при этом точностью вычислений и не используя различные упрощения и приближения [11, 12, 25].

При этом необходимо заметить, что наиболее часто применяемыми разрешениями видеопоследовательностей в настоящее время являются 320×240, 640×480, 720×576 при использовании максимально возможных смещений пикселов изображения в пределах ±8, ±16 по осям X и Y. Можно также заметить, что с повышением разрешения зачастую необходимо увеличивать и значения максимально возможных смещений пикселов изображения. Вычислительной мощности современных процессоров персональных компьютеров при использовании известных алгоритмов недостаточно для вычисления оптического потока в режиме реального времени при указанных разрешениях и максимально возможных смещениях пикселов изображения. Тенденции в развитии современных процессоров указывают на то, что в ближайшем будущем повышение их производительности будет достигаться за счет увеличения количества их вычислительных ядер, которые могут выполнять вычисления в параллельном режиме. Таким образом, на данный момент актуальной является разработка алгоритма вычисления оптического потока, который может эффективно выполняться в параллельном режиме на многоядерных процессорах.

В статьях Сана [23, 24] предлагается алгоритм вычисления оптического потока, который позволяет получить плотную карту оптического потока при помощи быстрой корреляции и техники поиска минимального 2D- либо 3D-пути в 3D-объеме куба коэффициентов корреляции. Быстрая корреляция вычисляется при помощи алгоритма бегущего окна, который инвариантен к размеру окна корреляции. Автор рассматривает такие меры сходства окон, как нормированная центрированная корреляция (ZNCC – zero mean normalized cross-correlation), сумма квадратов разности (SSD – sum of squared difference), сумма модулей разности (SAD – sum of absolute difference). Однако даже при таком подходе алгоритм не позволяет достичь производительности реального времени для сколь-нибудь существенного количества возможных сдвигов (области допустимых скоростей движущихся объектов).

В статьях [5, 6, 26] предлагается использовать алгоритм Сана для вычисления оптического потока в задаче поиска и отслеживания движущихся объектов в режиме реального

времени. Для ускорения вычислений применяются команды MMX (multimedia extensions) и SSE (streaming SIMD extensions). Авторам удается достичь производительности реального времени при использовании небольших разрешений видеопоследовательностей и небольшом количестве рассматриваемых возможных сдвигов объектов, они применяют MMX- и SSE-команды для ускорения лишь наиболее трудоемких частей алгоритма вычисления оптического потока.

В статье [25] представляется алгоритм вычисления оптического потока почти в режиме реального времени на обычном оборудовании без специальных аппаратных средств. Однако алгоритм применяет пирамиду изображений, что в некоторых задачах (например, обнаружения и отслеживания быстрых объектов небольшого размера) может снижать точность результатов.

Многие быстрые алгоритмы стереовосстановления могут быть применены для вычисления оптического потока, среди них алгоритмы Фогераса и соавт. [1], Канады и соавт. [13], Сана [14–19], а также алгоритмы, описанные в статьях [21, 22]. Однако практически все из них используют эпиполярное ограничение, которое значительно сокращает количество возможных сдвигов при вычислении соответствия окон корреляции и позволяет достичь производительности реального времени только в задаче стереовосстановления. В задаче вычисления оптического потока применить эпиполярное ограничение не представляется возможным, так как оно не выполняется – в поле зрения камеры объекты могут двигаться в произвольных направлениях. Например, если в задаче стереовосстановления рассматриваются возможные смещения точек вдоль эпиполярных линий в пределах ± 16, то необходимо рассмотреть только 33 сдвига, при таком же диапазоне смещений по осям X и Y для оптического потока необходимо вычислить уже 1089 оценок для соответствующих смещений. Таким образом, задача вычисления оптического потока в отличие от задачи стереовосстановления является вычислительно значительно более сложной и все указанные выше алгоритмы не позволяют решать ее в режиме реального времени для разумной области возможных смещений объектов.

Многие современные алгоритмы сжатия видео (MPEG-1/2/4/7 от moving picture experts group и H.26x-1/2/3/4 от video coding experts group) применяют оценку движения при помощи сопоставления блоков. Сопоставление блоков фактически дает несколько разреженную карту оптического потока, так как вектор смещения вычисляется не для всех точек изображения, а для некоторого количества непересекающихся блоков (чаще всего квадратных, наиболее часто используются размеры 4×4, 8×8, 16×16), на которое разбивается изображение. Алгоритм сопоставления блоков имеет значительно меньшую вычислительную сложность по сравнению с оптическим потоком, так как вычисляет векторы смещения для значительно меньшего количества точек и при реализации на SSE2 может работать в режиме реального времени [8, 9]. Оптический поток имеет большую вычислительную сложность, так как вычисления производятся не для отдельных точек, а для всех точек изображения, поэтому применение алгоритмов оценки движения при помощи SSE2 не позволяет достичь скорости реального времени. Однако можно заметить сильную избыточность вычислений при применении алгоритмов оценки смещения блоков для вычисления оптического потока за счет пересечения соседних окон, чего не наблюдается при оценке смещения непересекающихся блоков.

На данный момент широкое применение оптического потока ограничивается его высокой вычислительной сложностью, поэтому многие авторы применяют для вычисления оптического потока приблизительные методы [11, 12, 25]. Зачастую при вычислении оптического потока учитывают специфику задачи и вычисляют его не для всех точек изображения, а только для движущихся областей либо некоторых ключевых точек изображения [3, 7, 27–30]. При этом, вычисляя вместо плотной карты векторов движения только смещения отдельных областей или точек, не всегда удается достичь производительности реального времени без дополнительных ухищрений и ограничений.

В настоящей статье предлагается алгоритм быстрого вычисления оптического потока при помощи SSE2-команд процессоров семейства x86, который может выполняться в режиме реального времени на персональных компьютерах.

1. Эффективная реализация алгоритма оптического потока

Среди различных методик вычисления оптического потока можно выделить несколько наиболее часто применяемых [25, 31]:

– дифференциальный подход (differential) [31–33];

– алгоритмы поиска наилучшего соответствия областей (region-based matching) [31];

– частотные алгоритмы (energy based) [31];

– фазовые алгоритмы (phase based) [31].

Частотные алгоритмы дают малую точность вычислений [31]. Фазовые алгоритмы имеют высокую вычислительную сложность [31], а градиентные методы оптического потока, например метод Лукаса – Канаде [33], достигают высокой точности для сцен с малыми пиксельными смещениями (менее 1–2 пикселов на кадр), при этом хуже работают для больших смещений [11, 34]. Однако, например, в задачах обнаружения и отслеживания движущихся объектов на видеопоследовательностях, а также в задачах стереовосстановления смещения объектов от кадра к кадру могут быть большими (вплоть до нескольких десятков пикселов).

Из описанных методов наиболее простым и легко реализуемым является алгоритм поиска наилучшего соответствия областей, который широко применяется для решения задач компенсации движения при сжатии видео [10–12], поиска движущихся объектов [35, 36], стереореконструкции [1, 13–19, 21, 22]. Значительным плюсом алгоритма поиска наилучшего соответствия областей является возможность его реализации при помощи SSE2-команд, что значительно ускоряет вычисления [8, 9].

При существующих вычислительных мощностях персональных компьютеров вычисление оптического потока в режиме реального времени является сложной задачей, поэтому для ее решения были разработаны различные подходы:

подвыборка – вычисление оптического потока не во всех точках изображения. Такой подход могут использовать, например, алгоритмы на основе сопоставления блоков изображений [10–12, 35, 36], а также алгоритмы, вычисляющие оптический поток только в областях интереса [3, 7, 27–30];

иерархический метод – последовательное вычисление оптического потока на пирамиде изображений [33, 34];

различные аппроксимации оптического потока – например трехшаговый поиск (TSS – tree-step search), четырехшаговый поиск (FSS – four-step search), логарифмический поиск (LS – logarithmic search). Вычисления ускоряются за счет оценки не всех возможных сдвигов, а только тех, которые наиболее вероятно (в соответствии с некоторым предположением) дадут лучшую оценку [11, 12];

отсечения – проверка выполнения некоторого условия при вычислении оптического потока, которое позволяет отсеять заведомо плохие оценки до выполнения всех вычислений [12, 21, 22, 37];

вычисление оптического потока при помощи бегущей суммы – вычисление оценок для сдвигов данного окна выполняется на основе оценок, полученных для окна, соседнего с ним, что позволяет избежать большого количества лишних вычислений [1, 13, 23, 24].

Описанные подходы либо не дают точного вычисления оптического потока (подвыборка, иерархический метод, аппроксимации), либо не позволяют достичь скорости реального времени (отсечения, бегущая сумма).

Для ускорения вычисления оптического потока можно применять SIMD (single instruction, multiple data)-команды, например SSE2, что позволяет значительно ускорить вычисления [5, 6, 8, 9, 29, 30]. Возможны различные варианты использования подобных команд в зависимости от применяемого алгоритма вычисления оптического потока. Использование SSE2-команд для описываемого в статье алгоритма позволяет значительно ускорить вычисление оптического потока.

2. Алгоритм поиска наилучшего соответствия областей для вычисления оптического потока

В алгоритме поиска наилучшего соответствия областей вычисление оптического потока из первого изображения во второе осуществляется путем подбора для каждого пиксела второго

изображения наилучшего соответствия среди пикселов первого изображения в пределах заданного интервала возможных сдвигов. Оценки схожести пикселов вычисляются путем сравнения фиксированной окрестности пиксела (чаще всего квадратной или прямоугольной формы) на втором изображении со сдвинутой окрестностью в первом. Для каждого пиксела изображения осуществляются всевозможные сдвиги его окрестности в некоторых заранее заданных пределах, и для каждого сдвига подсчитывается оценка соответствия окрестности из второго изображения и той части первого изображения, на которое она легла. Существуют различные оценки для определения наилучшего соответствия двух частей изображений, чаще всего применяются следующие [1, 38]:

– корреляция (correlation between the two blocks – COR):

$$COR = \sum_{j=1}^{M}\sum_{i=1}^{N}\left(P_1(i,j)\times P_2(i,j)\right);$$

– сумма квадратов разности (sum of squared difference – SSD):

$$SSD = \sum_{j=1}^{M}\sum_{i=1}^{N}\left(P_1(i,j) - P_2(i,j)\right)^2;$$

– сумма модулей разности (sum of absolute difference – SAD):

$$SAD = \sum_{j=1}^{M}\sum_{i=1}^{N}\left|P_1(i,j) - P_2(i,j)\right|,$$

где P_1 и P_2 – прямоугольные части некоторых изображений размера $M \times N$, а $P_1(i,j)$ и $P_2(i,j)$ – яркости пикселов P_1 и P_2 с соответствующими координатами.

При поиске наилучшего соответствия первая мера максимизируется, а вторая и третья минимизируются.

С вычислительной точки зрения наилучшей мерой является SAD вследствие следующих причин:

– имеет более простую формулу;

– обладает возможностью эффективной реализации при помощи SSE2-команд;

– результат вычисления модуля разности занимает меньше места в памяти, чем квадрат разности и помещается в один байт, что позволяет обрабатывать параллельно большее количество пикселов.

При использовании SAD качество найденного оптического потока получается несколько худшим, чем при использовании других мер [13, 38], однако применение SAD при незначительном ухудшении качества оптического потока позволяет значительно повысить скорость вычислений [13]. Поэтому при вычислении оптического потока в качестве меры различия областей изображений будем использовать SAD.

Пусть I_1 и I_2 – два последовательных изображения размера $IW \times IH$, $p_1(x,y)$ – пикселы изображения I_1, $p_2(x,y)$ – пикселы изображения I_2, где $0 \le x < IW$, $0 \le y < IH$, а $I_1(x,y)$ и $I_2(x,y)$ – яркости пикселов соответствующих изображений. Обозначим через $W_1^r(x,y)$ прямоугольную окрестность пиксела $p_1(x,y)$ размера $(2r+1)\times(2r+1)$:

$$W_1^r(x,y) = \{p_1(x',y') \mid x - r \le x' \le x + r, y - r \le y' \le y + r\}.$$

Окрестность $W_2^r(x,y)$ пиксела $p_2(x,y)$ определяется аналогично. Определим R как максимально возможное смещение пиксела $p_2(x,y)$ изображения I_2 по отношению к предыдущему изображению последовательности I_1. Тогда $S = \{(u,v) \mid -R \le u,v \le R\}$ – множество возмож-

ных векторов движения (u,v) пиксела $p_2(x,y)$. Пусть $W_1^r(x+u,y+v)$ и $W_2^r(x,y)$ – окрестности радиуса r пикселов $p_1(x+u,y+v)$ и $p_2(x,y)$, где $(u,v)\in S$. Определим меру отличия двух окрестностей $W_1^r(x+u,y+v)$ и $W_2^r(x,y)$:

$$SAD_{(x,y)}(u,v)=\sum_{j=-r}^{r}\sum_{i=-r}^{r}\left|I_1(x+i+u,y+j+v)-I_2(x+i,y+j)\right|.$$

Тогда будем искать для пиксела $p_2(x,y)$ его вектор движения $(\hat{u},\hat{v})$ как вектор (u,v), для которого оценка $SAD_{(x,y)}(u,v)$ минимальна:

$$(\hat{u},\hat{v})=\arg\min_{(u,v)\in S} SAD_{(x,y)}(u,v).$$

В такой постановке задачи вычисления векторов движения возможно использование только окрестности пикселов нечетного диаметра, что не всегда удобно. При необходимости использования окрестности пиксела четного диаметра будем задавать окрестность

$$W_1^r(x,y)=\{p_1(x',y')\,|\,x-r\le x'<x+r,y-r\le y'<y+r\}.$$

Окрестность четного диаметра $W_2^r(x,y)$ пиксела $p_2(x,y)$ определяется аналогично. Тогда вычисление $SAD_{(x,y)}(u,v)$ для окрестностей с четным диаметром необходимо выполнять следующим образом:

$$SAD_{(x,y)}(u,v)=\sum_{j=-r}^{r-1}\sum_{i=-r}^{r-1}\left|I_1(x+i+u,y+j+v)-I_2(x+i,y+j)\right|.$$

С помощью указанных формул оптический поток можно вычислить не для всех пикселов изображения I_2, а только для тех, для которых при любом возможном сдвиге $(u,v)\in S$, $S=\{(u,v)\,|\,-R\le u,v\le R\}$, и заданном радиусе окна оптического потока r существуют все пикселы окрестности $W_1^r(x+u,y+v)$, т. е. для пикселов $p_2(x,y)\in P$, где $P=\{p_2(x,y)\,|\,R+r\le x<IW-R-r,R+r\le y<IH-R-r\}$ (рис. 1).

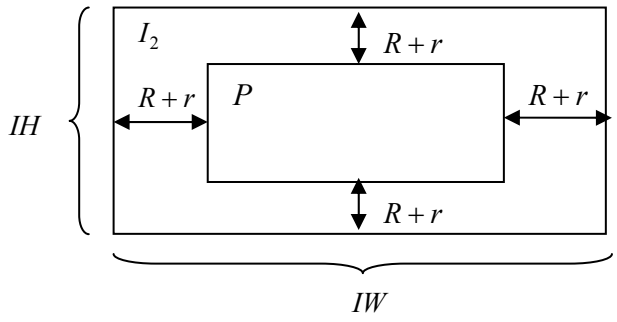


Рис. 1. Множество пикселов P изображения I_2, для которых можно вычислить оптический поток

Будем вычислять оптический поток только для множества пикселов P, так как для любого пиксела $p_2(x,y)\notin P$ можно найти сдвиг $(u',v')\in S$, при котором часть окрестности $W_1^r(x+u',y+v')$ будет выходить за пределы изображения.

Среди алгоритмов вычисления оптического потока на основе поиска наилучшего соответствия областей можно выделить алгоритмы, основанные на применении бегущего окна, сложность которых не зависит от размера окна оптического потока [1, 13, 23, 24]. Описание

эффективного вычисления оконных сумм на основе бегущего окна для изображений дано в [39]. Пусть I – изображение, для которого необходимо вычислить локальные суммы элементов по окну размера $(2r+1)\times(2r+1)$:

$$SUM(x,y)=\sum_{j=y-r}^{y+r}\sum_{i=x-r}^{x+r} I(i,j). \quad (1)$$

Обозначим через $SUMV(x,y)$ сумму элементов окна $1\times(2r+1)$, образующих вертикальный столбец:

$$SUMV(x,y)=\sum_{j=y-r}^{y+r} I(x,j).$$

Перепишем формулу (1) следующим образом:

$$SUM(x,y)=\sum_{i=x-r}^{x+r} SUMV(i,y).$$

Заметим, что сумма $SUMV(x,y)$ может быть вычислена на основе суммы $SUMV(x,y-1)$:

$$SUMV(x,y)=SUMV(x,y-1)-I(x,y-r-1)+I(x,y+r), \quad (2)$$

а сумма $SUM(x,y)$ – на основе суммы $SUM(x-1,y)$:

$$SUM(x,y)=SUM(x-1,y)-SUMV(x-r-1,y)+SUMV(x+r,y). \quad (3)$$

Тогда локальные суммы элементов по окну можно находить при помощи алгоритма с константной сложностью, вычисляя в несколько этапов:

– начальные суммы $SUMV(x,r+1)$, $x=\overline{1,IW}$;

– все суммы $SUMV(x,y)$, $x=\overline{1,IW}$, $y=\overline{r+2,(IH-r)}$, при помощи бегущего окна (см. формулу (2));

– начальные суммы $SUM(r+1,y)$, $y=\overline{r+1,(IH-r)}$;

– все суммы $SUM(x,y)$, $x=\overline{r+2,(IW-r)}$, $y=\overline{r+2,(IH-r)}$ при помощи бегущего окна (см. формулу (3)).

Вычисления, описанные выше, можно производить построчно, определяя $SUMV(x,y)$ на основе формулы (2) не для всего изображения сразу, а только для одной строки, после чего можно вычислять $SUM(x,y)$ на основе формулы (3) только для этой строки. После этого алгоритм переходит к следующей строке и выполняет для нее вычисления таким же образом. Построчное выполнение алгоритма требует меньшего объема памяти для хранения промежуточных вычислений, таких как бегущие вертикальные суммы $SUMV(x,y)$.

С помощью описанного алгоритма бегущего окна можно вычислять оценки $SAD_{(x,y)}(u,v)$ за константное время в зависимости от размера окна оптического потока. К алгоритмам вычисления оптического потока, использующим указанный подход, можно отнести алгоритмы Фогераса и соавт. [1], алгоритм Канаде и соавт. [13], Сана [14–19, 23, 24]. Основным отличием алгоритма Сана от похожих алгоритмов Фогераса и соавт. [1] и Канады и соавт. [13] является реализация основного цикла алгоритма по сдвигам, а не по пикселам, т. е. в алгоритмах Фогераса и соавт., а также Канады и соавт. для каждого пиксела изображения сразу вычисляются все оценки для всевозможных сдвигов окрестности пиксела, после чего вычисления продолжаются для следующих пикселов. В алгоритме Сана первоначально фиксируется сдвиг, для которого вычисляются оценки для окрестностей всех пикселов при дан-

ном сдвиге, после чего осуществляется переход к следующему сдвигу. Подобные небольшие отличия в организации циклов вычисления оценок для различных сдвигов окрестностей пикселов приводят к различной программной реализации алгоритмов, особенно при использовании SSE2-инструкций.

3. Реализация алгоритма вычисления оптического потока при помощи SSE2

SSE2 – дополнительный набор инструкций, позволяющий ускорить обработку мультимедиаданных (звука, видеопотоков, изображений). В настоящее время он поддерживается практически всеми процессорами как фирмы Intel, так и AMD. SSE2 включает в себя набор 128-битных регистров, а также дополнительные команды, которые позволяют выполнять некоторые арифметические операции параллельно. Например, за одну операцию можно параллельно складывать, вычитать либо выполнять другие операции над 16 байтами, 8 словами либо 4 двойными словами.

Одним из методов вычисления оптического потока на основе SSE2-команд является применение алгоритма сопоставления блоков, используемого для кодирования видеопоследовательностей (см. табл. 4, алгоритм 1) [9]. Сопоставление блоков применяется для окрестности каждого пиксела изображения, что приводит к значительно большим вычислительным затратам по сравнению с исходным алгоритмом, где сопоставление производится только для достаточно крупных блоков изображения (4×4, 8×8, 16×16). Очевидно, что время работы подобного алгоритма неудовлетворительно для вычислений в режиме реального времени. В процессорах Intel Pentium 4 появились новые SSE3-команды, в частности LDDQU, которые позволяют ускорить вычисления в алгоритме сопоставления блоков [8], но даже при их использовании скорость вычисления оптического потока не достигает реального времени.

Ускорить вычисления в указанных выше алгоритмах можно при помощи организации вычислений на основе бегущего окна. Однако специфика данных SSE2-алгоритмов позволяет получить на их основе только алгоритм с линейной сложностью вычислений в зависимости от размера окна оптического потока (см. табл. 4, алгоритм 2). Вычисления векторов смещения производятся по столбцам сверху вниз, при этом результаты оценок для всех смещений данного пиксела сохраняются в массиве и используются для вычисления оценок следующего пиксела в столбце.

Указанные выше алгоритмы хорошо работают с окнами оптического потока размера 4×4, 8×8, 16×16, однако их сложно адаптировать для работы с окнами произвольного размера.

Высокую скорость вычислений оптического потока можно достичь, используя алгоритмы бегущего окна с константной сложностью. Реализация алгоритма Фогераса и соавт. для стереовосстановления при помощи MMX-инструкций была предложена в статье [22]. Предлагаемый в данной статье алгоритм отличается от указанных выше тем, что он основан на алгоритмах Сана [14–19, 23, 24] и реализуется при помощи MMX- и SSE-команд на базе метода, отличного от SIMD-реализации алгоритма Фогераса и соавт.

Реализация алгоритмов Сана [14–19, 23, 24] вычисления оптического потока при помощи MMX- и SSE-команд описана в статьях [5, 6], однако применение SIMD-команд в указанных статьях ограничивается только наиболее трудоемкой частью алгоритма, в которой производится вычисление корреляции на основе оконных сумм, полученных без применения MMX- и SSE-команд. Авторы статей [5, 6] описывают реализацию вычисления оптического потока при помощи корреляции, однако применение SAD для вычисления оптического потока дает преимущество в более быстром вычислении оценок схожести блоков изображений (модуль разности вычисляется при помощи SSE2 значительно проще, чем корреляция), а также позволяет обрабатывать сразу по 16 элементов одновременно при вычислении модуля разности и по 8 элементов при суммировании, что ускоряет выполнение алгоритма за счет более высокой степени параллелизма и не требует использования вещественных чисел. Также при применении SAD, в отличие от корреляции, не нужно вычислять квадратные корни, что является достаточно трудоемкой операцией. В данной статье предлагается эффективная реализация алгоритма Сана на основе SSE2-команд, отличная от упомянутой выше.

Алгоритм Сана условно состоит из двух частей:

1) вычисление куба оценок для всех смещений каждого пиксела второго изображения относительно первого;

2) вычисление при помощи динамического программирования оптического потока на основе поиска максимальной поверхности в кубе.

Для получения быстрого алгоритма вычисления оптического потока будет использоваться алгоритм Сана со следующими модификациями:

1. В качестве меры отличия областей изображений используется SAD.

2. Оценки, вычисленные для всех пикселов при некотором фиксированном сдвиге, не сохраняются в массиве (кубе) всех оценок, а оцениваются сразу, при этом сохраняются только наилучшие оценки для каждого пиксела.

3. Реализация всех этапов работы алгоритма основана на применении SSE2-инструкций.

Зададим максимальный размер окна оптического потока 16×16, для того чтобы любая возможная оценка SAD для окрестности некоторого пиксела помещалась в двухбайтовую ячейку памяти. Пусть $S=\{(u,v)\,|\,-R\le u,v\le R\}$ – множество возможных сдвигов. Пронумеруем все сдвиги из множества S и пусть $S_i=(u',v')$ – сдвиг с порядковым номером i, где $1\le i\le(R+1)^2$. Пусть M – двумерный массив оценок SAD и $M(x,y)$ – наилучшая полученная на данный момент оценка SAD для пиксела с координатами (x,y) изображения I_2. Пусть MU, MV – двумерные массивы, содержащие сдвиги по осям X и Y, при которых получены наилучшие на данный момент оценки SAD, хранящиеся в массиве M: $MU(x,y)=\bar{u}$, $MV(x,y)=\bar{v}$, где $(\bar{u},\bar{v})$ – сдвиг, при котором получена оценка $M(x,y)$.

Получим следующий алгоритм вычисления оптического потока:

1. Пусть $i=1$, $M(x,y)=MAX$, где $0\le x<IW$, $0\le y<IH$, а MAX – некоторая константа, заведомо превышающая любую возможную оценку SAD для данного размера окна оптического потока.

2. Зафиксируем сдвиг S_i.

3. Вычислим модуль разности двух изображений I_1 и I_2 с учетом сдвига S_i и сохраним в двумерный массив D:

$$D(x,y)=\left|I_1(x+u',y+v')-I_2(x,y)\right|,$$

где $(x,y)\in PD$, $PD=\{(x,y)\,|\,R\le x<IW-R, R\le y<IH-R\}$ – множество координат пикселов изображения I_2, для которых можно вычислить модуль разности с пикселами изображения I_1 при любом сдвиге из множества S.

4. При помощи вертикального бегущего окна размера $1\times(2r+1)$, где 1 – ширина окна, а $(2r+1)$ – высота окна, вычислим вертикальные суммы модулей разности по столбцам массива D и сохраним в двумерный массив V:

$$V(x,y)=\sum_{k=-r}^{r}D(x,y+k),$$

где $(x,y)\in PV$, $PV=\{(x,y)\,|\,R\le x<IW-R, R+r\le y<IH-R-r\}$ – множество координат элементов массива D, для которых можно вычислить вертикальные суммы по столбцам при любом сдвиге из множества S и данном радиусе окна оптического потока r.

5. При помощи горизонтального бегущего окна размера $(2r+1)\times1$, где $(2r+1)$ – ширина окна, а 1 – высота окна, вычислим горизонтальные суммы по строкам массива V и сохраним в двумерный массив H:

$$H(x,y)=\sum_{k=-r}^{r}V(x+k,y),$$

где $(x,y) \in PH$, $PH = \{(x,y) \mid R + r \le x < IW - R - r, R + r \le y < IH - R - r\}$ – множество координат элементов массива V, для которых можно вычислить горизонтальные суммы по строкам при любом сдвиге из множества S и данном радиусе окна оптического потока r.

6. Сравним полученные оценки SAD, сохраненные в массиве H, с оценками, хранящимися в массиве M: если $H(x,y) < M(x,y)$, то $M(x,y) = H(x,y)$, $MU(x,y) = u'$, $MV(x,y) = v'$.

7. $i = i + 1$; если $i \le (R+1)^2$, идем на шаг 2.

8. Конец.

Таким образом, после выполнения алгоритма массивы MU, MV будут содержать векторы движения пикселов изображения I_2, а массив H – оценки SAD для соответствующих пикселов.

Представленный выше алгоритм использует SSE2-команды на всех этапах вычисления оптического потока, в частности, для получения:

– модуля разности изображений со сдвигом;

– вертикальных бегущих сумм (как вертикальных, так и горизонтальных);

– горизонтальных бегущих сумм.

Опишем подробно реализацию каждого шага алгоритма при помощи SSE2-инструкций. Так как в процессе вычисления оптического потока бегущая сумма вычисляется много раз – столько же, сколько рассматривается сдвигов, то скорость ее вычисления, как и скорость вычисления модуля разности изображений, должна быть как можно выше. Процесс вычисления бегущей суммы состоит из двух этапов. Первоначально вычисляется вертикальная бегущая сумма, затем – горизонтальная. Если максимальное окно оптического потока не превосходит 16×16, то вертикальное и горизонтальное суммирование кадров, представляющих собой модуль разности двух исходных кадров со сдвигом, можно выполнять в элементах типа WORD, так как максимальная сумма элементов блока размера 16×16 не превосходит $255 \times 16 \times 16$, что помещается в ячейку памяти WORD. В таком случае при вычислении вертикальной и горизонтальной бегущих сумм при помощи SSE2-команд можно добиться параллельного вычисления восьми бегущих сумм.

Вычисление модуля разности двух кадров со сдвигом при помощи SSE2-команд. Для вычисления оптического потока необходимо определить, столько модулей разностей изображений со сдвигом и сколько сдвигов рассматривается при вычислении оптического потока. Таким образом, при интервале возможных сдвигов ± 16 по осям X и Y необходимо вычислить 1089 модулей разности исходных кадров со сдвигом. При размере изображений, например, 720×576 функция вычисления модуля разности и модуля разности со сдвигом должна быть хорошо оптимизирована, чтобы занимать как можно меньше времени. Чтобы уложиться в режим реального времени (40 мс на обработку одного кадра) при вычислении оптического потока, вычисления одного модуля разности изображений должно занимать время, в несколько раз меньшее, чем 40/1089 = 0,036 мс.

Существует несколько способов реализации эффективного вычисления модуля разности при помощи SSE2-команд:

1. Применение арифметики с насыщением [40]. Пусть в регистрах R1 и R2 находятся 16 байтовых элементов двух изображений, тогда модуль разности этих 16 элементов определяется следующим образом:

R3 = _mm_subs_epu8 (R1,R2);

R4 = _mm_subs_epu8 (R2,R1);

R4 = _mm_or_si128 (R4,R3).

2. Применение команд поиска минимума и максимума. Для тех же регистров R1 и R2 модуль разности 16 байтовых элементов можно искать таким образом:

R3 = _mm_min_epu8(R1,R2);

R4 = _mm_max_epu8(R1,R2);

R4 = _mm_subs_epu8(R4,R3).

Модуль разности изображений вычисляется со сдвигом второго изображения относительно первого. Для чтения данных со сдвигом из изображения используется SSE2-команда MOVDQU (интринсик _mm_loadu_si128), которая позволяет загрузить из памяти в SSE2-регистр сразу

16 байт. При поддержке процессором SSE3-команд вместо MOVDQU-команды возможно использование LDDQU-команды (интринсик _mm_lddqu_si128), которая позволяет быстрее читать данные по адресам в памяти, не выровненным на границу в 16 байт. Если организовать вычисление модуля разности специальным образом (использовать изображения с шириной, кратной 16, и максимальный сдвиг, кратный 16), то данные со сдвигом из изображения могут читаться MOVDQA-командой для выровненных данных (интринсик _mm_load_si128), что также увеличит быстродействие.

Сравнительное быстродействие вычислений модуля разности двух кадров со сдвигом как при помощи SSE2-команд, так и без них приведено в табл. 1.

Таблица 1

Быстродействие операций поиска модуля разности кадров

Операция	Размер изображения	Время, мс
Вычисление модуля разности без SSE2-команд	320×240	0,077
	640×480	0,340
	720×576	0,490
Вычисление модуля разности с SSE2-командами	320×240	0,006
	640×480	0,039
	720×576	0,053

Вычисление вертикальных бегущих сумм по столбцам изображения при помощи SSE2-команд. Будем учитывать тот факт, что результатом вычисления модуля разности двух изображений со сдвигом является байтовый массив, который будем называть изображением модуля разности кадров. При этом вертикальные бегущие суммы по столбцам могут не помещаться в байтовое значение, поэтому хранить их необходимо в двухбайтовых ячейках памяти (при высоте окна оптического потока не более 256 пикселов).

Рассмотрим первые 16 столбцов изображения модуля разности кадров: $S_1, \ldots, S_{16}$. Пусть $S_n(i)$ – i-й элемент столбца с номером n сверху, а $H = 2 \times r + 1$ – высота вертикального окна, в котором необходимо вычислить сумму элементов столбца, равную высоте окна оптического потока, тогда $SUM_n(i)$ – сумма элементов i-го окна бегущей суммы:

$$SUM_n(i) = \sum_{k=i}^{i+H-1} S_n(k).$$

Каждая сумма $SUM_n(i+1)$ за исключением первой суммы $SUM_n(1)$ вычисляется на основе предыдущей суммы по формуле

$$SUM_n(i+1) = SUM_n(i) - S_n(i) + S_n(i+H).$$

Тогда вычисление вертикальных бегущих сумм для рассматриваемых 16 столбцов $S_1, \ldots, S_{16}$ при помощи SSE2-инструкций осуществляется следующим образом:

1. $i = 1$, $SUM_1 = 0$, $SUM_2 = 0$.

2. Копируем в SSE2-регистр REG_1 16 i-х элементов рассматриваемых 16 столбцов:

$$REG_1 = S_1(i), \ldots, S_{16}(i).$$

3. Преобразуем 16 байтовых элементов, находящихся в регистре REG_1, в 16 двухбайтовых элементов и помещаем первые 8 в регистр REG_2 и вторые 8 – в регистр REG_3.

4. Аккумулируем полученные элементы в двух регистрах SUM_1 и SUM_2:

$$SUM_1(i) = SUM_1(i) + REG_2(i);$$
$$SUM_2(i) = SUM_2(i) + REG_3(i).$$

5. $i = i + 1$; если $i \le H$, идем на шаг 2.

6. Сохраняем 16 полученных вертикальных сумм из регистров SUM_1 и SUM_2 в массив V.

7. Копируем в регистр REG_1 следующие 16 i-х элементов:

$$REG_1 = S_1(i),...,S_{16}(i).$$

8. Преобразуем 16 байтовых элементов, находящихся в регистре REG_1, в 16 двухбайтовых элементов и помещаем первые 8 в регистр REG_2 и вторые 8 – в регистр REG_3.

9. Копируем в регистр REG_4 16 элементов с индексом $(i - H)$:

$$REG_4 = S_1(i - H),...,S_{16}(i - H).$$

10. Преобразуем 16 байтовых элементов, находящихся в регистре REG_4, в 16 двухбайтовых элементов и помещаем первые 8 в регистр REG_5 и вторые 8 – в регистр REG_6.

11. Вычисляем новые суммы на основе имеющихся 16 сумм при помощи бегущего окна:

$$SUM_1(i) = SUM_1(i) - REG_5(i) + REG_2(i);$$
$$SUM_2(i) = SUM_2(i) - REG_6(i) + REG_3(i).$$

12. Сохраняем 16 полученных вертикальных сумм из регистров SUM_1 и SUM_2 в массив V.

13. $i = i + 1$; если $i \le IH$, то идем на шаг 7.

14. Конец.

Шаги 1–6 алгоритма выполняют вычисление первых 16 сумм, на основе которых при помощи бегущего окна вычисляются все остальные суммы (шаги 7–14). Вычисление вертикальных сумм для остальных столбцов изображения выполняется аналогичным образом. Сравнительное быстродействие вычисления вертикальных бегущих сумм как при помощи SSE2-команд, так и без них отражено в табл. 2.

Таблица 2

Быстродействие операции поиска вертикальных бегущих сумм

Операция	Размер изображения	Время, мс
Вычисление вертикальных бегущих сумм без SSE2-команд	320×240	0,286
	640×480	1,549
	720×576	2,117
Вычисление вертикальных бегущих сумм с SSE2-командами	320×240	0,010
	640×480	0,058
	720×576	0,080

Вычисление горизонтальных бегущих сумм по строкам изображения при помощи SSE2-команд. Данный шаг алгоритма сложно реализовать при помощи SSE2-инструкций, применяя метод, аналогичный вычислению вертикальных бегущих сумм, так как SSE2-команды чтения элементов массива работают только с горизонтально расположенными данными. Для реализации алгоритма горизонтальной бегущей суммы по аналогии с вертикальной бегущей

суммой необходимо было бы выполнять чтение сразу восьми элементов массива, расположенных вертикально, что невозможно осуществить при помощи одной SSE2-команды. Для решения данной задачи можно предложить несколько подходов:

1. Транспонирование массива и применение алгоритма вертикальной бегущей суммы.

2. Чтение из массива прямоугольной части размера 8×8 при помощи восьми SSE2-команд чтения и транспонирование полученного квадрата с помощью SSE2-команд. Последующие вычисления производятся со строками транспонированного квадрата, представляющими собой транспонированные столбцы по восемь элементов исходного массива.

3. Первоначальное преобразование изображений, для которых будет вычисляться оптический поток таким образом, чтобы можно было осуществить параллельные вычисления восьми бегущих горизонтальных сумм одновременно. После вычисления оптического потока на преобразованных изображениях необходимо обратное преобразование полученных данных, которое даст правильный массив векторов движения.

Первый вариант требует транспонирования всех изображений модуля разности двух кадров, полученных для всех рассматриваемых сдвигов. Например, для интервала сдвигов ± 16 по осям X и Y это потребует выполнения 1089 транспонирований изображений (общее количество операций чтения и записи элементов будет равно $1089\times IW\times IH\times 2$), что занимает значительное время и не позволяет реализовать вычисления достаточно быстро.

Второй вариант фактически повторяет первый, только с транспонированием при помощи SSE2-команд. При проходе по всему изображению с вычислением горизонтальных бегущих сумм необходимо будет транспонировать практически все квадратики 8×8, на которые разбивается изображение. При этом данная процедура повторяется заново при каждом новом сдвиге, т. е., например, для интервала сдвигов ± 16 по осям X и Y она повторяется 1089 раз (транспонирование квадрата 8×8 требует 24 SSE2-инструкции, тогда общее количество операций составляет $1089\times IW\times IH\times(24/64)$). Таким образом, при данном подходе общее количество операций уменьшается благодаря эффективной процедуре транспонирования, реализованной при помощи SSE2-команд, однако число излишних вычислений остается большим.

Третий вариант предусматривает смешение восьми частей изображения таким образом, что в каждой строке полученного изображения будут содержаться данные восьми частей изображения, при этом вычисление горизонтальных бегущих сумм можно выполнять параллельно, используя только SSE2-команды чтения горизонтально расположенных данных в рассматриваемой строке. Этот подход содержит дополнительные этапы предварительного смешения частей изображения и этап постобработки по восстановлению изображения из смешанных частей. Данные этапы выполняются только по одному разу и не занимают много вычислительного времени (общее количество операций чтения и записи элементов будет равно $IW\times IH\times 2$). При этом остальные этапы вычисления оптического потока (вычисление модуля разности и вычисление вертикальных бегущих сумм), как и раньше, выполняются в параллельном режиме по описанным выше алгоритмам, лишь с небольшими отличиями. При использовании данного варианта вычисления горизонтальных бегущих сумм количество операций хотя и увеличивается за счет общего увеличения размера обрабатываемого изображения, однако остается наименьшим из описанных трех вариантов.

Для вычисления горизонтальной бегущей суммы при помощи SSE2 будем использовать метод предварительного разбиения изображения на восемь частей и их перемешивания. Тогда вычислять горизонтальную сумму можно таким же образом, как в реализации горизонтальной суммы без SSE, только читать и обрабатывать необходимо сразу восемь WORD-элементов. Каждый из этих восьми элементов будет относиться к разным частям изображения и, таким образом, можно будет вычислять параллельно восемь бегущих сумм для восьми разных частей изображения.

Для вычисления восьми горизонтальных бегущих сумм необходимо разбить исходные изображения на восемь частей. При этом, так как бегущая сумма вычисляется только для множества пикселов P, необходимо разбивать именно эту область на части. Необходимо также учесть, что для вычисления крайних пикселов каждой части этой области необходимо иметь данные о пикселах, выходящих за пределы этой части.

Разобьем область P изображения I_2 на восемь одинаковых частей $P_0,...,P_7$ (рис. 2). Обозначим через WP_i область P_i, расширенную на $R+r$ в каждую сторону (рис. 3).

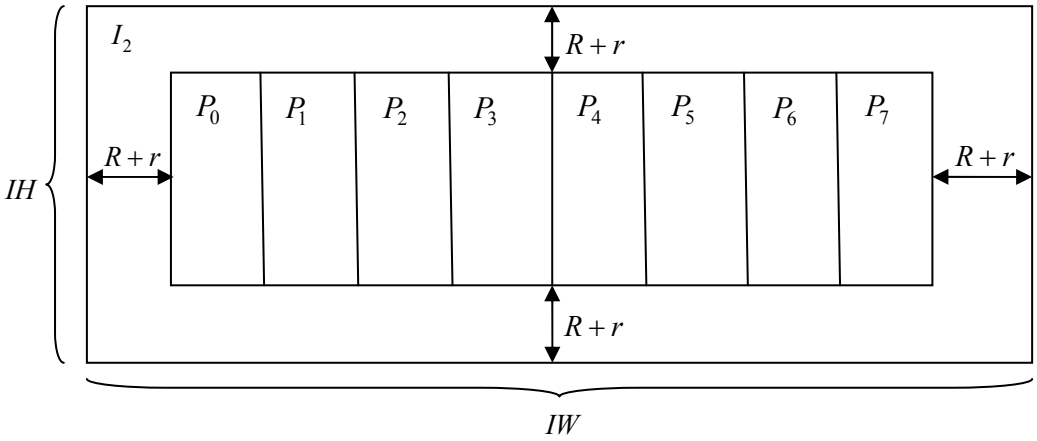


Рис. 2. Разбиение множества пикселов P изображения I_2 на части

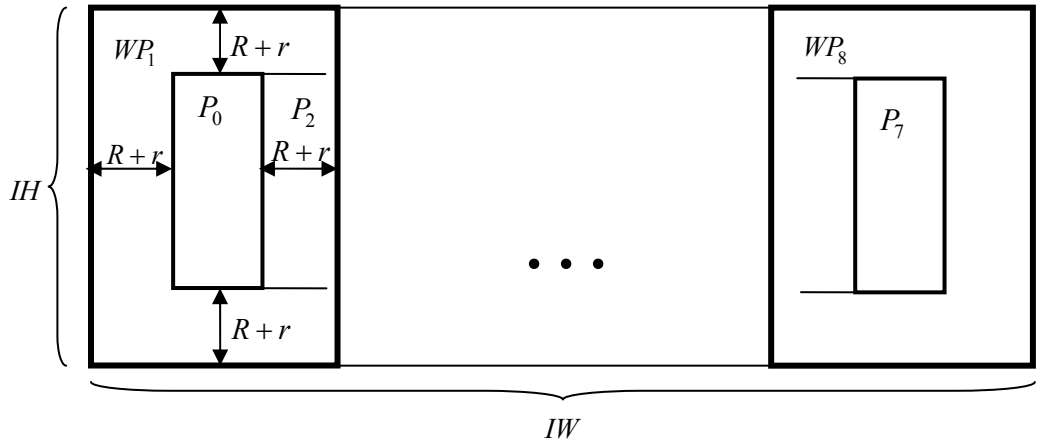


Рис. 3. Расширенные области P_i изображения I_2

Таким образом, получим восемь пересекающихся областей $WP_i, i = 0,...,7$. Если I_2 имеет размеры $IW \times IH$, область изображения P будет иметь размер $PW \times PH = (IW - 2\times(R+r)) \times (IH - 2\times(R+r))$, а каждая область P_i будет иметь размер $(PW/8) \times PH$. Следовательно, каждая расширенная область изображения WP_i будет иметь размер $(\frac{PW}{8} + 2\times(R+r)) \times (PH + 2\times(R+r))$. Обозначим через $NC = \frac{PW}{8} + 2\times(R+r)$ количество столбцов в расширенной области WP_i, $C_i(k)$ – k-й столбец расширенной области WP_i (нумерация столбцов начинается с нуля). Сформируем из расширенных областей WP_i изображения I_2 новое изображение I_2' следующим образом. Будем располагать последовательно столбцы расширенных областей WP_i: сначала все первые столбцы, потом вторые и т. д. Пусть S_i – i-й столбец изображения I_2' (нумерация столбцов начинается с нуля), тогда

$$S_i = C_{i-8\times\left[\frac{i}{8}\right]}\left(\left[\frac{i}{8}\right]\right).$$

Столбцы S_i образуют изображение I_2', размер которого составит $(8\times NC)\times IH$. Таким образом, изображение I_2' имеет ширину $IW_2' = 8\times NC = IW + 7\times 2\times(R+r)$, что больше, чем у изображения I_2, на $2\times(R+r)\times 7$ столбцов. Например, при размере кадра видеопоследовательности 720×576, радиусе окна оптического потока $r = 8$ и максимально возможном смещении пиксела

$R = 16$ ширина преобразованного кадра составит $720 + 7 \times 2 \times (8 + 16) = 1056$, а общий размер преобразованного кадра – 1056×576. Описанное выше преобразование необходимо производить как для изображения I_2, так и для изображения I_1. Однако можно заметить, что при обработке видеопоследовательностей достаточно выполнять одно преобразование изображения для каждого нового кадра видеопоследовательности, так как второй кадр, необходимый для вычисления оптического потока, уже будет преобразован на предыдущем шаге. Так как соседние столбцы исходных изображений I_1 и I_2 лежат в преобразованных изображениях I_1' и I_2' с шагом в восемь столбцов, становится возможным читать при помощи одной SSE2-операции чтения данных элементы сразу восьми различных столбцов изображения и оперировать при вычислении горизонтальной бегущей суммы сразу восемью элементами одновременно. Это позволяет вычислять бегущие горизонтальные суммы аналогично вычислению вертикальных бегущих сумм одновременно для восьми различных частей исходных изображений I_1 и I_2. Также заметим, что количество операций, необходимых для вычисления оптического потока на полученном изображении, возрастает непропорционально увеличению площади изображения. Для полученного изображения 1056×576 по сравнению с исходным изображением 720×576 количество операций возрастет в 1,16 раза, тогда как площадь первого больше, чем площадь второго, в 1,46 раза.

Сравнительное быстродействие вычислений горизонтальных бегущих сумм как при помощи SSE2-команд, так и без них приведено в табл. 3.

Таблица 3

Быстродействие операций поиска горизонтальных бегущих сумм

Операция	Размер изображения	Время, мс
Вычисление горизонтальных бегущих сумм без SSE2-команд	320×240	0,158
	640×480	0,730
	720×576	0,956
Вычисление горизонтальных бегущих сумм с SSE2-командами	320×240	0,017
	640×480	0,071
	720×576	0,099

Оптический поток для полученных после преобразования изображений I_1' и I_2' вычисляется при помощи описанного выше алгоритма с небольшими отличиями:

– при вычислении модуля разности двух кадров со сдвигом каждый сдвиг по оси X необходимо увеличивать в восемь раз;

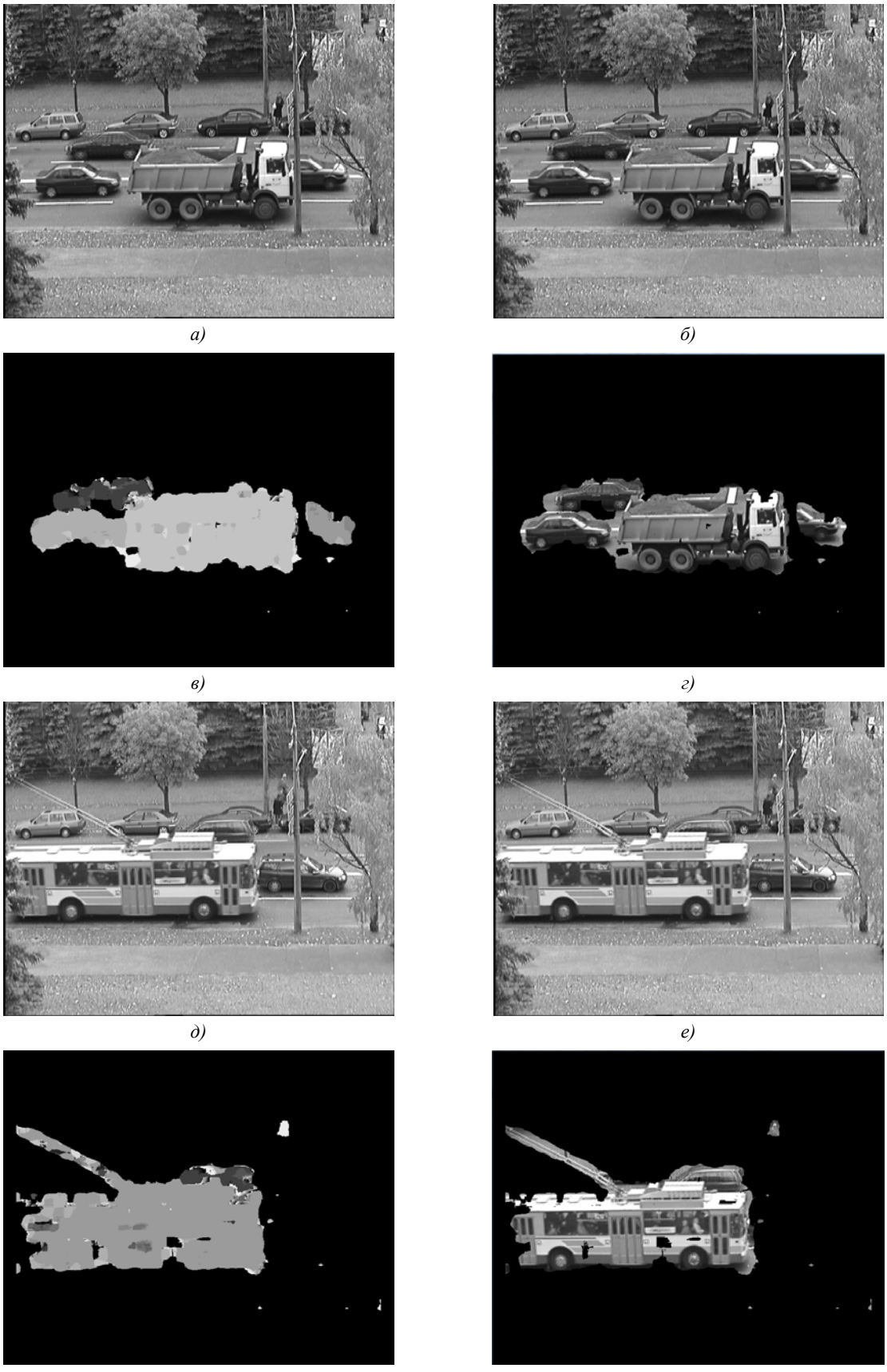
– при вычислении горизонтальной бегущей суммы можно использовать алгоритм, аналогичный вычислению вертикальных бегущих сумм на основе SSE2, что позволяет вычислять параллельно по восемь бегущих сумм;

– после вычисления оптического потока на преобразованных изображениях необходимо выполнить обратное преобразование оптического потока.

Необходимо заметить, что потери производительности за счет увеличения размеров изображений по отношению к исходным меньше, чем выигрыш, получаемый за счет параллельной обработки при вычислении горизонтальных бегущих сумм.

4. Результаты тестирования алгоритма быстрого вычисления оптического потока при помощи SSE2-команд

Работа алгоритма тестировалась на примере реальной видеопоследовательности, содержащей изображения различных движущихся объектов (транспорта, людей) при разрешении кадров 320×240, 640×480, 720×576 (рис. 4) на персональном компьютере с процессором Intel Core i5 750. Кадры видеопоследовательности при более низких разрешениях были получены масштабированием кадров исходной видеопоследовательности с разрешением 720×576.



а) *б)*

в) *г)*

д) *е)*

ж) *з)*

Рис. 4. Результаты работы алгоритма вычисления оптического потока: *а)*, *д)* первый кадр пары; *б)*, *е)* второй кадр пары; *в)*, *ж)* вычисленный оптический поток; *г)*, *з)* обнаруженные движущиеся объекты

Результаты быстродействия алгоритма на персональном компьютере с процессором Intel Core i5 750 с использованием одного ядра приведены в табл. 4. Можно заметить, что при разрешении 320×240, окне оптического потока 16×16 и максимальных возможных сдвигах ±8 алгоритм практически достигает быстродействия реального времени (15 кадров/с).

Таблица 4

Быстродействие алгоритмов вычисления оптического потока с окном 16×16

Алгоритмы оптического потока	Размер кадра видеопоследовательности	Интервал сдвигов	Время работы на кадр, мс
Алгоритм 1 оценки движения при помощи SSE2-команд	320×240	±8	291
	320×240	±16	1034
	640×480	±16	5028
	720×576	±16	6424
Алгоритм 2 оценки движения с линейной сложностью при помощи SSE2-команд	320×240	±8	102
	320×240	±16	332
	640×480	±16	1604
	720×576	±16	2213
Алгоритм Сана вычисления оптического потока без помощи SSE2-команд	320×240	±8	168
	320×240	±16	636
	640×480	±16	3096
	720×576	±16	4324
Быстрый алгоритм вычисления оптического потока при помощи SSE2-команд (горизонтальная бегущая сумма без SSE2-команд)	320×240	±8	78
	320×240	±16	289
	640×480	±16	1356
	720×576	±16	1860
Быстрый алгоритм вычисления оптического потока при помощи SSE2-команд (преобразование изображений)	320×240	±8	**49**
	320×240	±16	**176**
	640×480	±16	**768**
	720×576	±16	**1050**

Для реализации параллельной версии алгоритма была использована библиотека OpenMP. Результаты быстродействия параллельной версии алгоритма на персональном компьютере с процессором Intel Core i5 750 с использованием четырех ядер приведены в табл. 5.

Таблица 5

Быстродействие параллельной версии алгоритма вычисления оптического потока с окном 16×16 (четыре ядра процессора)

Алгоритм оптического потока	Размер кадра видеопоследовательности	Интервал сдвигов	Время работы на кадр, мс
Параллельная версия быстрого алгоритма вычисления оптического потока при помощи SSE2-команд (преобразование изображений)	320×240	±8	**25**
	320×240	±16	**85**
	640×480	±16	**252**
	720×576	±16	**328**

Заключение

Предложен алгоритм быстрого вычисления оптического потока при помощи SSE2 SIMD-инструкций процессоров семейства x86. Алгоритм имеет константную сложность (O(1)) в зависимости от радиуса окна оптического потока и применяет SSE2 SIMD-команды на всех этапах вычисления оптического потока (в том числе для вычисления горизонтальных и вертикальных бегущих сумм), что значительно ускоряет его работу. Применение SAD позволяет при помощи SSE2-команд одновременно обрабатывать большее количество пикселов изображения по сравне-

нию с корреляцией и SSD, а также быстрее вычислять меру отличия блоков изображений вследствие более простой формулы. При работе на многоядерных процессорах алгоритм использует параллельный режим работы, что позволяет ускорить вычисления. Скорость работы алгоритма на процессоре Intel Core i5 750 в режиме параллельного выполнения на четырех ядрах при разрешении видеопоследовательности 320×240, окне оптического потока 16×16 и максимальных возможных сдвигах ± 8 по осям X и Y (всего вычисляется $17 \times 17 = 289$ сдвигов) достигает 40 кадров в секунду, а при максимальных возможных сдвигах ± 16 (всего вычисляется $33 \times 33 = 1089$ сдвигов) – 11 кадров в секунду. Это позволяет применять алгоритм для вычисления оптического потока в режиме реального времени на персональном компьютере. Алгоритм может быть полезен для решения таких задач, как обнаружение и отслеживание движущихся объектов на видеопоследовательностях.

Список литературы

1. Faugeras, O. Real time correlation-based stereo: Algorithm, implementations and applications / O. Faugeras, B. Hotz, H. Mathieu // Technical Report RR-2013. – France : INRIA, 1993.

2. Yokoyama, M. A Contour-Based Moving Object Detection and Tracking / M. Yokoyama, T. Poggio // 2[d] Joint IEEE Intern. Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance. – Breckenridge, Colorado, USA, 2005. – P. 271–276.

3. Denman, S. An Adaptive Optical Flow Technique for Person Tracking Systems / S. Denman, V. Chandran, S. Sridharan // Pattern Recognition Letters. – 2007. – Vol. 28, № 10. – P. 1232–1239.

4. Kang, S. Estimation of Moving Information for Tracking of Moving Objects / S. Kang, J. Park, S. Jeong // Journal of Mechanical Science and Technology. – 2001. – Vol. 15, № 3. – P. 300–308.

5. Sadykhov, R. Fast cross correlation algorithm for optical flow estimation / R. Sadykhov, D. Lamovsky // Proc. of Nordic Signal Processing Symposium (NORSIG'2006). – Reykjavik, Iceland, 2006. – P. 322–325.

6. Sadykhov, R. Estimation of the cross correlation based optical flow for video surveillance / R. Sadykhov, D. Lamovsky // Computing. – 2006. – Vol. 5, № 3. – P. 112–117.

7. Tian, Y. Robust Salient Motion Detection with Complex Background for Real-time Video Surveillance / Y. Tian, A. Hampapur // IEEE Computer Society Workshop on Motion and Video Computing. – Breckenridge, Colorado, USA, 2005. – Vol. 2. – P. 30–35.

8. Intel Corp. Block-Matching in Motion Estimation Algorithms Using Streaming SIMD Extensions 3. – USA : Intel Applications Notes, 2003.

9. Intel Corp. Motion Estimation with Intel® Streaming SIMD Extensions 4 (Intel® SSE4) [Electronic resource]. – Mode of access : http://software.intel.com/en-us/articles/motion-estimation-with-intel-streaming-simd-extensions-4-intel-sse4/. – Date of access : 09.09.11.

10. Tran, T. Performance Enhancement of Motion Estimation Using SSE2 Technology / T. Tran, H. Cho, S. Cho // World Academy of Science, Engineering and Technology. – 2008. – Vol. 40. – P. 168–171.

11. Chen, Z. Efficient Block Matching Algorithm for Motion Estimation / Z. Chen // Intern. Journal of Signal Processing. – 2009. –Vol. 5, № 2. – P. 133–137.

12. Chen, Y. Fast block matching algorithm based on the winner-update strategy / Y. Chen, Y. Hung, C. Fuh // IEEE Transactions on Image Processing. – 2001. – Vol. 10, № 8. – P. 1212–1222.

13. Kanade, T. Development of a video-rate stereo machine / T. Kanade, H. Kato, S. Kimura // In Proc. of Intern. Robotics and Systems Conference (IROS'95). – Pittsburgh, Pennsylvania, USA, 1995. – Vol. 3. – P. 95–100.

14. Sun, A. Fast Stereo Matching Method / A. Sun // Proc. of Digital Image Computing: Techniques and Applications. – Auckland, New Zealand, 1997. – P. 95–100.

15. Sun, C. Multi-Resolution Rectangular Subregioning Stereo Matching Using Fast Correlation and Dynamic Programming Techniques / C. Sun // CMIS Report 98/246. – Australia : CSIRO, 1998.

16. Sun, C. Multi-Resolution Stereo Matching Using Maximum-Surface Techniques / C. Sun // Proc. of Digital Image Computing: Techniques and Applications. – Perth, Australia, 1999. – P. 195–200.

17. Sun, C. Fast Stereo Matching Using Rectangular Subregioning and 3D Maximum-Surface Techniques / C. Sun // Intern. Journal of Computer Vision. – 2002. – Vol. 47, № 1. – P. 99–117.

18. Sun, C. Fast Algorithms for Stereo Matching and Motion Estimation / C. Sun // Proc. of Australia-Japan Advanced Workshop on Computer Vision. – Adelaide, Australia, 2003. – P. 38–48.

19. Sun, C. Fast Panoramic Stereo Matching Using Cylindrical Maximum Surfaces / C. Sun, S. Peleg // IEEE Transactions on Systems, Man, and Cybernetics. Part B. – 2004. – Vol. 34, № 1. – P. 760–765.

20. Hirschmüller, H. Real-Time Correlation-Based Stereo Vision with Reduced Border Errors / H. Hirschmüller, P. Innocent, J. Garibaldi // International Journal of Computer Vision. – 2002. – Vol. 47, № 1–3. – P. 229–246.

21. Stefano, L. A Fast Area-Based Stereo Matching Algorithm / L. Stefano, M. Marchionni, S. Mattoccia // Image and Vision Computing. – 2004. – Vol. 22, № 12. – P. 983–1005.

22. Stefano, L. A PC-based Real-Time Stereo Vision System / L. Stefano, M. Marchionni, S. Mattoccia // Machine Graphics & Vision. – 2004. – Vol. 13, № 3. – P. 197–220.

23. Sun, C. Fast Optical Flow Using Cross Correlation and Shortest-Path Techniques / C. Sun // Proc. of Digital Image Computing: Techniques and Applications. – Perth, Australia, 1999. – P. 143–148.

24. Sun, C. Fast optical flow using 3d shortest path techniques / C. Sun // Image and vision computing. – 2002. – Vol. 20, № 13/14. – P. 981–991.

25. Zhao, P. Near Real-Time Optical Flow / P. Zhao, M. Spetsakis // Proc. of the 14th Intern. Conf. on Vision Interface. – Ottawa, ON, Canada, 2001. – P. 47–55.

26. Садыхов, Р.Х. Инструментальная система для обработки видеоинформации / Р.Х. Садыхов, Д.В. Ламовский // Доклады БГУИР. – 2007. – № 4 (20). – С. 175–180.

27. Chang, M. Optical Flow Measurement Based on Boolean Edge Detection and Hough Transform / M. Chang, I. Kim, J. Park // Intern. Journal of Control, Automation and Systems. – 2003. – Vol. 7, № 5. – P. 788–798.

28. Lu, N. Motion Detection Based On Accumulative Optical Flow and Double Background Filtering / N. Lu, J. Wang, L. Yang // Proc. of World Congress on Engineering. – London, UK, 2007. – P. 602–607.

29. Krauchonak, A. Detection of Moving Objects on Videosequences Based on Region Growing Optical Flow / A. Krauchonak // Proc. of the 10th International Conference «Pattern Recognition and Image Analysis: New Informational Technologies» (PRIA-10-2010), St. Petersburg, Russian Federation, December 5–12, 2010. – St. Petersburg, 2010. – Vol. 1. – P. 223–226.

30. Kravchonok, A. Detection of moving objects in video sequences by the computation of optical flow based on region growing / A. Kravchonok // Pattern Recognition and Image Analysis. – 2011. – Vol. 21, № 2. – P. 283–286.

31. Barron, J. Performance of optical flow techniques / J. Barron, D. Fleet, S. Beauchemin // Intern. Journal of Computer Vision. – 1994. – Vol. 12, № 1. – P. 43–77.

32. Horn, B. Robot Vision / B. Horn. – MIT Press. Cambridge, 1986. – 509 p.

33. Lucas, B. An Iterative Image Registration Technique with an Application to Stereo Vision / B. Lucas, T. Kanade // Proc of Intern. Joint Conference on Artificial Intelligence. – Vancouver, BC, Canada, 1981. – P. 674–679.

34. Anandan, P. A Computational Framework and an Algorithm for the Measurement of Visual Motion / P. Anandan // Intern. Journal of Computer Vision. – 1989. – Vol. 2. – P. 283–310.

35. Bartolini, F. Motion Estimation and Tracking for Urban Traffic Monitoring / F. Bartolini, V. Cappellini, C. Giani // Proc of IEEE Intern. Conf. on Image Processing. – Lausanne, Switzerland, 1996. – P. 787–790.

36. Stefano, L. Vehicle Detection and Tracking Using the Block Matching Algorithm / L. Stefano, E. Viarani // Proc. of Intern. Multiconference on Circuits, Systems, Communications and Computer. – Athens, Greece, 1999. – P. 4491–4496.

37. Mattoccia, S. Efficient and optimal block matching for motion estimation / S. Mattoccia, F. Tombari, L. Stefano // 14th IAPR Intern. Conf. on Image Analysis and Processing (ICIAP 2007). – Modena, Italy, 2007. – P. 705–710.

38. Toivonen, T. A New Algorithm for Fast Full Search Block Motion Estimation Based on Number Theoretic Transforms / T. Toivonen // Proc. of the 9th Intern. Workshop on Systems, Signals, and Image Processing. – Manchester, United Kingdom, 2002. – P. 90–94.

39. McDonnell, M. Box-filtering techniques / M. McDonnell // Computer Graphics and Image Processing. –1981. – Vol. 17, № 1. – P. 65–70.

40. Intel Corp. Absolute-Difference Motion Estimation for Intel® Pentium® 4 Processors [Electronic resource]. – Mode of access : http://software.intel.com/en-us/articles/absolute-difference-motion-estimation-for-intel-pentiumr-4-processors/. – Date of access : 31.08.10.

Поступила 26.01.12

Объединенный институт проблем
информатики НАН Беларуси,
Минск, Сурганова, 6
e-mail: alpha_storm@mail.ru

A.I. Kravchonok

FAST OPTICAL FLOW COMPUTATION ALGORITHM BASED ON SSE2-INSTRUCTIONS OF X86 PROCESSOR FAMILY

An algorithm for fast computation of optical flow based on SSE2-instructions of a PC is proposed. The algorithm has a constant complexity of the radius of the optical flow window, applies SSE2 SIMD-commands at all computation stages while working on multi-core processors and uses a parallel mode of operation. The algorithm can significantly speed up the computation of optical flow, which makes it applicable for real time mode of personal computers.