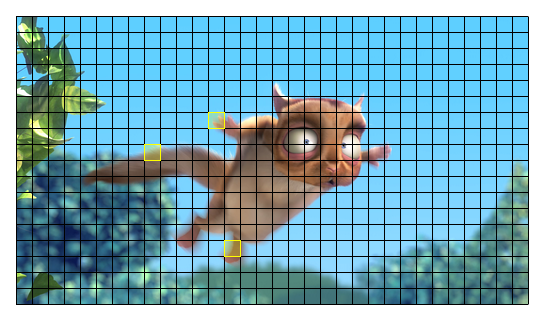
Оценка и компенсация движения

Теория ([https://ru.wikipedia.org/wiki/компенсация движения](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B5%D0%BD%D1%81%D0%B0%D1%86%D0%B8%D1%8F_%D0%B4%D0%B2%D0%B8%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F))

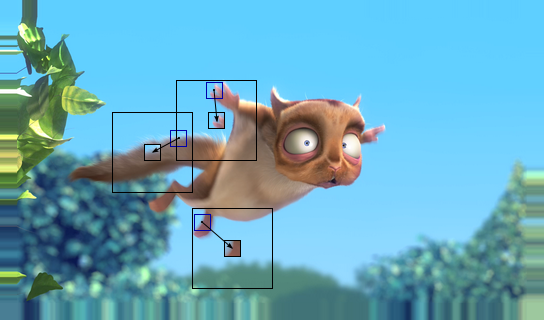
## Пример работы алгоритма

В связи с высокой вычислительной сложностью алгоритмов распознавания образов и недостаточной точности их работы применяют различные методы, позволяющие быстро находить векторы движения (естественно, не без потерь).

1. Загружается текущий кадр.   
2. Кадр делится на блоки (например 16×16).

[](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:MoComp_-_%D0%94%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BD%D0%B0_%D0%B1%D0%BB%D0%BE%D0%BA%D0%B8.png)Поделённый на блоки кадр

3. Производится обход блоков (каждый блок в данном случае обрабатывается отдельно).   
4. При счете одного блока производится обход некоторой окрестности блока в поиске максимального соответствия изображению блока на предыдущем кадре в пределах этой окрестности.

[](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:MoComp_-_%D0%9F%D0%BE%D0%B8%D1%81%D0%BA_%D0%B1%D0%BB%D0%BE%D0%BA%D0%BE%D0%B2.png)Наглядная иллюстрация поиска: изображен предыдущий кадр (тот, в котором производится поиск) и три блока нового кадра, который мы хотим приблизить фрагментами предыдущего

5. Таким образом, после завершения поиска мы получаем набор векторов, указывающий «движение» блоков изображения между кадрами. Эти векторы могут быть естественным образом использованы для создания изображения скомпенсированного кадра, который лучше приближает кадр, для которого производилась компенсация движения.

Анализ вычислительной сложности

Пусть w и h ширина и высота соответственно, а max\_motion – область поиска, block – размер блока, на которые разбивается изображение, тогда для обработки каждого блока нам необходимо совершить:

1. Вычитаний: (max\_motion \* max\_motion)

Всего блоков для обработки:

w \* h / block^2

## Тесты для верификации алгоритма

Для верификации алгоритма предполагается использование нескольких видов тестирования:

1. Визуальное
   * проверка, получился ли ожидаемый результат
2. Аналитическое
   * скорость работы алгоритма
   * профилирование алгоритма

## Подготовка данных

На данном этапе важна разработка алгоритма, без учёта организации ввода и вывода архитектуры MIPS, поэтому для обработки данных в языке C, необходимо получить их в удобном виде.

Для этого было решено использовать готовое решение, работающее с изображением в формате данных bitmap, а именно bitmap\_image.hpp (http://partow.net/programming/bitmap/index.html)

## Модель реализации алгоритма на C. (см. Приложение 1)

## Модель реализации алгоритма на MIPS c расширением MSA. (см. Приложение 2)

Оценка времени работы различных вариаций алгоритма.  
(max\_motion = 8, block = 16, width = 640, height = 360)

|  |  |  |
| --- | --- | --- |
| № | вариант | время |
| 1 | nativ | 250.54 ms |
| 2 | msa\_1\_pixel | 277.61 ms |
| 3 |  |  |

Профилирование алгоритма.   
(max\_motion = 8, block = 16, width = 640, height = 360)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | nativ | | msa\_1\_pixel | |
| 1 |  |  | COPY\_S.df : | 472894 |
| 2 |  |  | INSERT.df : | 472888 |
| 3 |  |  | LD.df : | 693731 |
| 4 |  |  | ST.df : | 466334 |
| 5 |  |  | SUBSUS\_U.df : | 227394 |
| 6 | OPC\_ARITH : | 6278960 | OPC\_ARITH : | 4440938 |
| 7 | OPC\_ARITH\_IMM : | 10276413 | OPC\_ARITH\_IMM : | 10230082 |
| 8 | OPC\_COND\_MOVE : | 7814 | OPC\_COND\_MOVE : | 7630 |
| 9 | OPC\_COPL\_LDST : | 24 | OPC\_COPL\_LDST : | 24 |
| 10 | OPC\_FLT\_LDST : | 24 | OPC\_FLT\_LDST : | 24 |
| 11 | OPC\_LD : | 33885214 | OPC\_LD : | 34896244 |
| 12 | OPC\_LOGIC : | 17876654 | OPC\_LOGIC : | 15278284 |
| 13 | OPC\_LOGIC\_IMM : | 1186021 | OPC\_LOGIC\_IMM : | 1185158 |
| 14 | OPC\_SHIFT : | 143 | OPC\_SHIFT : | 143 |
| 15 | OPC\_SHIFT\_IMM : | 3017019 | OPC\_SHIFT\_IMM : | 2320268 |
| 16 | OPC\_SLT : | 5532 | OPC\_SLT : | 5349 |
| 17 | OPC\_SLT\_IMM : | 5023 | OPC\_SLT\_IMM : | 3965 |
| 18 | OPC\_ST : | 16381524 | OPC\_ST : | 17209131 |
| 19 | OPC\_ST\_COND : | 239 | OPC\_ST\_COND : | 104 |
| 20 | ------- |  | ------- |  |
| 21 | total: 88920605 | 88920605 | total: | 87910586 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Пример работы алгоритма.

|  |  |
| --- | --- |
| prev\_frame | https://vk.com/doc5387385_463124824?hash=844cc6e6f729164b34&dl=5b40b8a4974cf8f4f6&wnd=1 |
| cur\_frame | https://vk.com/doc5387385_463124825?hash=f88fa42785eb40f493&dl=4d864660617fd3c929&wnd=1 |
| motion\_vector | https://vk.com/doc5387385_463124826?hash=1bcc1b338ce9b2473b&dl=67fb344779462c8503&wnd=1 |

Рекомендации для программной реализации на MIPS c расширением MSA.

Нет

Предложение рекомендаций по организации новых команд на основе профилирования.

Нет

Приложение 1

nativ.cpp

#include <stdio.h>

#include <string>

#include <msa.h>

#include <iostream>

#include <climits>

#include "bitmap\_image.hpp"

#include "timer.cpp"

int max\_motion = 8, block = 16;

struct motion\_vector

{

int y;

int x;

};

bitmap\_image previous\_frame("1.bmp");

bitmap\_image current\_frame("2.bmp");

bitmap\_image motion\_frame("2.bmp");

image\_drawer draw(motion\_frame);

rgb\_t colour\_prev, colour\_cur;

int width = previous\_frame.width();

int height = previous\_frame.height();

long SAD(int y, int x)

{

long error;

if ((x > 0) && (y > 0) && (x < width) && (y < height))

{

previous\_frame.get\_pixel(x, y, colour\_prev);

current\_frame.get\_pixel(x, y, colour\_cur);

error = (abs(colour\_prev.red - colour\_cur.red)) +

abs(colour\_prev.green - colour\_cur.green) +

abs(colour\_prev.blue - colour\_cur.blue);

}

return error;

}

int main()

{

int x = 0, y = 0, t = 0, i = 0, j = 0;

draw.pen\_color(255, 0, 0);

long min\_error, error;

motion\_vector motion\_vector;

std::cout << "width = " << width << "; height = " << height << std::endl;

int n = 1;

stopwatch sw;

sw.tick();

for (int i = 0; i < n; i++)

{

for (y = 0; y < height; y = y + block)

{

for (x = 0; x < width; x = x + block)

{

min\_error = LONG\_MAX;

motion\_vector.y = 0;

motion\_vector.x = 0;

for (int i = -max\_motion; i < max\_motion; i = i + 1)

{

for (int j = -max\_motion; j < max\_motion; j = j + 1)

{

error = SAD(y + i, x + j);

if (error < min\_error)

{

motion\_vector.x = i;

motion\_vector.y = j;

min\_error = error;

}

}

}

if (SAD(y, x) == min\_error)

{

motion\_vector.y = 0;

motion\_vector.x = 0;

}

draw.line\_segment(x, y, x + motion\_vector.x, y + motion\_vector.y);

}

}

}

sw.tock();

std::cout << "version " << sw.report\_ms() / (double)n << "ms.\n";

sw.reset();

motion\_frame.save\_image("motion\_vector.bmp");

return 0;

}

Приложение 2 (msa)

msa\_1\_pixel.cpp

# include <stdio.h>

# include <string>

# include <msa.h>

# include <iostream>

# include <climits>

# include "bitmap\_image.hpp"

# include "timer.cpp"

int max\_motion = 8, block = 16;

struct motion\_vector

{

int y;

int x;

};

bitmap\_image previous\_frame("1.bmp");

bitmap\_image current\_frame("2.bmp");

bitmap\_image motion\_frame("1.bmp");

image\_drawer draw(motion\_frame);

rgb\_t colour\_prev, colour\_cur;

int width = previous\_frame.width();

int height = previous\_frame.height();

v4u32 SAD(int y, int x)

{

v4u32 error;

if ((x > 0) && (y > 0) && (x < width) && (y < height))

{

v4u32 pix\_col\_cur;

v4i32 pix\_col\_prev;

previous\_frame.get\_pixel(x, y, colour\_prev);

pix\_col\_prev[0] = colour\_prev.red;

pix\_col\_prev[1] = colour\_prev.green;

pix\_col\_prev[2] = colour\_prev.blue;

current\_frame.get\_pixel(x, y, colour\_cur);

pix\_col\_cur[0] = colour\_cur.red;

pix\_col\_cur[1] = colour\_cur.green;

pix\_col\_cur[2] = colour\_cur.blue;

error = \_\_builtin\_msa\_subsus\_u\_w(pix\_col\_cur, pix\_col\_prev);

}

return error;

}

int main()

{

int x = 0, y = 0, t = 0, i = 0, j = 0;

draw.pen\_color(255, 0, 0);

v4u32 min\_error, error;

motion\_vector motion\_vector;

std::cout << "width = " << width << "; height = " << height << std::endl;

int n = 1;

stopwatch sw;

sw.tick();

for (int i = 0; i < n; i++)

{

for (y = 0; y < height; y = y + block)

{

for (x = 0; x < width; x = x + block)

{

for (int i = 0; i < 4; i++)

{

min\_error[i] = LONG\_MAX;

}

motion\_vector.y = 0;

motion\_vector.x = 0;

for (int i = -max\_motion; i < max\_motion; i = i + 1)

{

for (int j = -max\_motion; j < max\_motion; j = j + 1)

{

error = SAD(y + i, x + j);

if (error[0] < min\_error[0])

{

motion\_vector.x = i;

motion\_vector.y = j;

min\_error = error;

}

else if (error[1] < min\_error[1])

{

motion\_vector.x = i;

motion\_vector.y = j;

min\_error = error;

}

else if (error[2] < min\_error[2])

{

motion\_vector.x = i;

motion\_vector.y = j;

min\_error = error;

}

}

}

error = SAD(y, x);

if (error[0] <= min\_error[0])

{

motion\_vector.x = 0;

motion\_vector.y = 0;

}

else if (error[1] <= min\_error[1])

{

motion\_vector.x = 0;

motion\_vector.y = 0;

}

else if (error[2] <= min\_error[2])

{

motion\_vector.x = 0;

motion\_vector.y = 0;

}

draw.line\_segment(x, y, x + motion\_vector.x, y + motion\_vector.y);

}

}

}

sw.tock();

std::cout << "version " << sw.report\_ms() / (double)n << "ms.\n";

sw.reset();

motion\_frame.save\_image("motion\_vector.bmp");

return 0;

}