

маСанкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчёт о лабораторной работе №4

Дисциплина: Методы и средства цифровой обработки информации

Выполнил студент гр. 13541/1

_____ А.А. Дроздовский
(подпись)

Руководитель

_____ Н.А. Абрамов
(подпись)

«__» _____ 2017 г.

Санкт – Петербург
2017

СОДЕРЖАНИЕ

1. Цель работы	3
2. Ход работы.....	3
2.1. Фильтр Гаусса	Ошибка! Закладка не определена.
2.2. Оператор Лапласиана	Ошибка! Закладка не определена.
2.3. Повышение резкости... ..	Ошибка! Закладка не определена.
3. Выводы	9
ПРИЛОЖЕНИЕ	10
Текст программы.....	10

1. Цель работы

Изучить алгоритм поиска векторов смещения на последовательных кадрах.

Восстановить предшествующий кадр, зная вектора смещения.

2. Ход работы

Алгоритм работы:

- 1) Считать два последовательных кадра $t-1$ и t ;
- 2) Изображение $t-1$ разбить на блоки размера $Block$ с шагом $Step$;
- 3) Для каждого блока найти область поиска на изображении t , размером $Search_Area*2 \times Search_Area*2$;
- 4) В области поиска найти блок с наименьшим SAD;
- 5) Построить вектор смещения;
- 6) Восстановить кадр $t-1$.

$$SAD = \sum_{i=0, j=0}^{Block} |Px_{t-1}(i, j) - Px_t(i, j)|, \text{ где}$$

Px_{t-1}, Px_t – оттенок пикселя (с координатами i, j) в изображениях $t-1$ и t соответственно.

Ниже представлены два последовательных кадра:

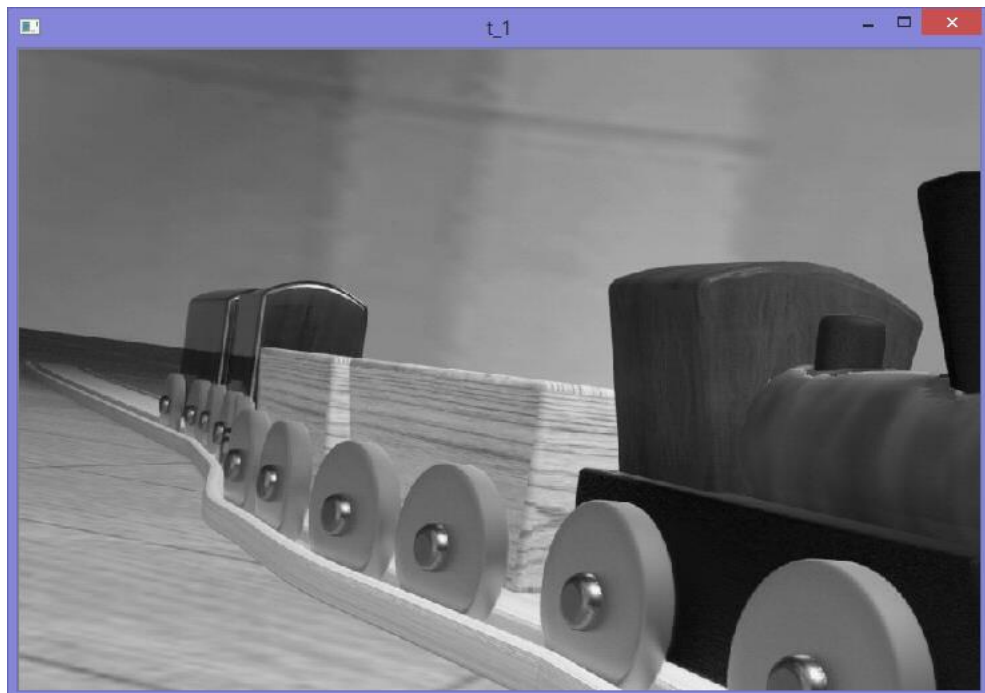


Рис. 2.1. Кадр $t-1$.

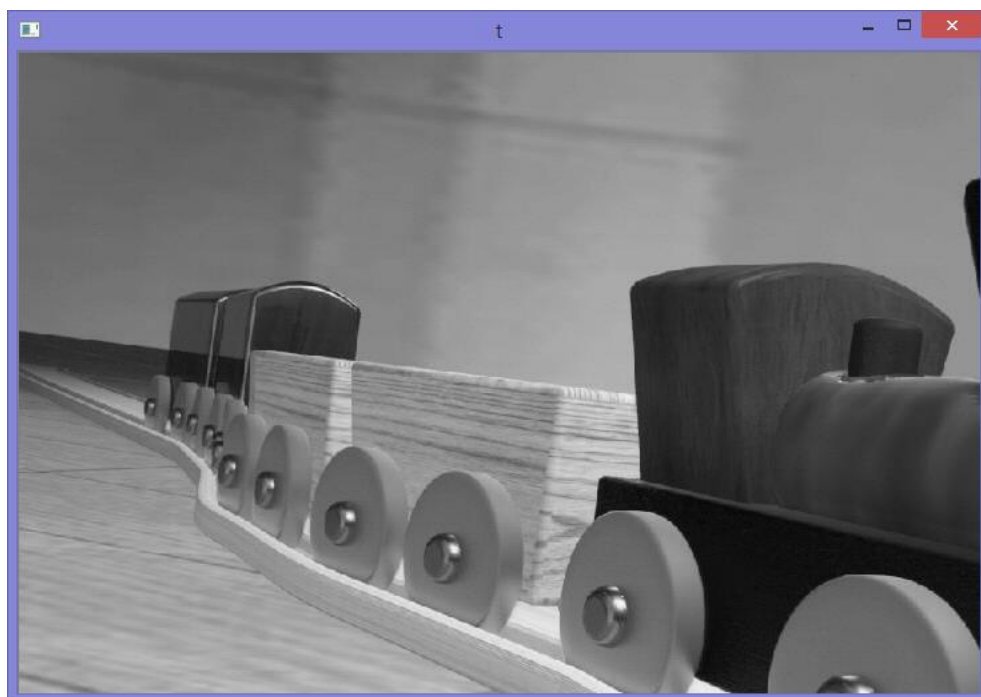


Рис. 2.2. Кадр t .

2.1. Block = 9, Step = 9, Search_Area = 25

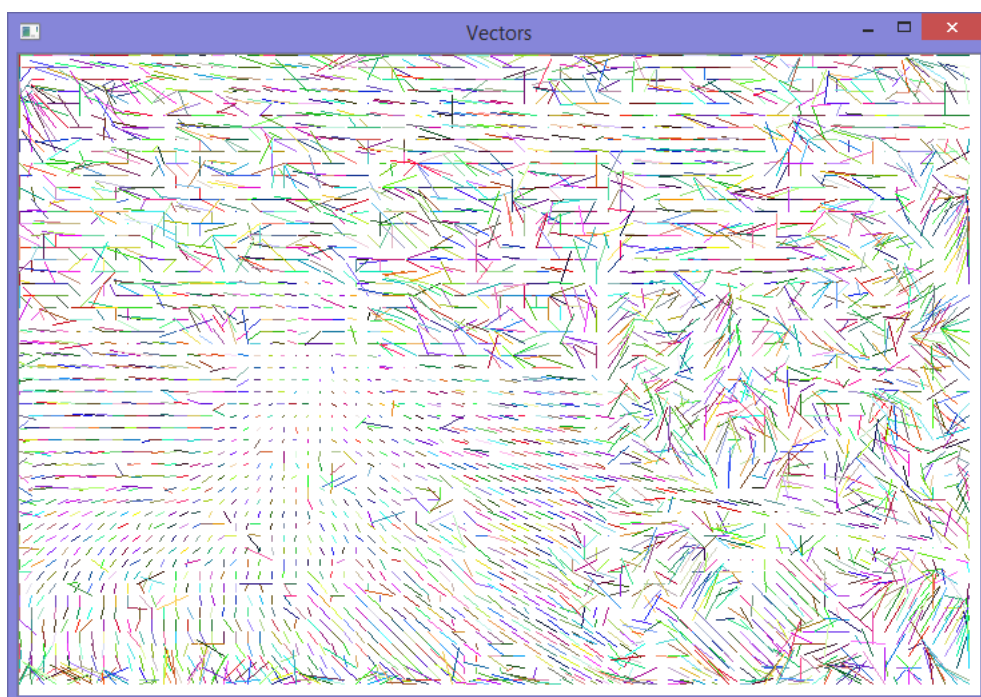


Рис. 2.3. Вектора смещения.



Рис. 2.4. Восстановленный кадр.

2.2. Block= 9, Step = 4 Search_Area = 25

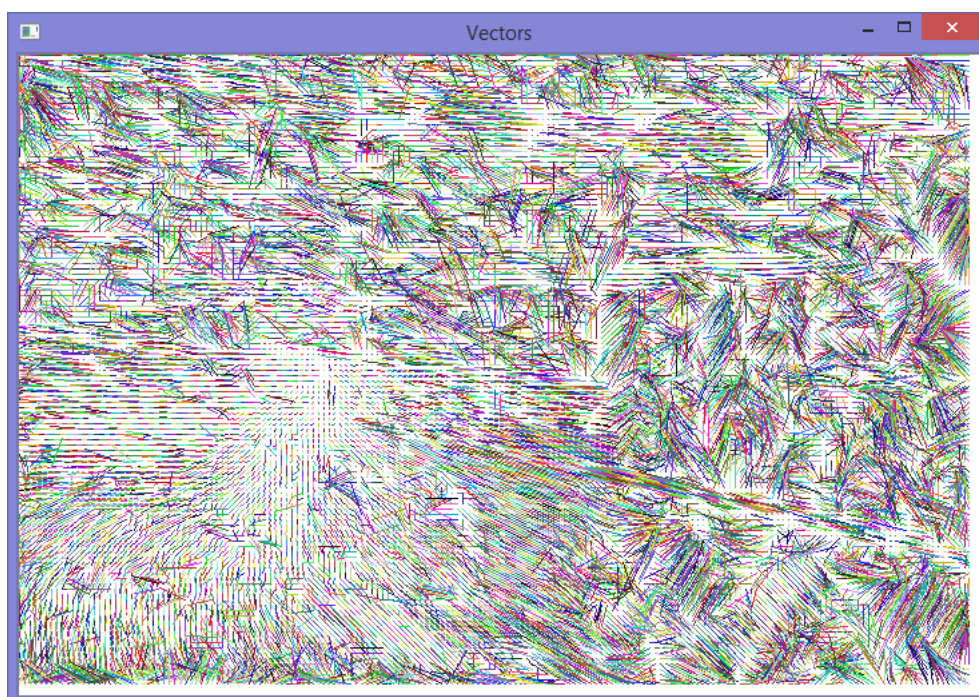


Рис. 2.5. Вектора смещения.

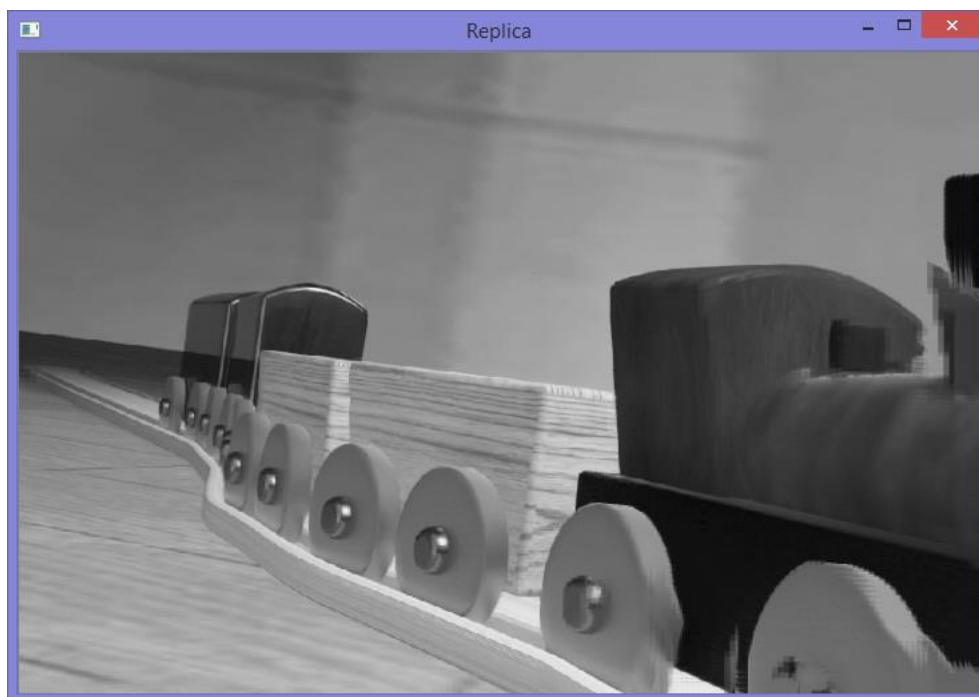


Рис. 2.6. Восстановленный кадр.

2.3. Block= 5, Step = 5, Search_Area = 25

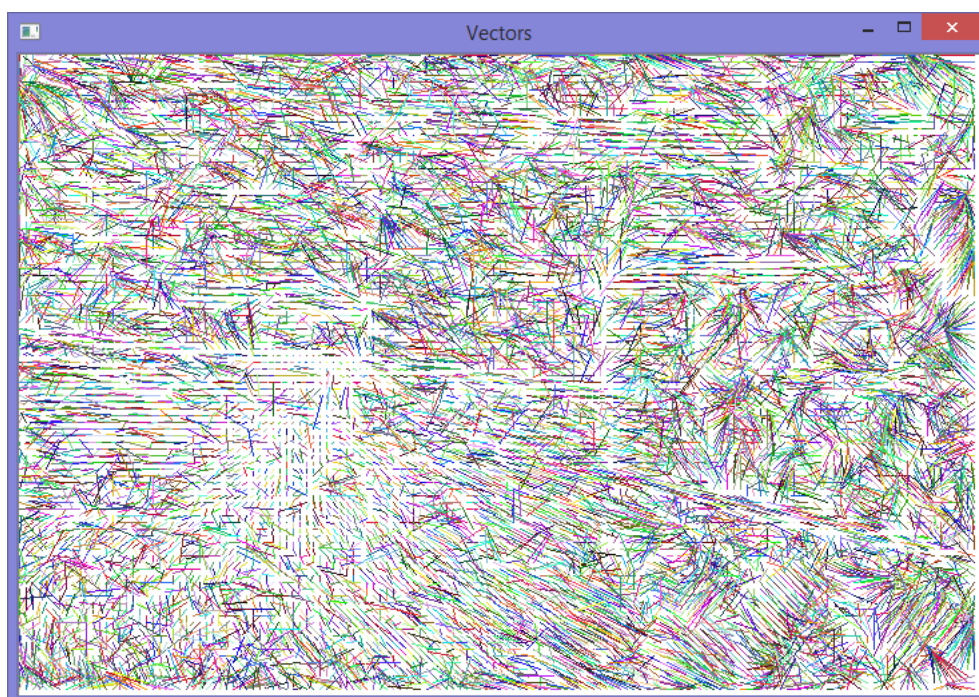


Рис. 2.7. Вектора смещения.



Рис. 2.8. Восстановленный кадр.

2.4. Block = 5, Step = 2, Search_Area = 25

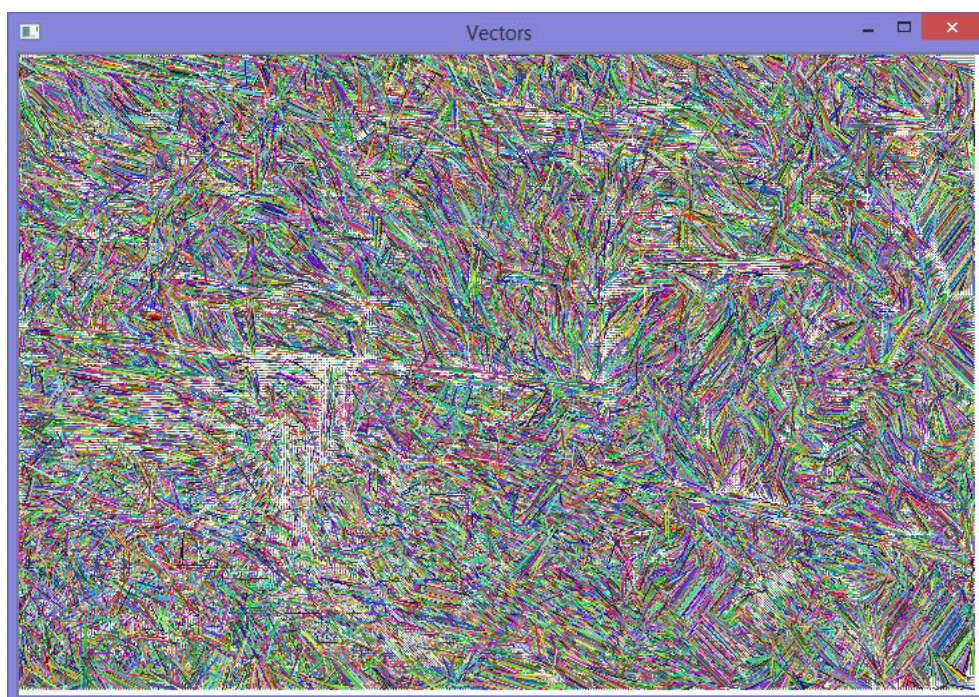


Рис. 2.9. Вектора смещения.

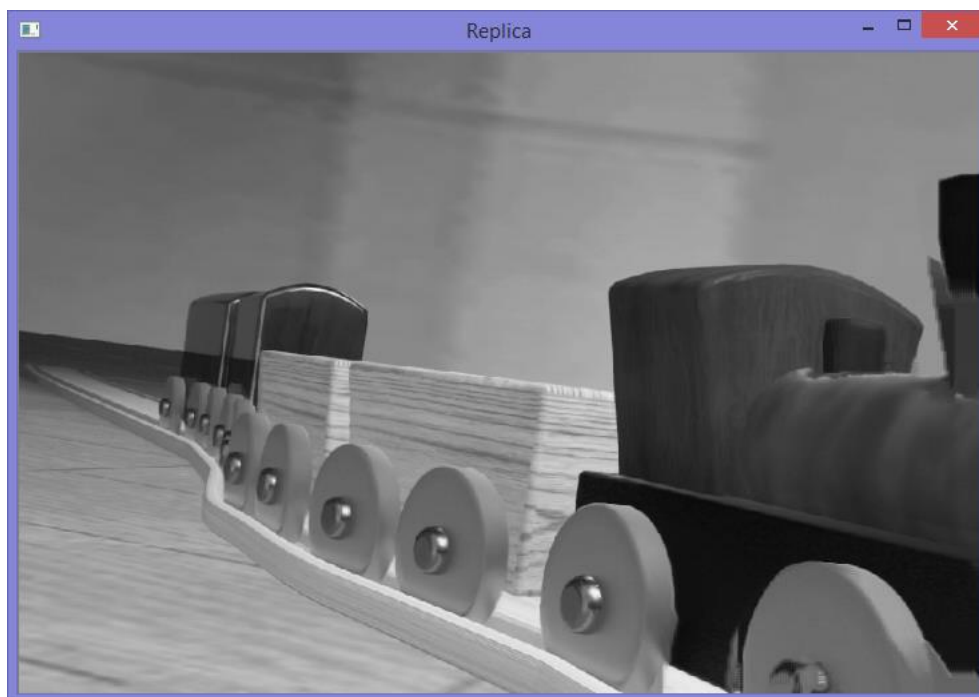


Рис. 2.10. Восстановленный кадр.

2.5. Block = 4, Step = 2, Search_Area = 32

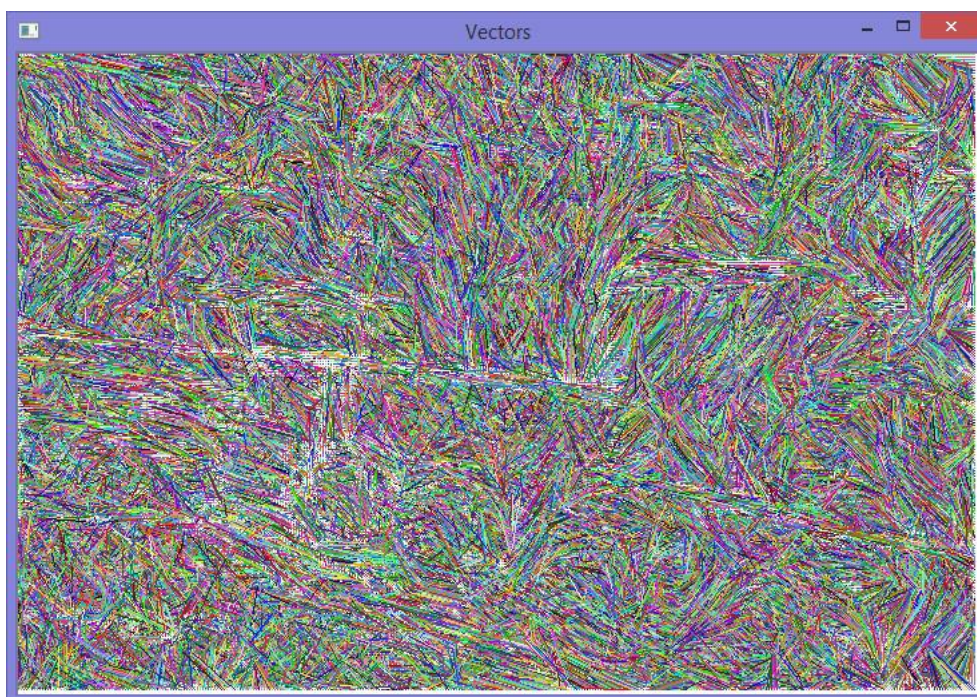


Рис. 2.11. Вектора смещения.

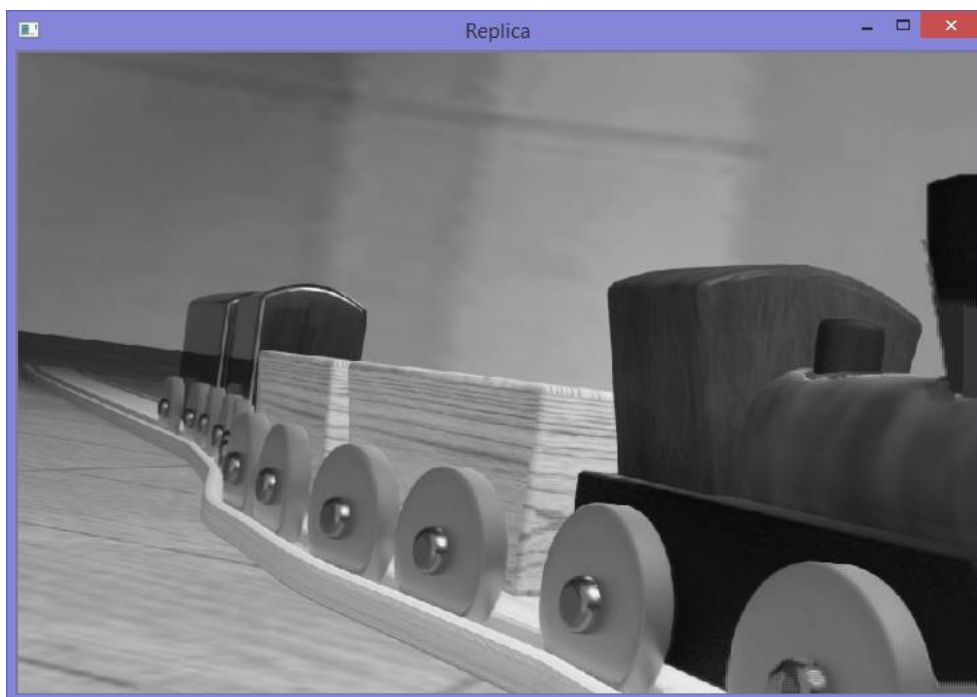


Рис. 2.12. Восстановленный кадр.

3. Выводы

Из результатов видно, что алгоритм, применяемый в данной работе, полностью рабочий, однако качество выходного изображения сильно зависит от параметров, а именно от размера блока (Block), шага блоков(Step), размера поискового окна(Search_Area). Чем больше размер окна для поиска, тем выше шанс найти наиболее подходящее смещение, но при его увеличении значительно уменьшается быстродействие (при опыте в пункте 2.5. полное выполнение алгоритма заняло ~15 минут). Чем меньше шаг и размер блока, тем точнее выходной результат.

Для улучшения качества выходного изображения, при наложении двух и более пикселей нужно с весом SAD (чем меньше SAD, тем выше вероятность совпадения) высчитать выходные значения пикселя. Но из-за трудоемкости и для упрощения алгоритма в работе значения SAD никак не влияют на конечный оттенок пикселя. В качестве оттенка записывается среднее арифметическое из восстановленных в пиксель значений.

Текст программы

```

#define Block 4
#define Search_Area 32
#define Step 2
#define _SAD true
#define _SSD false

using namespace cv;
using namespace std;

template <typename _Tp>
Mat_<uchar> Convert(Mat_<_Tp> input);
template <typename _Tp>
_Block Search_Block(Mat_<_Tp> t_1, Mat_<_Tp> t, int row, int col, int count_row, int
count_col, int i_old, int j_old);
template <typename _Tp>
void White(Mat_<_Tp> &input);

int main()
{
    namedWindow("LR4");
    String filename("LR4.avi"); // Имя видеофрагмента
    VideoCapture capture = VideoCapture(filename);
    Mat_<uchar> frame, // Для считывания кадра
    t, t_1; // Два последовательных
кадра
    vector<_Block> Blocks_info; // Массив содержащий информацию про блоки
    vector<Mat_<uchar>> Blocks // Массив для хранения блоков размера Block x
Block
    , Search_win; // Массив для хранения окна для поиска
    // Считывание первого кадра
    capture >> frame;
    cvtColor(frame, t_1, CV_BGR2GRAY );
    imshow("t_1", t_1);
    // Считываем второго кадр
    capture >> frame;
    cvtColor(frame, t, CV_BGR2GRAY);
    imshow("t", t);
    Mat_<Vec3b> vector // Изображение для отрисовки векторов
    = Mat_<Vec3b>(t.rows, t.cols, Vec3b(255, 255, 255));
    Mat_<int> replica = // Для накопления оттенков
    восстановленного кадра
    Mat_<int>(t_1.rows, t_1.cols);
    Mat_<uchar> accumulator = // Аккумулятор количества перезаписи значения
    пикселя
    Mat_<uchar>(t_1.rows, t_1.cols);
    White(replica);
    White(accumulator);
    // Формируем 2 массива из кадра t_1 и t:
    // 1 - содержащий блоки размером Block x Block
    // 2 - содержащий окная для поиска
    for (int row = 0; row < t_1.rows; row += Step)
    {
        if (t_1.rows - row < Block)
            row = t_1.rows - Block;
        // формирование окна для поиска по вертикали
        int i_window = max(0, row - Search_Area);
        int count_row = min(row - i_window, Search_Area)
        + min(Search_Area, t_1.rows - row);
    }
}

```

```

        for (int col = 0; col < t_1.cols; col += Step)
        {
            if (t_1.cols - col < Block)
                col=t_1.cols - Block;
            // формирование окна для поиска по горизонтали
            int j_window = max(0,col-Search_Area);
            int count_col = min(col - j_window, Search_Area)
                + min(Search_Area, t_1.cols - col);
            // Запись в массив информации о текущем блоке
            _Block bl = Search_Block(t_1, t, i_window, j_window, count_row,
count_col, row, col);
            Blocks_info.push_back(bl);
            // Отрисовка векторов:
            int RED = rand() % 256;
            int GREEN = rand() % 256;
            int BLUE = rand() % 256;
            line(vector, Point(bl.ret_j_old(),bl.ret_i_old()),
                Point(bl.ret_j_new(), bl.ret_i_new()),
                Scalar(RED, GREEN, BLUE), 1, 8, 0);
            // Восстановление изображения
            for (int _row = 0; _row < Block; _row++)
                for (int _col = 0; _col < Block; _col++)
                {
                    replica.at<int>(bl.ret_i_old() + _row,
bl.ret_j_old() + _col)
                        += t.at<uchar>(bl.ret_i_new() + _row,
bl.ret_j_new() + _col);
                    accumulator.at<uchar>(bl.ret_i_old() + _row,
bl.ret_j_old() + _col) += 1;
                }
            if (col == t_1.cols - Block)
                break;
        }
        if (row == t_1.rows - Block)
            break;
    }
    // Подготовка восстановленного изображения к выводу на экран
    Mat<uchar> rep =
        Mat<uchar>(replica.rows, replica.cols);
    for (int i = 0; i < replica.rows*replica.cols; i++)
        rep.at<uchar>(i) = round(replica.at<int>(i) / accumulator.at<uchar>(i));
    imshow("Vectors", vector);
    imshow("Replica", rep);
    cv::waitKey(0);
    return 0;
}

/**
 * Преобразование Mat_<typename> в Mat_<uchar>
 *
 * Исходная матрица изображения нормализуется
в формат оттенков пикселей от 0 до 255.
Затем выводится в качестве выходного параметра.
 * @input Mat_<typename> исходное изображение
 * @return Mat_<uchar>
 */
template <typename _Tp>
Mat<uchar> Convert(Mat_<_Tp> input)
{
    Mat<uchar> output(input.rows, input.cols);
    normalize(input, input, 0, // Линейное растяжение
        255, NORM_MINMAX, -1, Mat());
    output = input;
}

```



```

        return output;
    }

    /**
     * Поиска подходящего блока
     *
     * В окне для поиска находится блок с наименьшим
     * значением SAD. Формируется переменная класса _Block
     * которая содержит координаты исходного блока,
     * координаты подходящего блока,
     * найденный SAD.
     * Переменная выводится в качестве выходного параметра
     * @t_1 - Mat_<typename> - исходное изображение,
     * содержащее блок для поиска
     * @t - Mat_<typename> - исходное изображение,
     * содержащее окно для поиска
     * @row - int - начальная координата окная для поиска по вертикали
     * @col - int - начальная координата окная для поиска по горизонтали
     * @count_row, @count_col - int - величина окна для поиска
     * @i_old, @j_old - int - координаты текущего блока
     * @return _Block - информация по текущему блоку
     */
    template <typename _Tp>
    _Block Search_Block(Mat_<_Tp> t_1, Mat_<_Tp> t, int row, int col, int count_row, int
    count_col, int i_old, int j_old)
    {
        _Block b1 = _Block(i_old, j_old);
        int SAD = 0;
        int SSD = 0;
        for (int i = row; i < row + count_row - Block; i++)
            for (int j = col; j < col + count_col - Block; j++)
            {
                SAD = 0;
                for (int ii = 0; ii < Block; ii++)
                {
                    for (int jj = 0; jj < Block; jj++)
                        SAD += abs(t_1.at<_Tp>(i_old + ii, j_old + jj) -
t.at<_Tp>(i + ii, j + jj));
                    if (b1.ret_SAD() < SAD)
                        break;
                }
                if (_SAD)
                {
                    if (b1.ret_SAD() > SAD)
                    {
                        b1.reload(i, j, SAD, 9999);
                    }
                }
            }
        return b1;
    }

    /**
     * Заливка изображения черным цветом
     *
     * Исходная матрица изображения заполняется
     * черным цветом (оттенком с индексом 0)
     * @input Mat_<typename> исходное изображение
     */
    template <typename _Tp>
    void White(Mat_<_Tp> &input)
    {
        for (int i = 0; i < input.rows*input.cols; i++)
            input.at<_Tp>(i) = 0;
    }

```

}