МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Факультет «Информационные технологии и прикладная математика»

Кафедра «Вычислительная математика и программирование»

**Лабораторная работа №2**

**по курсу «Программирование графических процессоров»**

**Изучение технологии CUDA**

Выполнил: А.Д. Ушаков

Группа: 8О-406Б

Преподаватели: К.Г. Крашенинников,

А.Ю. Морозов

Москва, 2018

**Условие**

Научиться использовать GPU для обработки изображений. Использование текстурной памяти.

В данной лабораторной работе используются только цветовые составляющие изображения (r g b), альфа-канал не учитывается. При расчетах значений допускается ошибка в ±1. Ограничение: w < 216 и h < 216.

Вариант 4: SSAA.  
Необходимо реализовать избыточную выборку сглаживания. Исходное

изображение представляет собой “экранный буфер”, на выходе должно быть

сглаженное изображение, полученное уменьшением исходного.

Входные данные. ​На первой строке задается путь к исходному изображению,

на второй, путь к конечному изображению. На следующей строке, два числа wn и hn -

размеры нового изображения, гарантируется, что размеры исходного изображения

соответственно кратны им. w\*h ≤ 4 \* 10 .

**Программное и аппаратное обеспечение**

Name : GeForce GTX 1060

Compute capability : 6.1  
Total Global Memory : 1134690304  
Shared memory per block: 49152  
Registers per block : 65536  
Warp size : 32  
Max threads per block : 1024  
Total constant memory : 65536

Процессор Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, 2801 МГц, ядер: 4, логических процессоров: 8

Оперативная память 8ГБ

Жесткий диск 1ТБ

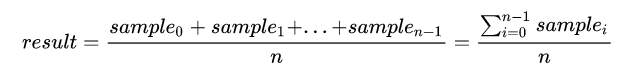
Программное обеспечение:

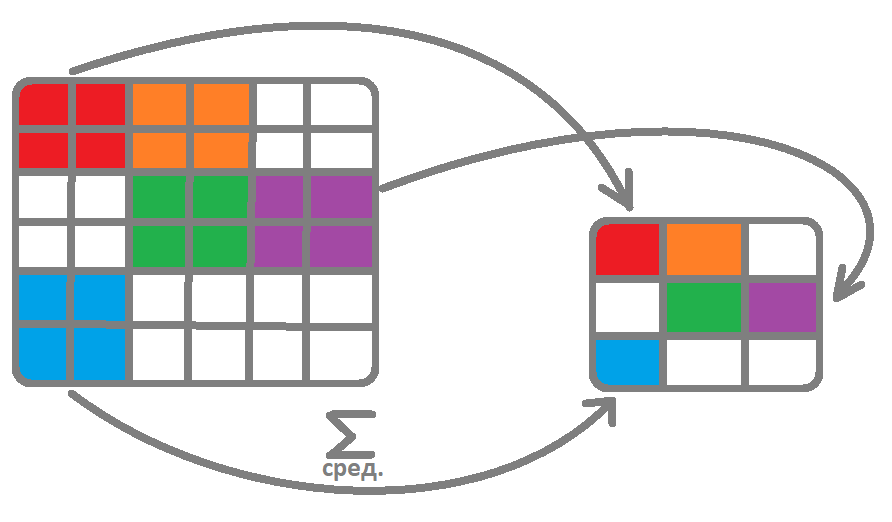
nvcc V7.5.17

g++ (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0 20160609

**Метод решения**

GPU определяет цвет каждого пикселя на основе выборки из центра этого пикселя. Итоговый цвет пикселя получается из усредненной суммы всех пикселей выборки.





**Описание программы**

Программа состоит из 3-х функций: main, ssaa и kernel.  
В main считываются данные со стандартного ввода, заполняется массив данными об исходной картинке. Далее вызывается функция ssaa с переданными в нее данными о картинке, её размерами и размерами новой, уменьшенной картинки.

В ssaa выделяется память под текстурный массив и копируются в него данные о картинке, настраивается конфигурация отображения данных. Далее вызывается функция kernel c количеством блоков 32x32 и количеством тредов в блоке 32х32.

В kernel производится вычисление пикселей новой картинки. Каждый тред вычисляет сумму пикселей выборки и ставит её на нужное место в массиве пикселей новой картинки.

\_\_global\_\_ void kernel(uchar4\* dst, int w, int h, int wn, int hn, int kw, int kh) {

int idx = threadIdx.x + blockIdx.x \* blockDim.x;

int idy = threadIdx.y + blockIdx.y \* blockDim.y;

int offsetx = blockDim.x \* gridDim.x;

int offsety = blockDim.y \* gridDim.y;

int x, y, yp, xp;

int cur\_idx\_x = idx;

int cur\_idx\_y = idy;

uchar4 p;

int px, py, pz;

for (y = idy \* kh; y < h; y += offsety \* kh) {

for (x = idx \* kw; x < w; x += offsetx \* kw) {

px = py = pz = 0;

for (yp = y; yp < y + kh; ++yp) {

for (xp = x; xp < x + kw; ++xp) {

p = tex2D(tex, xp, yp);

px += p.x;

py += p.y;

pz += p.z;

}

}

px = (int)(px / (kh \* kw));

py = (int)(py / (kh \* kw));

pz = (int)(pz / (kh \* kw));

dst[cur\_idx\_y \* wn + cur\_idx\_x] = make\_uchar4(px, py, pz, 255);

cur\_idx\_x += offsetx;

}

cur\_idx\_x = idx;

cur\_idx\_y += offsety;

}

}

**Результаты**

Исходная картинка



Сравнение после применения алгоритма

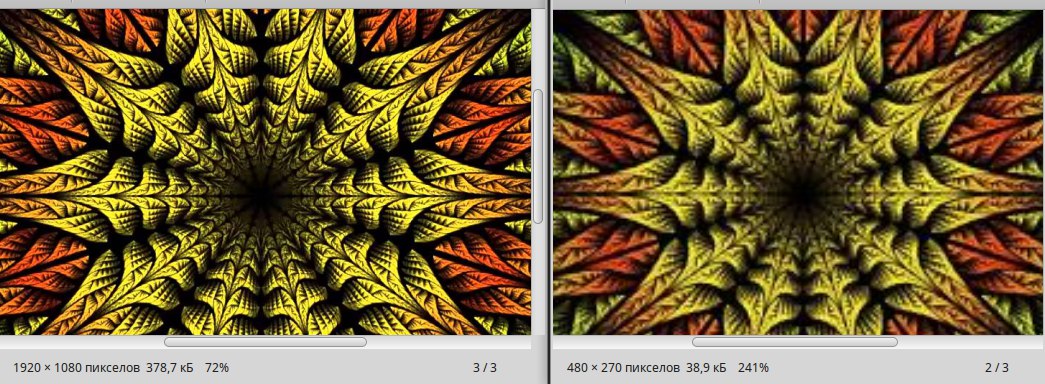




Исходная картинка

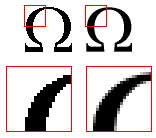


Сравнение после применения алгоритма



|  |  |  |
| --- | --- | --- |
| Исходный размер | Конечный размер | Время выполнения ядра |
| 1920х1080 | 480х270 | 90.2 us |
| 1920x1080 | 160х90 | 246.08 us |
| 1920x1080 | 30x30 | 1.34 ms |
| 1920x1200 | 960x600 | 122.15 us |
| 1920x1200 | 240x150 | 217.31 us |

**Выводы**

Реализованный алгоритм используется для устранения эффекта «зубчатости», возникающего на краях изображения. Проблема алгоритма состоит в том, что для монитора разрешением 1920х1080 нужно иметь картинку размером вдвое выше чем у монитора, что сильно сказывается на производительности.

Как видно из приведенных выше примеров, алгоритм все же не плохо справляется с задачей: картинка плавно переходит места стыков убирая эффект лесенки.