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.

Выполнил: И.Д. Черненко
Группа: 8О-406Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2021

Условие

1. **Цель работы**: Научиться использовать GPU для обработки изображений. Использование текстурной памяти.
2. **Вариант задания: Вариант 6 Выделение контуров. Метод Превитта.**
   Входные данные: На первой строке задается путь к исходному изображению, на второй, путь к конечному изображению. $w*h \leq 10^8$.

Программное и аппаратное обеспечение

Compute capability : 6.1
Name : GeForce GTX 1050
Total Global Memory : 2096103424
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 : 5

OS:Linux
compiler:nvcc
IDE:VS Code
Editor:nano

Метод решения

Метод Превитта основан на обработке пикселей изображения ядрами 3 на 3, которые сворачивают изображение, вычисляя производные по оси x и по оси y, используя следующие маски:

$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G_y} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix} * \mathbf{A}$$

Для вычисления значения градиента в точке, требуется посчитать среднекважратичное значение градиентов по **x** и по **y**.

$$\nabla f = |\nabla f| = \sqrt{G_x^2 + G_y^2}$$

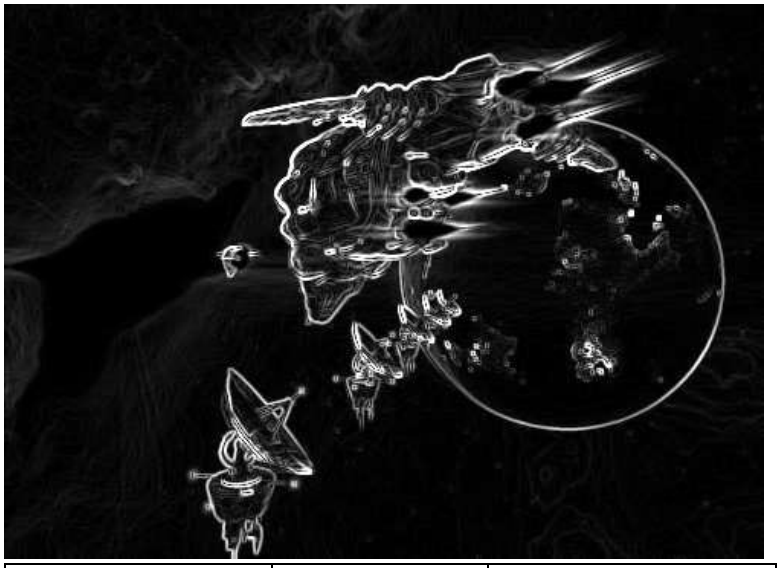
При рещении данной задачи очень помогли следующие статьи:
https://habr.com/ru/post/114452/
https://ru.wikipedia.org/wiki/Оператор_Прюитт

Описание программы

Вся программа была реализована в одном файле **main.cu**.
Реализованное ядро:

```
__global__ void kernel(uchar4* ans, uint32_t w, uint32_t h) {
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    int idy = blockDim.y * blockIdx.y + threadIdx.y;
    int offsetx = blockDim.x * gridDim.x;
    int offsety = blockDim.y * gridDim.y;
    uchar4 z[9];
    int first_col, last_col, first_str, last_str;
    for (int x = idx; x < w; x += offsetx) {
        for (int y = idy; y < h; y += offsety) {
            first_str = y - 1;
            last_str = y + 1;
            first_col = x - 1;
            last_col = x + 1;
            z[0] = tex2D(tex, first_col, first_str);
            z[1] = tex2D(tex, x, first_str);
            z[2] = tex2D(tex, last_col, first_str);
            z[3] = tex2D(tex, first_col, y);
            z[4] = tex2D(tex, x, y);
            z[5] = tex2D(tex, last_col, y);
            z[6] = tex2D(tex, first_col, last_str);
            z[7] = tex2D(tex, x, last_str);
            z[8] = tex2D(tex, last_col, last_str);
            float tmp = pw(z);
            unsigned char result = tmp;
            if (tmp >= 255) {
                result = 255;
            }
            ans[x + y * w] = make_uchar4(result, result, result, 255);
        }
    }
}
```

Результаты



blocks	threads	time
16x16	16x16	289.45us
16x16	32x32	488.15us
32x32	16x16	374.44us
32x32	32x32	376.83us
64x64	32x32	2.2700ms
1024x1024	32x32	458.78ms
CPU		41535,032us

Из полученных данных можно сделать вывод, что при правильных настройках вычисления на **GPU** превосходят **CPU** примерно в **143 раза.**

Выводы

Данный алгоритм можно было бы использовать для подготовки данных для нейросетей и фильтра изображений. Само написание кода и отладка заняло достаточно большое количество времени из-за работы с изображениями, что является далеко не тривиальной задачей. Как можно заметить из результатов, выполнение таких алгоритмов на GPU намного эффективнее чем на CPU, благодаря использованию текстурной памяти и прочих оптимизаций.