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: К.О. Вахрамян

Группа: 8О-406Б

Преподаватели: К.Г. Крашенинников,

А.Ю. Морозов

Москва, 2021

**Условие**

Кратко описывается задача:

1. Цель работы.

Научиться использовать GPU для обработки изображений. Использование текстурной памяти.

1. Вариант задания.

Медианный фильтр.

**Программное и аппаратное обеспечение**

**GPU:**

--- General Information for device ---

Name: NVIDIA GeForce GTX 1650

Compute capability: 7.5

Clock rate: 1560000

Device copy overlap: Enabled

Kernel execution timeout : Enabled

--- Memory Information for device ---

Total global mem: 4100521984

Total constant Mem: 65536

Max mem pitch: 2147483647

Texture Alignment: 512

--- MP Information for device ---

Multiprocessor count: 16

Shared mem per mp: 49152

Registers per mp: 65536

Threads in warp: 32

Max threads per block: 1024

Max thread dimensions: (1024, 1024, 64)

Max grid dimensions: (2147483647, 65535, 65535)

**CPU:**

Architecture: x86\_64

CPU op-mode(s): 32-bit, 64-bit

Byte Order: Little Endian

Address sizes: 39 bits physical, 48 bits virtual

CPU(s): 8

On-line CPU(s) list: 0-7

Thread(s) per core: 2

Core(s) per socket: 4

Socket(s): 1

NUMA node(s): 1

Vendor ID: GenuineIntel

CPU family: 6

Model: 158

Model name: Intel(R) Core(TM) i5-9300HF CPU @ 2.40GHz

Stepping: 13

CPU MHz: 1274.759

CPU max MHz: 2400.0000

CPU min MHz: 800.0000

BogoMIPS: 4800.00

Virtualization: VT-x

L1d cache: 128 KiB

L1i cache: 128 KiB

L2 cache: 1 MiB

L3 cache: 8 MiB

**OS:**

Linux Mint 20

**Compiler:**

nvcc

**Code Editor:**

VS Code

**Метод решения**

Пиксели обрабатываются в отдельных потоках. Поскольку поток обрабатывает радиус каждого пикселя, возможен выход за границы. Также очень велико количество обращений к памяти. Исходя из этого мы используем текстурную память.

Медианный фильтр выбирает среднее значение в некотором окне размера (2\*r+1)^2, где r — заранее заданный радиус. Проходим этим окном по изображению тем самым усредняем его.

**Описание программы**

Изображение хранится по указателю data и представляет собой массив из uchar4(структура которая имплементирует представление пикселя в RGB виде). Далее это изображение копируется в глобальную память устройства, которая связана с текстурной ссылкой.

int idx = blockDim.x \* blockIdx.x + threadIdx.x;

int idy = blockDim.y \* blockIdx.y + threadIdx.y;

Для каждого пикселя вызывается функция window и ее результат записывается в массив. Т.к. массив одномерный, индекс определяется как произведение текущей строки на ширину изображения плюс индекс текущего столбца.

data[y \* w + x] = window(x, y, radius);

Функция window находит медианное значение для данного радиуса. Для его нахождения используем гистограмму и неполную префиксную сумму по ней.

unsigned int s = 0;

for (int i = 0; i < 256; i++) {

s += r[i];

if (s >= n / 2 + 1) {

ans.x = i;

break;

}

}

И так для каждой компоненты пикселя.

**Результаты**

1-я фотография, разрешение 950х533.

До обработки:



Медианный фильтр с радиусом 4:



2-я фотография, разрешение 1280х720.

До обработки:



Медианный фильтр с радиусом 4:



Звезды пропали =)

Замеры производительности.

Все время представлено в миллисекундах.

|  |  |  |
| --- | --- | --- |
|  | Шрек 950х533 | Флаг 1280х720 |
| <<<(1,1),(1,32)>>>,R=4 | 1146.18000 | 2215.31000 |
| <<<(1,1),(1,32)>>>,R=15 | 3246.09 | 6194.11 |
| <<<(1,32),(1,32)>>>,R=4 | 147.51 | 215.63 |
| <<<(1,32),(1,32)>>>,R=15 | 332.74 | 571.17 |
| <<<(32,32),(32,32)>>>,R=4 | 93.9 | 130.92 |
| <<<(32,32),(32,32)>>>,R=15 | 953.38 | 1238.14 |
| <<<(1,128),(1,128)>>>,R=4 | 193.87 | 254.51 |
| <<<(1,128),(1,128)>>>,R=15 | 715.36 | 1134.69 |
| <<<(128,128),(8,128)>>>,R=4 | 73.26 | 125.94 |
| <<<(128,128),(8,128)>>>,R=15 | 722.35 | 1116.79 |
| CPU, R=4 | 1447.84 | 2824.5 |
| CPU, R=15 | 6063.07 | 11858.5 |

**Выводы**

В рамках 2-й лабораторной работы я реализовал медианный фильтр - один из видов цифровых фильтров, широко используемый в цифровой обработке сигналов и изображений для уменьшения уровня шума.

В процессе выполнения столкнулся с одной проблемой: текстурная память обрабатывает выход за границы и возвращает граничное значение, однако в медианном фильтре это не совсем правильно, потому что в подсчете среднего значения будут много раз учитываться граничные пиксели. Для борьбы с этим пришлось явно обрабатывать пороговые условия и «уменьшать» окно в некоторых случаях.

Касательно теста производительности, версия на CPU работает медленнее версии GPU с одним блоком на 32 потока. Версия с 128x128 блоками и 8x128 потоками работает на порядок быстрее версии хоста.