**Свёртка с окном**

**Анализ вычислительной сложности**

Наивная реализация с матрицей свёртки размером n требует порядка операций сложения и умножения для каждого канала. Для целочисленной версии дополнительно требуется  операций битового сдвига для каждого канала.

**Тесты для верификации алгоритма**

Для верификации алгоритма предполагается использование нескольких видов тестирования:

Визуальное:

Изменение изображения в соответствии с применяемым фильтром.

Аналитическое:

Сравнение эталонного алгоритма и целочисленного аналога.

**Подготовка данных**

Доступ к изображение с использованием сторонней библиотеки является наиболее ресурсозатратной частью, поэтому была создана функция, которая заносит изображение в память и не использует повторной загрузки из файла. Ниже приведена реализация описанной функции:

int\*\*\* image\_to\_memory(bitmap\_image file){

int width = file.width();

int height = file.height();

rgb\_t color;

int \*\*\*image = (int\*\*\*)malloc(height \* sizeof(int\*\*));

for(int i = 0; i < height; i++){

image[i] = (int\*\*)malloc(width \* sizeof(int\*));

for(int j = 0; j < width; j++){

file.get\_pixel(j, i, color);

image[i][j] = (int\*)malloc(3 \* sizeof(int));

image[i][j][0] = color.red;

image[i][j][1] = color.green;

image[i][j][2] = color.blue;

}

}

return image;

}

Использование типа int вместо unsigned char является более затратным по памяти, но позволяет достичь лучшего времени выполнения. При необходимости уменьшения выделяемой памяти возможна безболезненная замена типа на unsigned char.

**Реализация алгоритма на C**

Ниже представлен код наивной реализации алгоритма, которая была описана ранее. Используем его в качестве эталона при тестировании.

void naive\_version(int\*\*\* original, int\*\*\* result, int width, int height, int\*\* mat, int size){

int bit = 16;

int dS = size / 2;

for (int y = dS; y < height – dS; ++y){

for (int x = dS; x < width – dS; ++x){

float redF = 0, greenF = 0, blueF = 0;

for (int k = -dS; k <= dS; ++k) {

for (int m = -dS; m <= dS; ++m) {

int red = original[y + k][x + m][0];

int green = original[y + k][x + m][1];

int blue = original[y + k][x + m][2];

redF += (red \* mat[k][m]);

greenF += (green \* mat[k][m]);

blueF += (blue \* mat[k][m]);

}

}

result[y][x][0] = clamp(redF);

result[y][x][1] = clamp(greenF);

result[y][x][2] = clamp(blueF);

}

}

}

**Визуальный тест**

Дальнейшие измерения проводятся на фильтре размытия с помощью матрицы Гаусса.

|  |  |
| --- | --- |
| **До размытия** | **После размытия** |
| **D:\Работа\qp\Convolution Matrix\ship.bmp** | **D:\Работа\qp\Convolution Matrix\Result.bmp** |

**Аналитический тест**

Сравнение алгоритма с эталонным дало полное совпадение на случайных тестируемых блоках при шаге масштабирования в 16 бит.

**Использование SIMD парадигмы для оптимизации алгоритма**

Для потенциальной оптимизации алгоритма предлагается обрабатывать все три канала одновременно, в данном примере рассматривается трёхканальное изображение, поэтому применение SIMD парадигмы позволит сократить количество вызываемых команд в три раза, но потребует дополнительных расходов на инициализацию.

void msa\_integer\_version(int\*\*\* original, int\*\*\* result, int width, int height, int\*\* mat, int size){

int bit = 16;

int dS = size / 2;

for (int y = dS; y < width - dS; ++y) {

for (int x = dS; x < height - dS; ++x) {

v4i32 resultRGB, tempRGB;

for (int k = - dS; k <= dS; ++k) {

for (int m = -dS; m <= dS; ++m) {

tempRGB[0] = original[y + k][x + m][0];

tempRGB[1] = original[y + k][x + m][1];

tempRGB[2] = original[y + k][x + m][2];

tempRGB = tempRGB \* mat[k][m];

tempRGB = tempRGB >> bit;

resultRGB += tempRGB;

}

}

clamp(resultRGB);

result[y][x][0] = resultRGB[0];

result[y][x][1] = resultRGB[1];

result[y][x][2] = resultRGB[2];

}

}

}

**Статистика вызовов команд**

INSERT.df : 4

ADDV.df : 32400

MULV.df : 32400

LD.df : 194400

LDI.df : 2

MSA: 1

OPC\_ARITH : 633404

OPC\_ARITH\_IMM : 880706

OPC\_COND\_MOVE : 1213

OPC\_COPL\_LDST : 24

OPC\_FLT\_LDST : 24

OPC\_LD : 2336361

OPC\_LOGIC : 1226420

OPC\_LOGIC\_IMM : 75199

OPC\_SHIFT : 131

OPC\_SHIFT\_IMM : 333477

OPC\_SLT : 774

OPC\_SLT\_IMM : 454

OPC\_ST : 1343400

OPC\_ST\_COND : 44

SRA.df : 32400

ST.df : 97202

-------

total: 7220440