МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Факультет «Информационные технологии и прикладная математика»
Кафедра «Вычислительная математика и программирование»

Лабораторная работа №2
по курсу «Параллельная обработка данных»

Обработка изображений на GPU. Фильтры.

Выполнил: А.В. Скворцов
Группа: 8О-408Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2019

Условие

Цель работы: научиться использовать GPU для обработки изображений, научиться использовать текстурную память.
Формат изображения: в первых восьми байтах записывается размер изображения, далее построчно все значения пикселей.
Вариант 4: Необходимо реализовать избыточную выборку сглаживания. Исходное изображение представляет собой “экранный буфер”, на выходе должно быть сглаженное изображение, полученное уменьшением исходного.

Программное и аппаратное обеспечение

- GPU: Geforce 940MX
  - Compute capability : 5.0
  - Total Global Memory : 2147483648
  - Shared memory per block : 49152
  - Registers per block : 65536
  - Max threads per block : (1024, 1024, 64)
  - Max block : (2147483647, 65535, 65535)
  - Total constant memory : 65536
  - Multiprocessors count : 3
- CPU: Intel Core i5-6200U 2.30GHz
- RAM: 4GB
- Software: Windows 10, Visual Studio Code, nvcc

Метод решения

Так как сегодня мониторы состоят из множества мелких прямоугольных пикселей, то по-настоящему гладким прямыми получаются только горизонтальные и вертикальные линии. Напротив все прямые под углом получаются зубчатыми. Для устранения такого эффекта разработано множество алгоритмов сглаживания, один из них — ssaa(суперсэмплинг). Без суперсэмплинга пиксель закрашивается, только если через его центр проходит геометрический примитив. Включив эту технологию, разрешение нашей картинки увеличивается, и теперь одному исходному пикселю ставится в соответствие несколько новых субпикселей. После растеризации изображения в новом экранном буфере, итоговый цвет пискеля определяется как усредненный цвет всех соответствующих ему субпикселей.

$$result = \frac{sample_0 + sample_1 + \ldots + sample_{n-1}}{n} = \frac{\sum_{i=0}^{n-1} sample + i}{n}$$

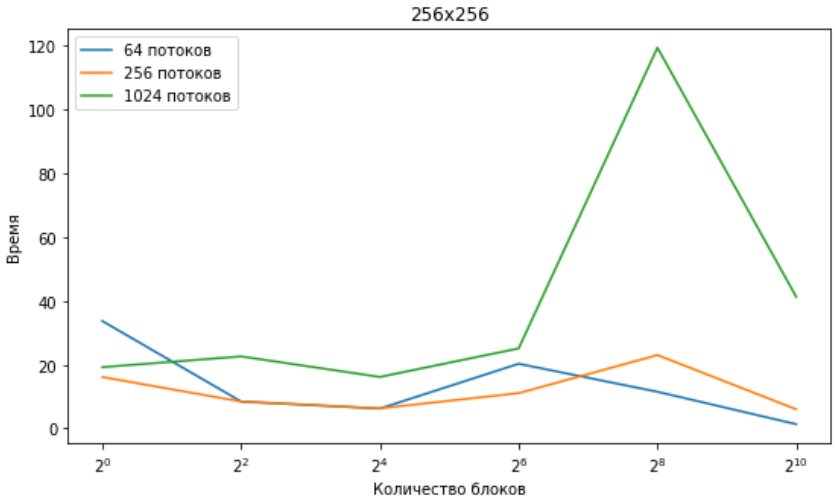
Описание программы

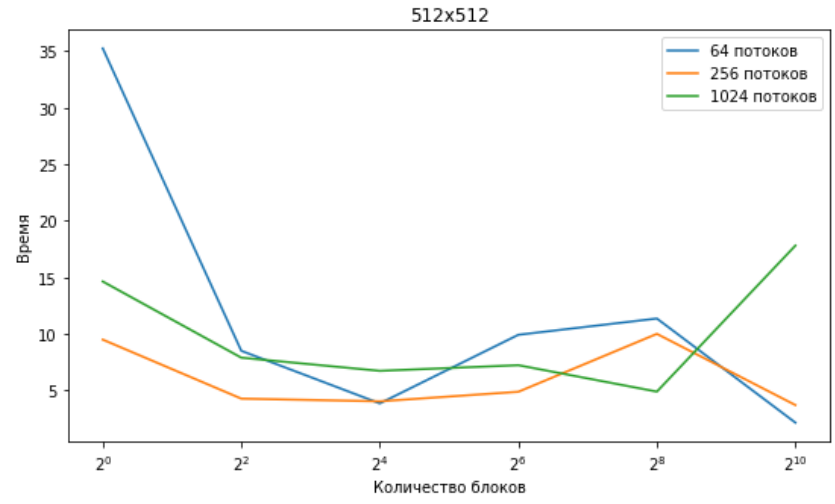
- void main() - функция, выполняемая хостом: считывание исходных данных, подготовка текстуры, копирование с и на gpu, запуск ядра, вывод результатов.

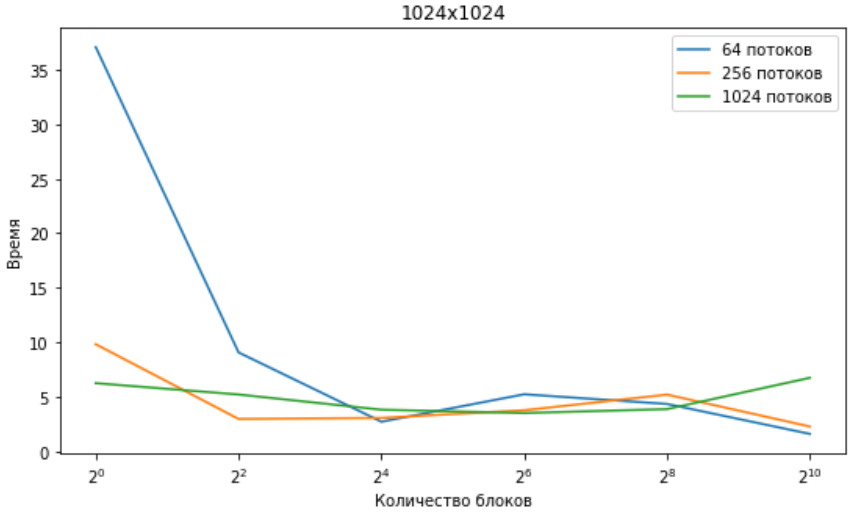
    - char *inname, *outname – имя входного и выходного файлов
    - int w, h , new_w, new_h – старые и новые размеры изображений
    - uchar4 *data – указатель на изображение
    - cudaArray *tex_ar – указатель на изображение в текстуре
- cudaArray *inti_texture(uchar4 *data, int w, int h) – функция, инициализирующая текстуру
- void write_data(char *outname, uchar4 *data, int w, int h) – запись изображения data в файл с именем outname
- void read_data(char *inname, uchar4 **data, int *w, int *h) – считывание изображения из файла inname в массив data
- __global__ void kernel(uchar4 *dev_arr, int w, int h, int h, int dw, int dh) – ядро, запускает поток для каждого пикселя выходного изображения.

Результаты

Посмотрим на зависимость скорости работы программы в зависимости от размеров выходного изображения.

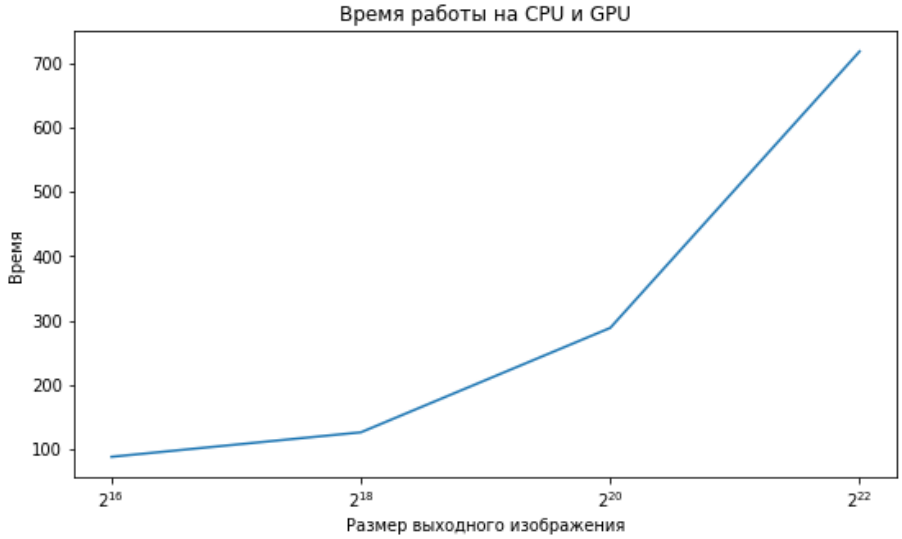






Как видно из этих графиков оптимальным соотношением количества потоков и блоков будет 8x8 потоков и 32x32 блоков. В этом случае использование текстурной памяти дает двукратный прирост в скорости по сравнению с использованием глобальной памяти. В остальных случаях использование текстурной памяти незначительно замедляет работу. Видимо потоки не слишком плотно располагаются к друг другу и кэширование не дает никаких преимуществ.

Посмотрим на скорость работы программы на cpu:



Как видно из графика, при увеличении выходного изображения в 4 раза, программа замедляется приблизительно в 2 раза, такое поведение можно объяснить. Все тесты проводились на входном изображении 4096x4096. Программа на cpu, проходит все пиксели только одни раз, но при росте выходного изображения издержки на циклах растут, требуется больше итераций на обработку одного пикселя.

Пример работы программы:

Оригинальное изображение 1600x800

Обработанное изображение 800x400

Обработанное изображение 400x200

Выводы

Основной принцип сглаживания — показ не только пикселей, через которые проходит примитив, но и закрашивание соседних с ним его оттенками. Таким образом достигается размытие границы. Один из таких алгоритмов сглаживания — ssaa. Он невероятно прост в реализации. Так в моей работе каждый поток обрабатывал один пиксель выходного изображения, однако недостаток моей реализации в том, что при размере выходного изображения много меньше исходного работает относительно небольшое количество потоков. Возможно было бы эффективнее применять по потоку для каждого пикселя входного изображения, но в таком случае пришлось бы использовать редукцию для подсчета значения результирующего пикселя.