МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №2
по курсу «Программирование графических процессоров»

Обработка изображений на GPU. Фильтры.

Выполнила: Е.И. Усачева
Группа: 8О-407Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2020

Условие

1. **Цель работы:** Научиться использовать GPU для обработки изображений. Использование текстурной памяти.
2. **Вариант задания:** 6. Выделение контуров. Метод Превитта.

Программное и аппаратное обеспечение

Лабораторная работа выполнялась в среде Google Collab с использованием встроенного аппаратного ускорителя GPU в облаке.

Свойства GPU:

Название:	Tesla T4
Compute capability:	7.5
Глобальная память:	15812263936
Разделяемая память (на блок):	49152
Константная память:	65536
Количество регистров на блок:	65536
Максимальное количество блоков:	2147483647*65535*65535
Максимальное количество нитей:	1024*1024*64
Количество мультипроцессоров:	40

nvcc: NVIDIA (R) Cuda compiler driver, release 9.2, V9.2.88

Процессор:	1,8 GHz 2−ядерный процессор Intel Core i5
Оперативная память:	8 ГБ 1600 MHz DDR3
Жесткий диск:	SSD-накопитель PCIe
OS:	macOS Catalina v10.15.4

Метод решения

Так как изображение это текстура и достаточно большой массив данных, будем работать с текстурной памятью GPU.

Выделение контуров. Поиск контуров - вычисление длины градиента яркости в точке. Градиент направлен перпендикулярно контуру.

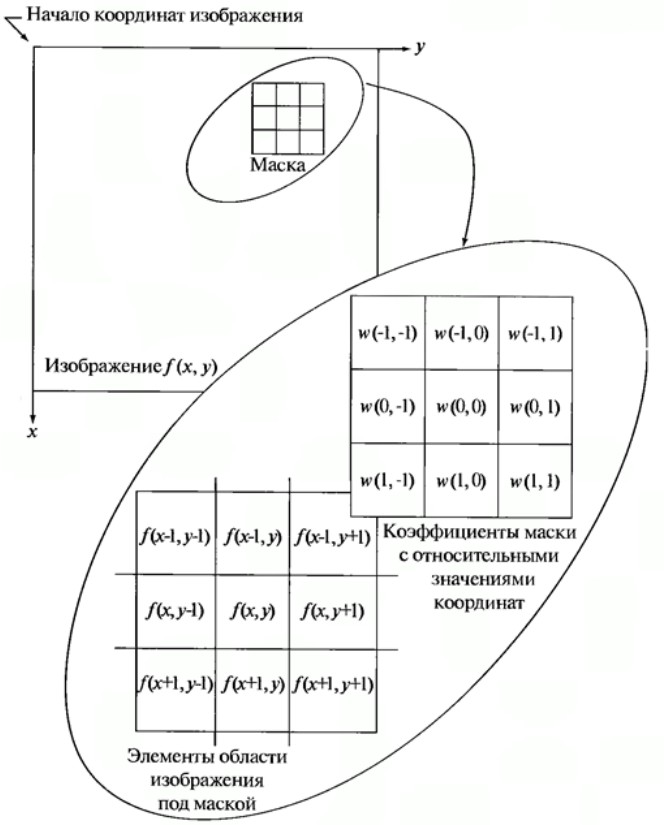
Метод Превитта. Процесс основан на простом перемещении маски 3x3 фильтра (ядро свертки для подсчета G_x и G_y) от точки к точке изображения;

• M_x - ядро свертки для G_x

- M_y - ядро свертки для G_y

$$M_x = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}, \quad M_y = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

- W - окно просмотра



- G_x и G_y - градиенты по x и y осям соответственно

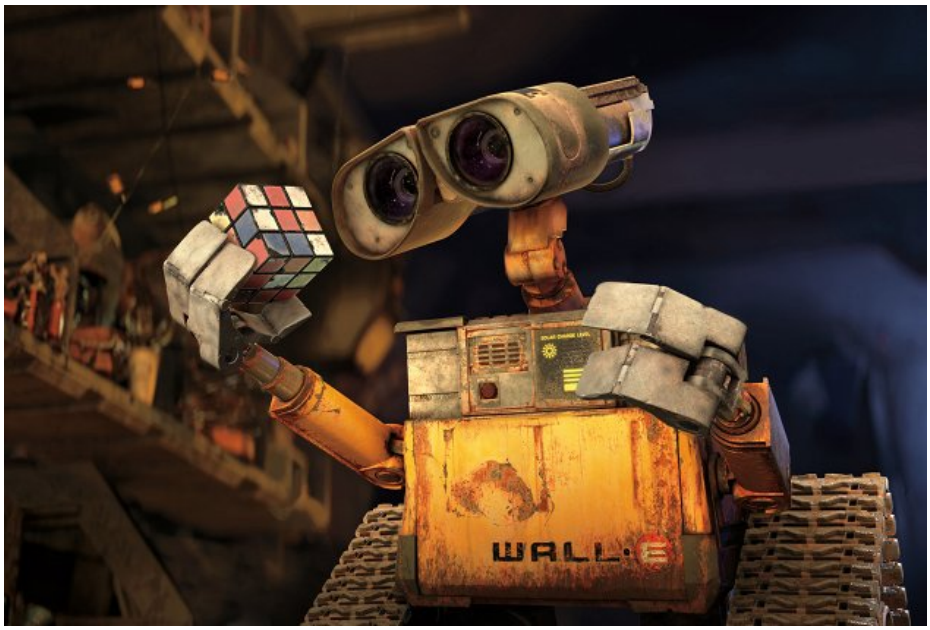
$$G_x = w_{13} + w_{23} + w_{33} - w_{11} - w_{21} - w_{31}$$
$$G_y = w_{31} + w_{32} + w_{33} - w_{11} - w_{12} - w_{13}$$

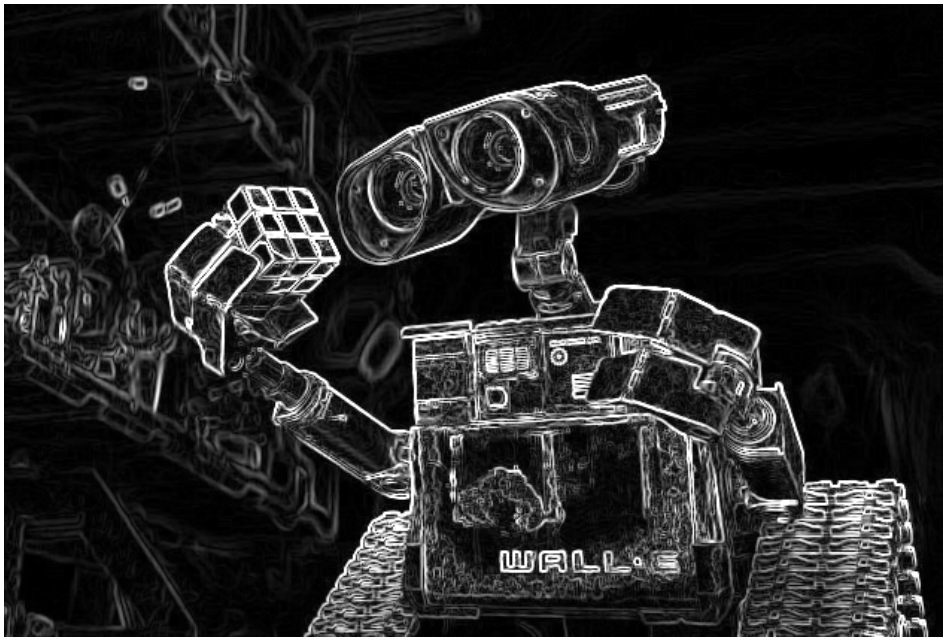
Описание программы

Программа принимает на вход бинарный файл, полученный предварительно путем конвертирования изображения с помощью конвертера, взятого из файлов преподавателя.
Для работы с текстурной памятью создаем т.н. глобальную текстурную ссылку.
С помощью функции *cudaMallocArray()* выделяем память под массив данных на устройстве. Эта функция требует описание формата созданного массива, которое создается функцией *cudaCreateChannelDesc()*. В результате будет выделена память для массива размером *w*h*, содержащего векторы *uchar4*, состоящие из четырех 8-ми битных знаковых целых чисел. С помощью функции *cudaMemcpyToArray()* копируем данные в выделенную память на устройство.
Затем, для обработки пограничных условий используется Clamp-адресация.
Далее «привязываем» данные *cudaArray* с помощью функции *cudaBindTextureToArray()*.
Вызываем ядро. Рассчитываем абсолютные адреса потоков и смещения для *x* и *y* измерений. Далее начинаем обход исходного массива слева-направо и сверху-вниз. Получаем для каждого элемента массива (по сути пикселя изображения) значения интенсивности трех каналов цвета, преобразовываем в оттенок серого. Полученное значение умножаем на соответствующее значение в ядре свертки и суммируем с остальными. Таким образом получаем градиент по каждой оси. Затем считаем длину градиента и сохраняем результат в выходной массив.
Далее копируем получившийся выходной массив в память хоста и очищаем память на устройстве.

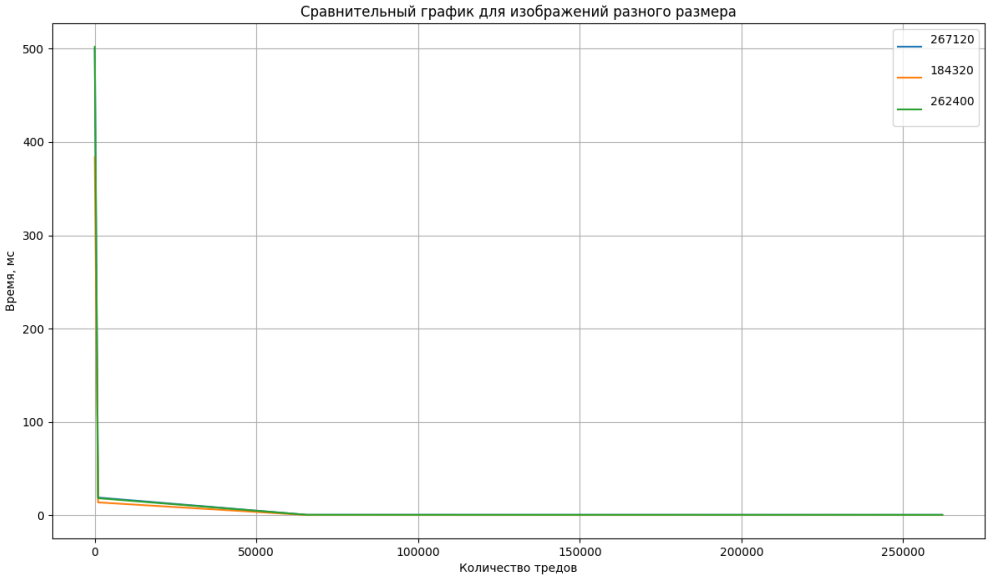
Результаты

Ниже приведены три результата работы алгоритма:

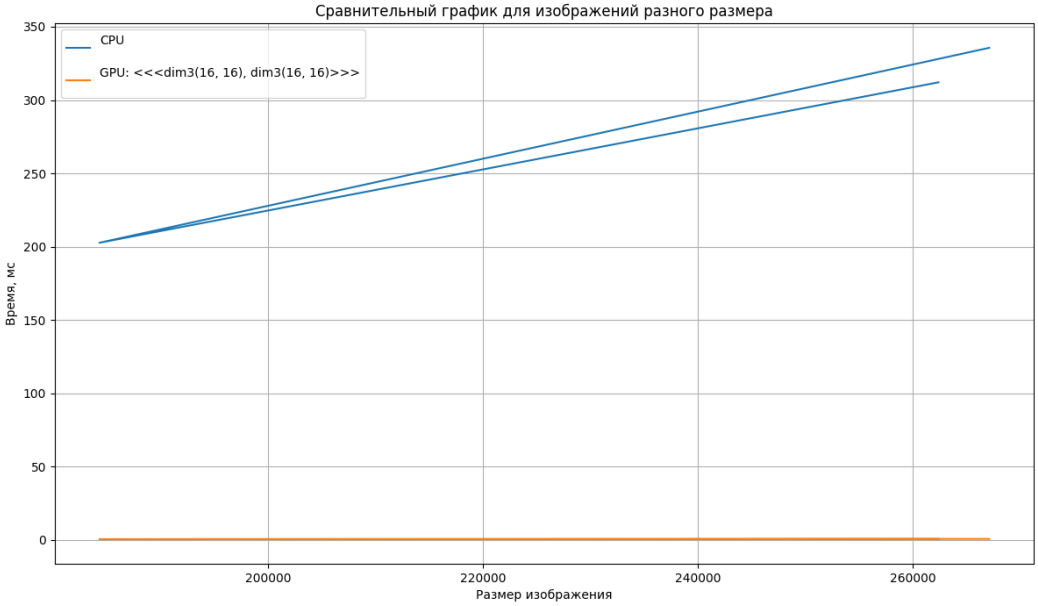



WALL·E

На графике ниже изображена зависимость время исполнения расчетов от различной конфигурации ядра (кол-во тредов). Видно, что после определенного значения зависимость становится константной. Однако, мой график мало информативен хотя бы потому что мне не удалось обработать большие изображения для проведения сравнительного анализа.



Посмотрим на разницу работы графического процессора с конфигурацией ядра *<<<dim3(16, 16), dim3(16, 16)>>>* и работы CPU для тех же примеров:



Выводы

В данной лабораторной работе я познакомилась с различными способами фильтрации изображений и реализовала один из них: метод Превитта.
Алгоритм, реализованный на GPU, показал хорошую скорость работы в сравнении с алгоритмом, реализованном на CPU. Также стоит отметить, что мне не удалось поэкспериментировать с изображениями большой размерности, однако даже на относительно небольших семплах отчетливо видно преимущество GPU над CPU.
Выделение контуров изображения может быть очень полезно для задач машинного обучения и компьютерного зрения. По этой причине я считаю данный метод выделения контуров, и ему подобные, необходимыми навыками в современном мире.