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: Корнев Максим Сергеевич
Группа: М8О-412Б-22
Преподаватели: А.Ю. Морозов,
Е.Е. Заяц

Москва, 2025

Условие

1. **Цель работы**. Научиться использовать GPU для обработки изображений. Использование текстурной памяти и двухмерной сетки потоков.
2. **Вариант 5.** Выделение контуров. Метод Робертса.
   Входные данные. На первой строке задается путь к исходному изображению, на второй, путь к конечному изображению. $w * h \leq 10^8$.

Пример:

Входной файл	hex: in.data	hex: out.data
in.data out.data	03000000 03000000 01020300 04050600 07080900 09080700 06050400 03020100 00000000 14141400 00000000	03000000 03000000 04040400 03030300 07070700 0C0C0C00 12121200 03030300 1C1C1C00 1C1C1C00 00000000
in.data out.data	03000000 03000000 00000000 00000000 00000000 00000000 80808000 00000000 00000000 00000000 00000000	03000000 03000000 80808000 80808000 00000000 80808000 80808000 00000000 00000000 00000000 00000000

Программное и аппаратное обеспечение

Аппаратное обеспечение

Графический процессор (GPU):

- Модель: NVIDIA Tesla T4
- Архитектура: Turing
- Compute Capability: **7.5**
- Графическая память: **16 ГБ GDDR6**
- Пропускная способность памяти: ~320 ГБ/с
- Количество мультипроцессоров (SM): **40**
- Максимальное число потоков на один SM: 1024
- Максимальное число потоков в блоке: 1024
- Максимальное число регистров на блок: 65 536 (по 32 бита)
- Разделяемая память на блок: до **64 КБ**
- Константная память: **64 КБ**

Процессор (CPU):

- Виртуальная машина Colab: 1 × Intel Xeon (Google Cloud)
- Частота: ~2.0–2.2 ГГц
- Количество ядер: обычно 2 доступных потока

Оперативная память (RAM):

- В Google Colab стандартно: **12–13 ГБ** (доступные пользователю)

Жёсткий диск (HDD/SSD):

- Виртуальный диск Colab: ~70–80 ГБ (SSD Google Cloud)

Программное обеспечение

- Операционная система: **Linux (Ubuntu 20.04 LTS, Google Colab среда)**
- Драйвер CUDA: **550.54.15**
- CUDA Toolkit: **12.5** (с nvcc 12.5.82, поддержка до Compute Capability 9.0)

- Язык программирования: **C++ (CUDA C)**
- Компилятор: nvcc (NVIDIA CUDA Compiler Driver)
- IDE / среда разработки: **Google Colaboratory (Jupyter Notebook web-IDE)**
- Дополнительно: доступ к стандартным библиотекам C (cstdio, cstdlib, cmath), а также к CUDA Runtime API

Метод решения

Программа выполняет выделение контуров на изображении методом Робертса с использованием вычислений на GPU. На первом этапе из стандартного ввода считываются пути к входному и выходному файлам, после чего входное изображение в бинарном формате загружается в оперативную память. Затем данные копируются в память видеокарты и связываются с текстурным объектом, настроенным в режиме Clamp для корректной обработки граничных пикселей. На GPU запускается kernel-функция с двухмерной сеткой потоков, где каждый поток обрабатывает часть изображения по схеме Grid-Stride Loop. Для каждого пикселя вычисляется яркость, определяется градиент по диагоналям согласно оператору Робертса, находится его модуль и результат ограничивается значением 255, чтобы сохранить корректный диапазон яркости. После завершения вычислений данные копируются обратно в память CPU и сохраняются в выходной файл. Для измерения производительности используется замер времени с помощью CUDA-событий, а все операции с GPU сопровождаются проверкой ошибок и последующим освобождением ресурсов.

Описание программы

Программа представлена одним файлом **lab2.cu**, в котором реализована полная логика выполнения оператора Робертса на GPU с использованием технологии CUDA. Основным типом данных служит структура uchar4, описывающая пиксель с четырьмя 8-битными компонентами (R, G, B, A). Для контроля корректности выполнения функций CUDA используется макрос CUDA_CHECK, который проверяет возвращаемый код ошибок и при необходимости выводит сообщение с указанием места сбоя и завершает программу.

В функции main() осуществляется чтение путей к входному и выходному файлам из стандартного ввода. Далее открывается входной бинарный файл, из которого считываются ширина и высота изображения, а затем массив пикселей загружается в вектор std::vector<uchar4>. После загрузки данных происходит подготовка ресурсов для вычислений на GPU: выделяется CUDA-массив под исходное изображение, выполняется копирование данных в него с помощью cudaMemcpy2DToArray, создается текстурный объект с режимом Clamp для корректной выборки на границах изображения и выделяется буфер для хранения выходных данных.

Основные вычисления выполняются в ядре roberts_kernel, которое запускается на GPU в виде двумерной сетки потоков. Каждый поток обрабатывает участок изображения, считывает значения четырёх соседних пикселей из текстуры и преобразует их в яркость по формуле Y = 0.299R + 0.587G + 0.114B. Далее вычисляются градиенты по диагональным направлениям в соответствии с оператором

Робертса, определяется модуль градиента и результат ограничивается диапазоном 0–255. Полученное значение записывается во все цветовые каналы выходного пикселя с сохранением исходного альфа-компонента.

Для оценки производительности используется замер времени с помощью CUDA-событий — в момент запуска и завершения ядра фиксируются значения времени, после чего вычисляется продолжительность работы GPU-вычислений в миллисекундах. После завершения обработки данные копируются обратно в оперативную память, формируется бинарный выходной файл, содержащий ширину, высоту и массив обработанных пикселей. Все выделенные ресурсы видеокарты, включая CUDA-массив, текстурный объект и буфер вывода, корректно освобождаются перед завершением программы.

Результаты

1. Сравнение результатов работы программы для различных конфигураций. Результаты отображены в таблице 1.

Конфигурация blocks, threads	**Потоки**	**1920 * 1048 (мс)**	**2560 * 2048 (мс)**
dim3(2,2), dim3(8,8)	256	9.006912	22.631968
dim3(4,4), dim3(8,8)	1024	2.335040	5.720832
dim3(16,16), dim3(16,16)	**65536**	**0.237536**	**0.448256**
dim3(32,32), dim3(32,32)	1048576	0.284736	0.453056
dim3(64,64), dim3(32,32)	4 194 304	0.379264	0.627808
dim3(128,128), dim3(32,32)	16 777 216	0.608064	0.941856
dim3(256,256), dim3(32,32)	67 108 864	1.368256	1.773504
dim3(512,512), dim3(32,32)	268 436 456	4.412064	4.958304
dim3(1024,1024), dim3(32,32)	1 073 741 824	16.297504	17.114944

Таблица 1 – таблица времени работы с различными конфигурациями

2. Для сравнения производительности была реализована версия программы на CPU. Также было проведено сравнение GPU с CPU на изображения разного разрешения. Результаты сравнения приведены в Таблице 2.

Разрешение	**GPU, мс**	**CPU, мс**
512 * 512	0.132896	3.042382

1920 * 1020	0.222128	23.336076
2560 * 2048	0.410592	54.552888

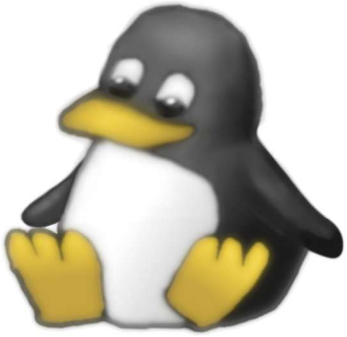
Таблица 2 – сравнение времени работы GPU и CPU

То есть чем больше входные данные, тем более целесообразно использовать GPU.

3. Примеры создания контура программой на GPU.

512 * 512

До



После

1920 * 1020

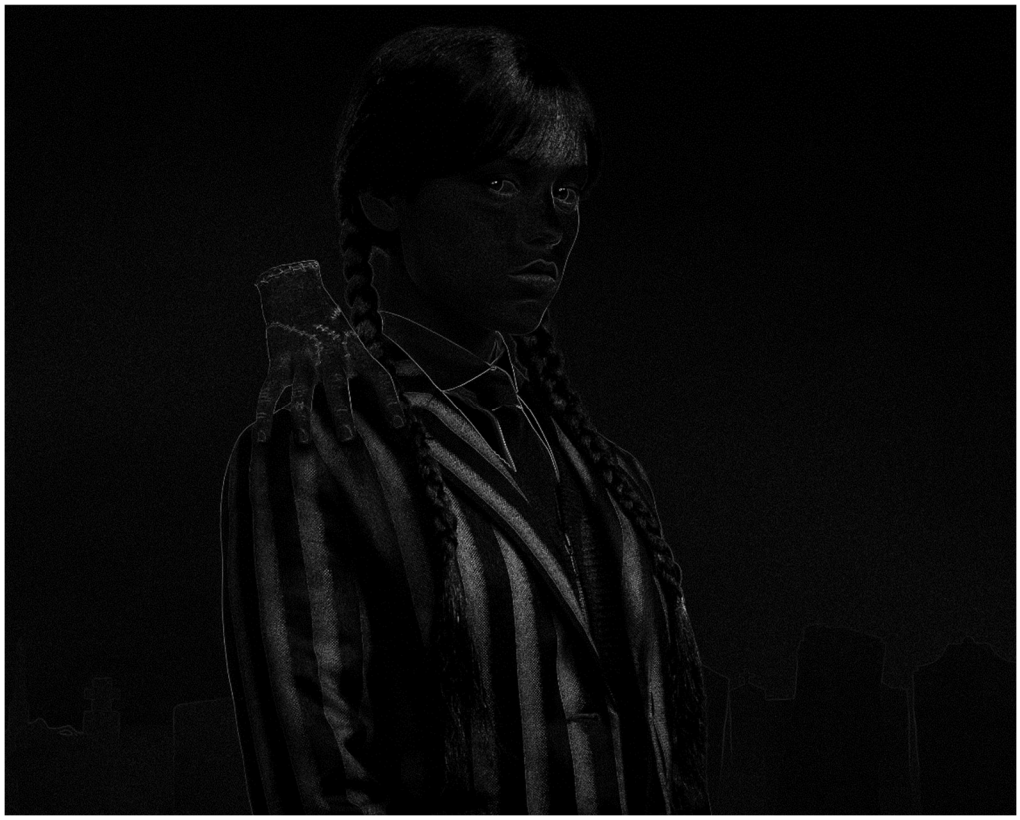
До

После

2560 * 2048

До

После



Выводы

В ходе работы был проведён анализ производительности алгоритма выделения контуров методом Робертса при различных конфигурациях сетки потоков на GPU. По полученным результатам видно, что время выполнения существенно зависит от выбранных параметров blocks и threads. При небольшом количестве потоков (например, dim3(2,2), dim3(8,8)) наблюдается наибольшее время работы — 9,0 мс для изображения 1920×1048 и 22,6 мс для 2560×2048. С увеличением количества потоков время резко сокращается, и наилучший результат достигается при конфигурации dim3(16,16), dim3(16,16), где время обработки составило 0,237 мс и 0,448 мс соответственно.

При дальнейшем росте числа блоков и потоков производительность начинает снижаться из-за увеличения накладных расходов на синхронизацию и управление большим количеством потоков. Таким образом, оптимальной конфигурацией для данной задачи является dim3(16,16), dim3(16,16), обеспечивающая наилучший баланс между вычислительной загрузкой и эффективным использованием памяти.