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.

Студент: А. А. Садаков
Группа: 8О-406Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2022

Условие

Цель работы: Научиться использовать GPU для обработки изображений. Использование *текстурной памяти* и *двухмерной сетки потоков*.

Вариант 4: SSAA.

Программное и аппаратное обеспечение

ГРАФИЧЕСКИЙ ПРОЦЕССОР
Имя устройства: NVIDIA GeForce GTX 1650
Compute capability: 7.5
Размер графической памяти: 4242604032
Размер разделяемой памяти: 49152
Размер константной памяти: 65536
Максимальное количество регистров на блок: 65536
Максимальное количество потоков на блок: 1024
Количество мультипроцессоров: 14

ПРОЦЕССОР
Имя устройства: Intel Core I5-10300H
Архитектура: Comet Lake-H
Количество ядер (потоков): 4(8)
Базовая (максимальная) частота: 2,5(4,5)ГГц
Кеш 1-го уровня: 64Кб (на ядро)
Кеш 2-го уровня: 256Кб (на ядро)
Кеш 3-го уровня: 6Мб (всего)

ОПЕРАТИВНАЯ ПАМЯТЬ
Объём: 8Гб
Тип: DDR4
Частота:2933МГц

ЖЁСТКИЙ ДИСК
Имя устройства: Intel SSDPEKNW512G8L
Тип диска: SSD
Объём памяти: 512Гб
Скорость чтения/записи: 1500/1000 Мб/с

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

OS: Windows 10,Подсистема Ubuntu 20.04
IDE: Visual Studio Code
Компилятор: nvcc

Метод решения

Для преобразования изображения алгоритмом SSAA необходимо разбить исходное изображение на блоки $n \times m$ и найти среднюю арифметическую сумму всех цветов в этих блоках.

Описание программы

Для преобразования изображения алгоритмом SSAA необходимо разбить исходное изображение на $n \times m$ блоков, где n — соотношение старой ширины к новой, m — старой высоты к новой. Внутри каждого блока происходит суммирование всех значений и деление результата на площадь блока. Таким образом находится средний цвет. Для хранения данных изображения используется текстурная память.

Вызов $SSAAkernel$ происходит с количеством нитей на блок — 1024 и количеством блоков — 1024. В самом $SSAAkernel$ я вычисляю общий индекс исполняемой нити idx который и будет индексом в массиве при условии $idx < block\_count$, где $block\_count$ — количество блоков, на которое поделено исходное изображение, смещение $offset$, которое позволяет найти следующий обрабатываемый блок для нити.

Код функции:

```
1  __global__ void SSAAkernel (uint32_t old_w, uint32_t old_h, uint32_t
       new_w, uint32_t new_h, uint32_t *new_buff) {
2      uint32_t idx = blockIdx.x * blockDim.x + threadIdx.x;
3      uint32_t offset = blockDim.x * gridDim.x;
4
5      uint32_t block_w = old_w / new_w, block_h = old_h / new_h;
6      uint32_t block_size = block_w * block_h, block_count = new_h *
           new_w;
7      while (idx < block_count) {
8          uint32_t i0 = idx / new_w, j0 = idx % new_w;
9          uint32_t r = 0, g = 0, b = 0, a = 0;
10         for (uint32_t i1 = 0; i1 < block_h; ++i1) {
11             for (uint32_t j1 = 0; j1 < block_w; ++j1) {
```

```
                    uint32_t color = tex2D(img_tex, j1 + j0 * block_w, i1 + i0 *
                        block_h);
                    r += (color >> 24) & 255;
                    g += (color >> 16) & 255;
                    b += (color >> 8) & 255;
                    a += color & 255;
                }
            }
            r = (r / block_size) << 24;
            g = (g / block_size) << 16;
            b = (b / block_size) << 8;
            a = a / block_size;
            new_buff[i0 * new_w + j0] = r + g + b + a;
            idx += offset;
        }
}
```

Результаты

1. Замеры времени работы ядер с различными конфигурациями (время указано в микросекундах).

Конфигурации ядра \ Размер изображений	16x16	256x256	1280x720	1920x1080	3840x2160
1, 32	13	213	3186	7154	28603
32, 32	8	16	150	321	1263
32, 256	8	11	52	97	357
256, 256	8	13	50	92	282
1024, 1024	63	64	107	144	352

2. Сравнение с CPU

Размер входных данных	16x16	256x256	1280x720	1920x1080	3840x2160
GPU(256, 256)	8	13	50	92	282
CPU	1	353	5126	10985	43778

3. Примеры изображений

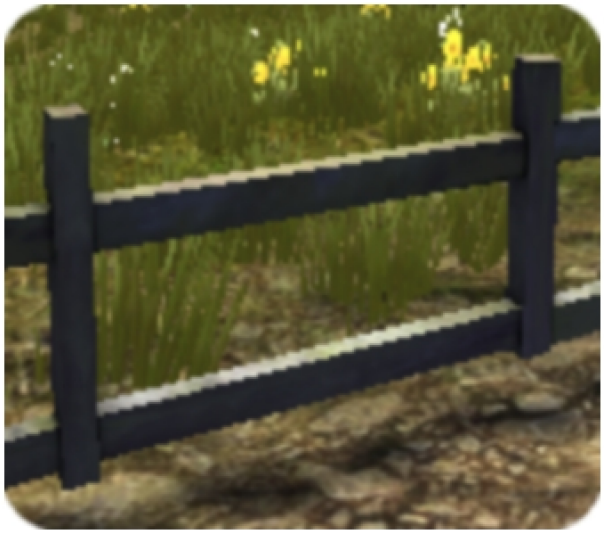

Рис. 1: До SSAA

Рис. 2: После SSAA

Рис. 3: До SSAA

Рис. 4: После SSAA

Выводы

Алгоритм сглаживания SSAA является как самым простым, так и одним из самых эффективных алгоритмов сглаживания. Но, к сожалению, он сильно снижает производительность, поэтому его использование рекомендуется для сильных ПК. Для слабых же есть и другие не менее результативные алгоритмы.