МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Лабораторная работа №1
по курсу «Технологии параллельного программирования**»**

Обработка изображений на GPU. Фильтры.

Выполнил: Филиппов Владимир Михайлович
Группа: М8О-410Б-22
Преподаватели: А.Ю. Морозов,
Е.Е. Заяц

Москва, *2025*

Условие

Кратко описывается задача:

1. Научиться использовать GPU для обработки изображений. Использование текстурной памяти и двухмерной сетки потоков.
2. SSAA.

Программное и аппаратное обеспечение

GPU.

• Compute Capability: 8.9
• Графическая память (Global memory): 8 GB GDDR6 (8188 MiB — видно в nvidia-
smi)
• Shared memory per SM (мультипроцессор): до 100 KB (может конфигурироваться:
64 KB shared + 32 KB L1 или наоборот)
• Shared memory per block: до 48 KB
• Constant memory: 64 KB (аппаратно выделено)
• Регистры на потоковый мультипроцессор (SM): 65,536
• Максимум регистров на блок: 32K (зависит от конфигурации)
• Максимальное число потоков на блок: 1024
• Максимальное число блоков в сетке (grid): 2^31 - 1 по X, 2^16 - 1 по Y и Z
• Количество мультипроцессоров (SM): 24 (у RTX 4060)
• Макс. потоков на SM: 2048
• Итого потоков на GPU: 24 × 2048 = 49,152 одновременно резидентных

CPU.

• Архитектура. x86_64
• Модель. AMD Ryzen 5 3600 6-Core Processor
• Количество физических ядер. 6
• Потоков. 12 (2 потока на ядро)
• Частота (BogoMIPS). 7200.05
• Кэш L1 (данные/инструкции). 192 KiB / 192 KiB
• Кэш L2. 3 MiB
• Кэш L3. 16 MiB

RAM.

• Общий объём. 15 GiB
• Swap. 4 GiB (свободен)

SSD.• Объем. 1 TB
• Использовано. 156 GB

ПО.

• OS. Windows 11 + WSL 2.0.
• IDE. Visual studio code.
• NVCC. 11.5 (Build V11.5.119)

Метод решения

Для каждой нити вычисляем пиксель, на котором она будет отрабатывать. Далее, на заранее определенном блоке размера m*n вычисляем средние значение для каждой компоненты цвета. Сохраняем полученное значение в выходной массив.

Описание программы

Вся программа находится в файле main.cu. Присутствует одно ядро и одна дополнительная функция (calc_average), исполняемая на GPU. Именно она занимается вычислением среднего значения пикселей на заданной области. В самой функции ядра вычисляем каждый пиксель нового изображения по алгоритму.

Результаты

В рамках тестирования я проверял свой алгоритм на вот таком изображении. На рисунке 1 вы можете увидеть картинку до применения фильтра, на рисунке 2 после. Наглядно видим, как изображение упало в качестве, линии стали более размытыми. Это связано как раз таки с усредненным значением пикселя.

Рисунок 1. Изображение до применения SSAA.

Рисунок 2. Изображение после применения SSAA.

Кроме того, как и для ЛР 1, я создал heatmap для наглядного отображения зависимости времени исполнения от количества потоков и блоков. Тут везде имеются ввиду «двухмерные» потоки и блоки. Видим, что чем больше блоков и потоков в них, тем быстрее происходит обработка. Резкий скачок наблюдается в тот момент, когда начинают быть задействованы все мультипроцессоры. Тестирование происходило на изображении размером 8000x6000 в выходное изображение размером 300x100. На CPU такое же преобразование требует около 400 мс. В таблице 1 вы можете увидеть сравнительную характеристику времени исполнения на CPU и на GPU

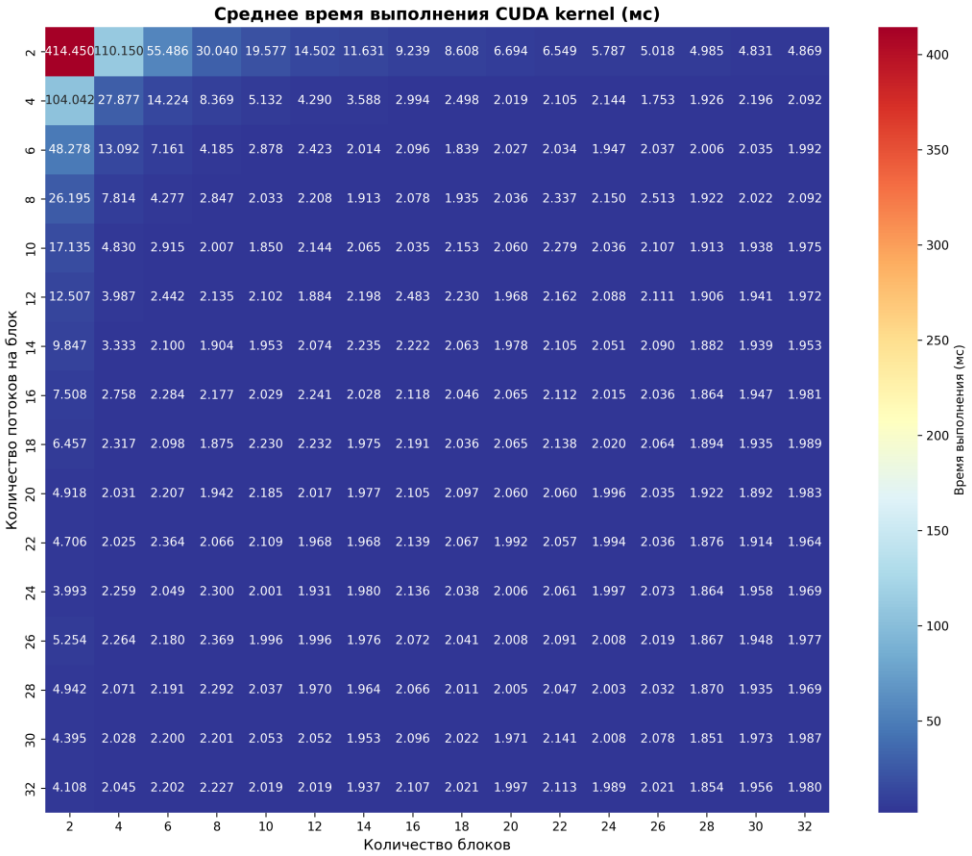

Рисунок 3. Heatmap времени исполнения в зависимости от числа потоков и блоков.

	CPU (мс)	GPU(мс)
16000x12000	1583.97	8.379032
8000x6000	397.381	1.792785
800x600	5.492	0.604800

Таблица 1. Сравнительная характеристика времени исполнения на GPU и CPU.

Выводы

Данный алгоритм используются, например, в компьютерной графике, когда необходимо избавиться от слишком пиксельных ломанных линий (так называемых лесенок). С его помощью такие фигуры становятся более плавными.

В реализации этот алгоритм крайне прост, однако само его исполнение является дорогостоящим. Так, для каждого пикселя нам необходимо посчитать среднее пикселей в блоке вокруг него, то есть O(h * w * h_block * w_block).