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: И. П. Моисеенков
Группа: М8О-408Б-19
Преподаватель: А.Ю. Морозов

Москва, 2022

Условие

Цель работы: научиться использовать GPU для обработки изображений. Использование текстурной памяти и двумерной сетки потоков.

Формат изображений: изображение является бинарным файлом со следующей структурой:

- в первых восьми байтах записывается размер изображений
- далее построчно значения пикселей r, g, b, a.

Вариант 3. Билинейная интерполяция:

Входные данные. На первой строке задается путь к исходному изображению, на второй - путь к конечному изображению. На следующей строке два числа - новые размеры изображения.

Выходные данные. Необходимо записать в выходной файл изображение.

Программное и аппаратное обеспечение

В качестве графического процессора использую видеокарту Nvidia GeForce GT 545, установленную на сервере преподавателя.

```
Compute capability      : 2.1
Name                    : GeForce GT 545
Total Global Memory     : 3150381056
Shared memory per block : 49152
Registers per block     : 32768
Warp size               : 32
Max threads per block   : (1024, 1024, 64)
Max block   : (65535, 65535, 65535)
Total constant memory   : 65536
Multiprocessors count   : 3
```

В качестве редактора кода использовался Visual Studio Code.

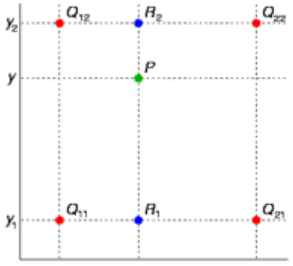
Метод решения

Суть метода билинейной интерполяции заключается в следующем: для каждого «нового» пикселя нужно найти четыре «старых» пикселя, окружающих его. Значение «нового» пикселя вычисляется по четырем окружающим его «старым» пикселям пропорционально расстоянию до них. Под «старыми» и «новыми» пикселями подразумеваются пиксели исходного и преобразованного (увеличенного/уменьшенного) изображения.

Более наглядное описание метода представлено на изображении:

$$Q_{11} = (x_1, y_1),\ Q_{12} = (x_1, y_2)$$
$$Q_{21} = (x_2, y_1),\ Q_{22} = (x_2, y_2)$$

$$R_1 = (x, y_1), \quad R_2 = (x, y_2)$$



$$f(R_1) = \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$

Аналогично $f(R_2)$

$$f(P) = \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2)$$

На видеокарте можем проходить по пикселям нового изображения и вычислять их значения. Исходное изображение будет находиться в текстурной памяти.

Описание программы

Исходное изображение помещается в текстурную память с помощью функции cudaMallocArray. Так оно будет доступно всем потокам.

Создание нового изображения будет производиться параллельно на графическом процессоре в функции kernel. В ней реализована вся необходимая математика для вычисления значения нового пикселя по четырем окружающим. На вход этой функции подается местоположение новой картинки, а также исходные и результирующие размеры изображения.

Полученное изображение записывается в файл в бинарном виде.

Результаты

Рассмотрим время работы программы на различных тестах при различных размерах сетки (и без использования графического процессора вообще). Будем замерять непосредственно время работы алгоритма. В качестве тестов будем увеличивать картинку до различных размеров. Размер исходной картинки 840x908 пикселей. Результаты приведены в таблице ниже.

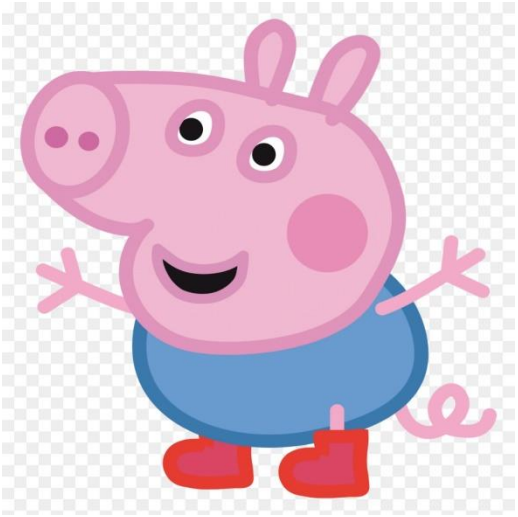
Размер сетки ядра	1к x 1к пикселей, мс	5к x 5к пикселей, мс	10к x 10к пикселей, мс
CPU	5.11	124.19	537.01
<<<(1, 32), (1, 32)>>>	5.15	118.65	465.21
<<<(32, 32), (1, 32)>>>	4.04	99.00	373.03
<<<(1, 128), (1, 128)>>>	4.07	82.27	321.28
<<<(1, 256), (1, 32)>>>	4.08	82.28	321.28

В этом алгоритме уже сильнее заметна разница между CPU и GPU - на графическом процессоре программа выполняется быстрее за счет параллельных вычислений.

Пример работы программы. Имеется исходное изображение:

С помощью билинейной интерполяции уменьшили его размер:

И немного расширили исходное изображение в ширину:



Выводы

Вторая лабораторная работа была посвящена обработке изображений на графическом процессоре. Эта работа показалась мне гораздо интереснее предыдущей.

Мне было предложено реализовать метод билинейной интерполяции для изменения размера изображения. Казалось бы, в этом методе заложена максимально простая математика. Но интерес в том, что эту простую математику мы можем еще и ускорить за счет параллельных вычислений на GPU. Это очень круто. Поэкспериментировав, я увидел, что на CPU последовательный алгоритм работает медленнее.

Работая над задачей, я более подробно изучил различные виды памяти и попользовался некоторыми из них. Также я познакомился с двумерной сеткой потоков.