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: Лобанов О. А.
Группа: 8О-408Б-20
Преподаватель: А.Ю. Морозов

Москва, 2023

Условие
Цель работы: Научиться использовать GPU для обработки изображений. Использование текстурной памяти и двухмерной сетки потоков.

Вариант 7. Выделение контуров. Метод Собеля.

Программное и аппаратное обеспечение

Device: Tesla T4
Compute capability: 7.5
Total constant memory: 65536
Registers per block: 65536
Max threads per block: 1024
Multiprocessors count: 40
OS: Ubuntu 22.04.2 LTS
Redactor: colab google

Метод решения

Для каждого пикселя выделяется по отдельному потоку, каждый из которых будет обрабатывать окрестность этого потока. Так как будет произведено много обращений к памяти, необходимо воспользоваться текстурной памятью. Результат обработки каждого пикселя будет записываться в массив.

Описание программы

Для выполнения операции я создал текстурную ссылку, после выделил память для массива на девайсе и скопировал в нее массив – изображение.
Для аппаратной обработки граничных условий я использую Clamp адресацию. В kernel вычисляем индекс исполняемой нити, который будет индексом в массиве при условии idy < размера массива. Далее производится расчет градиента по методу Собеля.

```
    for(long int a = idy; a < h; a += offsety) {
        for(long int b = idx; b < w; b += offsetx) {

            z1 = tex2D< uchar4 >(tex, b - 1, a - 1);
            w1 = grey(z1);
            z2 = tex2D< uchar4 >(tex, b, a - 1);
            w2 = grey(z2);
            z3 = tex2D< uchar4 >(tex, b + 1, a - 1);
            w3 = grey(z3);
            z4 = tex2D< uchar4 >(tex, b - 1, a);
            w4 = grey(z4);

            z5 = tex2D< uchar4 >(tex, b + 1, a);
            w5 = grey(z5);
            z6 = tex2D< uchar4 >(tex, b - 1, a + 1);
```

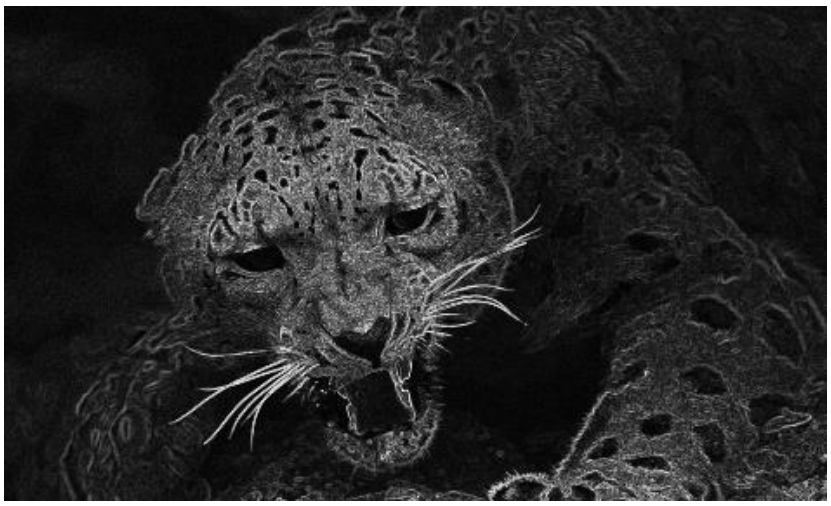
```
            w6 = grey(z6);
            z7 = tex2D< uchar4 >(tex, b, a + 1);
            w7 = grey(z7);
            z8 = tex2D< uchar4 >(tex, b + 1, a + 1);
            w8 = grey(z8);

            Gx = w3 + w5 + w5 + w8 - w1 - w4 - w4 - w6;
            Gy = w6 + w7 + w7 + w8 - w1 - w2 - w2 - w3;

            int GRAD = Gradient(Gx, Gy);
            int off = a * w + b;
            out[off].x = GRAD;
            out[off].y = GRAD;
            out[off].z = GRAD;
            out[off].w = 0;
        }
    }
```

После работы kernel я копирую данные в массив и освобождаю выделенную память.

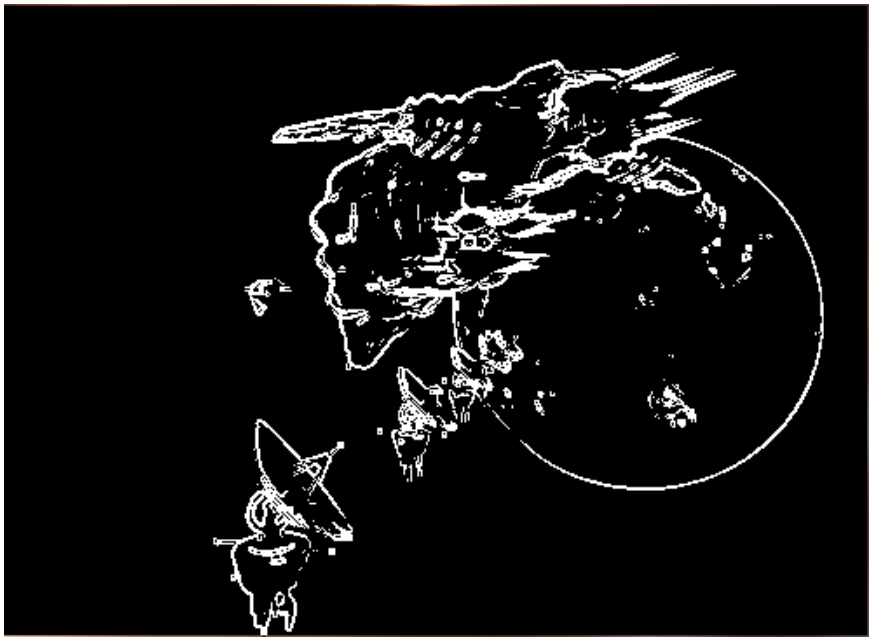
Результаты

Пример работы алгоритма:

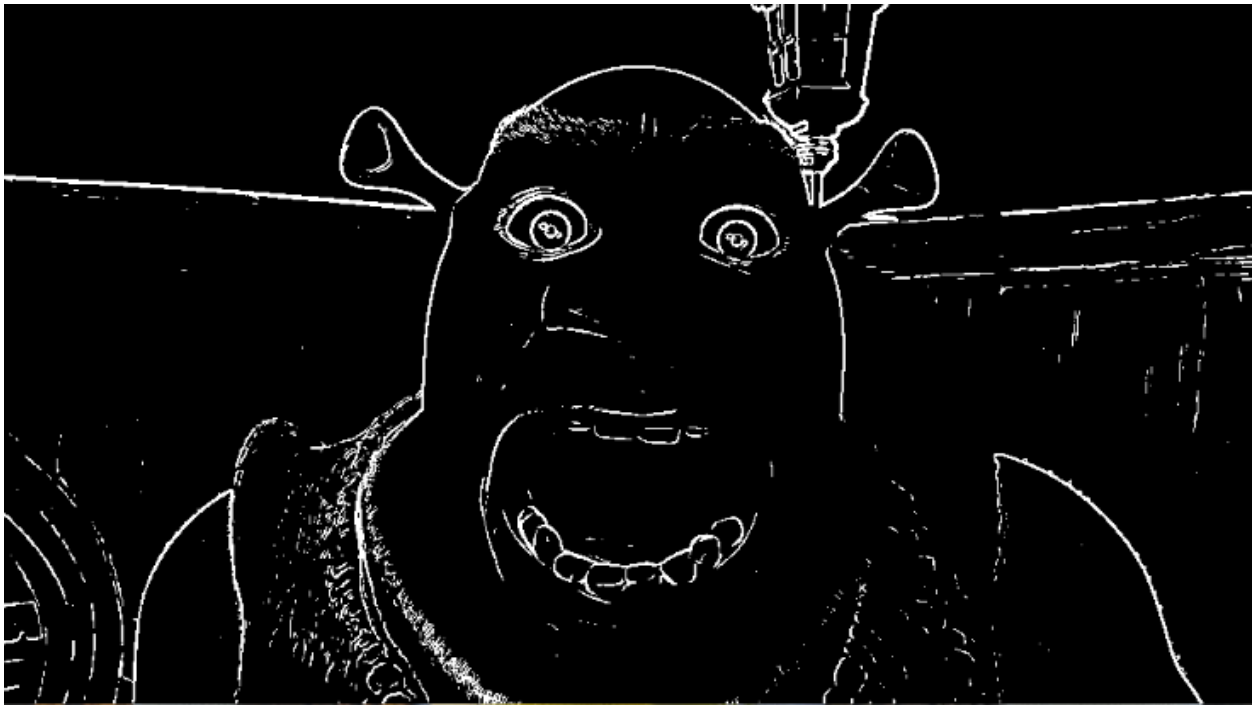
Картинка №1

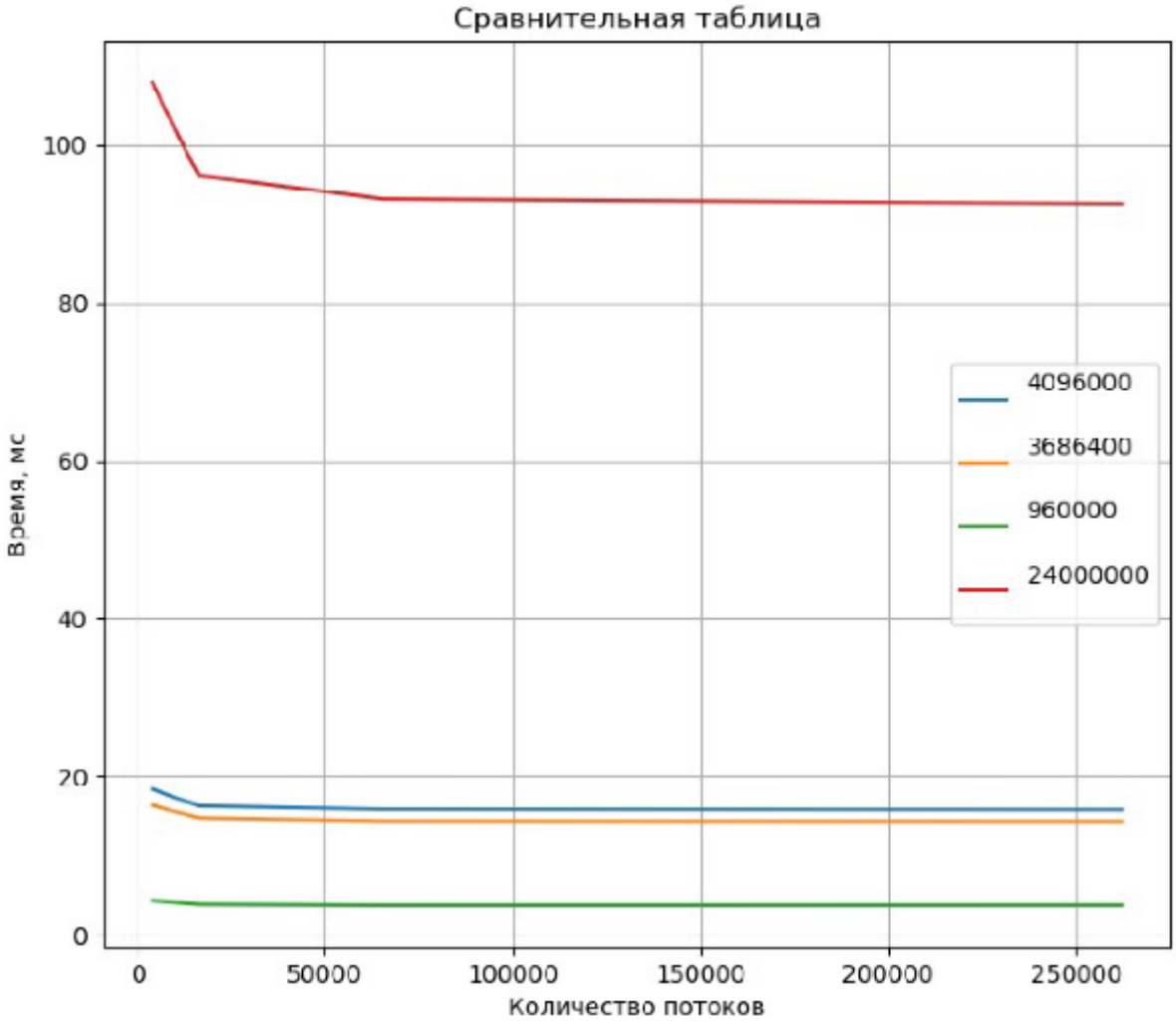


Картинка №2



Картинка №3





Из этой диаграммы видно, что с некоторого количества потоков производительность работы на GPU не возрастает. Порог для разных изображений разный, поскольку для больших изображений требуется большее количество нитей.

Выводы

Реализованный алгоритм применяется при обработке изображений, так как позволяет четко выделить контуры. Однако на конечном изображении можно увидеть светлый размывы, поэтому перед применением следует применить сглаживающие фильтры. Алгоритмы свертки хорошо распараллеливаются, что делает эффективным их использование на графических процессорах.
В ходе выполнения работы возникла трудность с нахождением элементов, расположенных вокруг пикселя, из – за чего долго не мог понять, почему результат получается не вырным.