МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №*2*
по курсу «Программирование графических процессоров»

Обработка изображений на GPU. Фильтры.

Выполнил: *Чистяков К.С.*
Группа: *8О-406Б-21*
Преподаватель: А.Ю. Морозов

Москва, 2025

Условие

1. **Цель работы.** Научиться использовать GPU для обработки изображений. Использование текстурной памяти и двухмерной сетки потоков.
2. **Вариант задания.** 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 исходномуизображению, 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. $w*h \leq 10^8$.

Программное и аппаратное обеспечение

Графический процессор (Google Colab)

Compute capability	7.5
Name	Tesla T4
Total Global Memory	15828320256
Shared memory per block	49152
Registers per block	65536
Warp size	32
Max threads per block	(1024, 1024, 64)
Max block	(2147483647, 65535, 65535)
Total constant memory	6553
Multiprocessors count	40

Процессор AMD Ryzen 5 4500U with Radeon Graphics (2.38 GHz)

Technology	7 nm
Cores	6
Threads	6
Core Speed	1390 MHz
Cache (L1 Data)	6 x 32 KBytes (8-way)

Cache (L1 Inst.)	6 x 32 KBytes (8-way)
Cache (Level 2)	6 x 512 KBytes (8-way)
Cache (Level 3)	2 x 4 MBytes (16-way)

Оперативная память

Number of modules	2
Module Manuf.	Samsung
Module Size	8 GBytes (total - 16 GBytes)
Generation	DDR4

Жесткий диск - 512 ГБ SSD
OS - Windows 10 Домашняя + Linux
IDE - VS Code + Google Colab
Compiler - nvcc, gcc

Метод решения

Общее описание алгоритма: проходимся по изображению блоками пикселей определенного размера (w/wn и h/hn, где w и h - размеры исходного изображения, а wn и hn - размеры выходного изображения) и усредняем значения по x, y, z, w. Далее записываем результат в выходной массив.

Описание программы

Один файл, включающий три функции: CSC - для отлова ошибок; kernel - для работы ядра (тут непосредственно и происходит избыточная выборка сглаживания) и main - тут мы считываем названия входного и выходного файлов, размеры нового изображения, потом работаем с файлом, считываем исходные размеры изображения и бинарные данные. Создаем текстурный объект. Далее работа ядра. Возвращаем полученных результирующий массив обратно на CPU. Передаём всё в выходной файл.

Ядро:

```
__global__ void kernel(cudaTextureObject_t tex, uchar4 *out, int w, int h, int wn, int hn) {
   int idx = blockDim.x * blockIdx.x + threadIdx.x;
            int idy = blockDim.y * blockIdx.y + threadIdx.y;
           int offsetx = blockDim.x * gridDim.x;
            int offsety = blockDim.y * gridDim.y;

   int step_x = w / wn;
   int step_y = h / hn;
```

```
    int size = step_x * step_y;

    for(int y = idy; y < hn; y += offsety){
                    for(int x = idx; x < wn; x += offsetx) {

        float4 blockAcc = make_float4(0, 0, 0, 0);

        for (int j = 0; j < step_y; ++j){
          for (int i = 0; i < step_x; ++i){
              uchar4 p = tex2D<uchar4>(tex, (x * step_x + i + 0.5f) / w, (y * step_y + j + 0.5f) /
h);
            blockAcc.x += p.x;
            blockAcc.y += p.y;
            blockAcc.z += p.z;
            blockAcc.w += p.w;
          }
        }

        out[y * wn + x] = make_uchar4(blockAcc.x / size, blockAcc.y / size, blockAcc.z / size,
blockAcc.w / size);
      }
    }
}
```

Результаты

Замеры времени работы ядер с различными конфигурациями и размерами изображений (512x288, 1200x800, 2048×1152). Гарантируется что размеры выходного изображения соответственно кратны размерам исходного изображения (Для примера, в 2 раза меньше). А также приведён замер работы аналогичной программы без использования технологии CUDA - на CPU.

1. **Конфигурация <<< dim3(2, 2), dim3(4, 4) >>>**

Размер исходного изображения	Время (в мс)
512x288	0.103680
1200x800	12.126496
2048x1152	28.764481

2. **Конфигурация <<< dim3(8, 8), dim3(16, 16) >>>**

Размер исходного изображения	Время (в мс)
512x288	1.877792
1200x800	0.202880
2048x1152	0.374240

3. **Конфигурация <<< dim3(16, 16), dim3(32, 32) >>>**

Размер исходного изображения	Время (в мс)
512x288	0.156032
1200x800	0.199328
2048x1152	0.308864

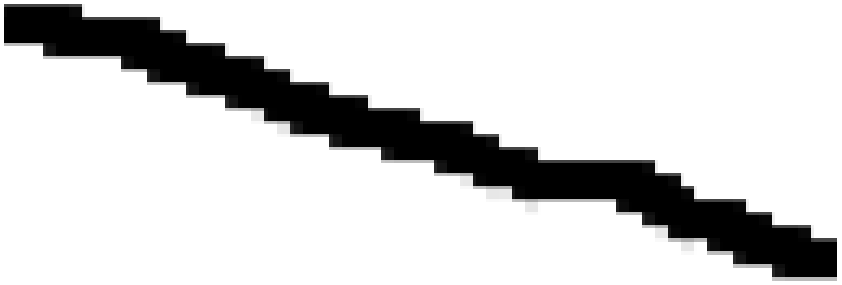
4. **CPU**

Размер исходного изображения	Время (в мс)
512x288	1,387
1200x800	8,916
2048x1152	23,64

Картинки:

Исходное изображение 512x288:

Выходное изображение 256x144:



Исходное изображение 1200x800:

Выходное изображение 600x400:

Исходное изображение 2048x1152:

Выходное изображение 1024x576:

Выводы

В ходе выполнения лабораторной работы я научиться использовать GPU для обработки изображений, а также работать с текстурной памятью и двухмерной сеткой потоков.

Этот алгоритм может применяться в области компьютерной графики и рендеринга (сглаживание лесенки на границах объектов), в области видеоигр (позволяет уменьшить эффект пикселизации в динамческих сценах), в области обработки изображений (гладкое уменьшение разрешения без потери качества) и т.д.

Программировать было несложно. Единственное, надо было разобраться с двумерной сеткой потоков и понять сам алгоритм SSAA.

Сравнивая время работы ядер с различными конфигурациями и размерами исходных изображений, а также аналогичной программы без использования технологии CUDA - на CPU, можно заметить, что конфигурации с достаточным количеством потоков всегда выигрывают по скорости у CPU, причём значительно. Однако конфигурация <<< dim3(2, 2), dim3(4, 4) >>> вполне сравнима с CPU, возможно, дело во встроенном в мой CPU графическом процессоре Radeon Graphics, он слабый и не сравнится с конфигурациями с большим числом потоков, но всё же. Из всех тестов можно сделать вывод, что при конфигурациях с достаточным количество потоков GPU всегда будет выигрывать по скорости у CPU.