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»

Изучение технологии CUDA

Выполнил: А. О. Дубинин
Группа: 8О-407Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2020

Условие

Цель работы:

Научиться использовать GPU для обработки изображений.
Использование текстурной памяти.

Вариант 4. SSAA.

Необходимо реализовать избыточную выборку сглаживания. Исходное изображение представляет собой “экранный буфер”, на выходе должно быть сглаженное изображение, полученное уменьшением исходного.

Программное и аппаратное обеспечение

GeForce 940MX

Compute capability:	5.0
Dedicated video memory:	4096 MB
shared memory per block:	49152 bytes
constant memory:	65536 bytes
Total number of registers available per block:	65536
Maximum number of threads per multiprocessor:	2048
Maximum number of threads per block:	1024
(3) Multiprocessors, (128) CUDA Cores/MP:	384 CUDA Cores

Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz

Architecture:	x86_64
Byte Order:	Little Endian
CPU(s):	4
Thread(s) per core:	2
Core(s) per socket:	2
CPU MHz: CPU max MHz: CPU min MHz:	713.848 3100,0000 400,0000
L1d cache: L1i cache: L2 cache: L3 cache:	64 KiB 64 KiB 512 KiB 3 MiB

RAM	8GiB SODIMM DDR4 Synchronous Unbuffered (Unregistered) 2400 MHz (0,4 ns)

SSD(SPCC_M.2_SSD)	223,6G
HDD(ST1000LM035-1RK172)	931,5G

OS: Ubuntu 20.04 focal
IDE: sublime3
compiler: nvcc

Метод решения

Переписал код с лекции поменяв пробег по субпикселями на device'e и ввод размеров нового изображения в main'e. Пробежав по всем субпиксилям берем среднее арифметическое, согласно данной формуле:

$$result = \frac{\sum_{i=0}^{2^n-1} sample_i}{2^n}$$

Описание программы

Для решения задачи, нужно было сначала понять, как пробежаться по субпикселями. Кол-во субпикселей = $\left(\frac{w}{w_{new}}\right) * \left(\frac{h}{h_{new}}\right)$, то сколько мы можем предоставить независимых субпикселей для одного нового пикселя. Следовательно кол-во субпикселей на оси x будет $\frac{w}{w_{new}}$, а на оси y $\frac{h}{h_{new}}$. Будем использовать текстурную память, так как она эффективно работает с текстурами\изображениями.

```
__global__ void kernel(uchar4 *out, int w, int h, int wScale, int hScale) {
	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;
	int n = wScale * hScale;
	int x, y, i, j;
	uchar4 p;
	uint4 s;

	for(y = idy; y < h; y += offsety) {
		for(x = idx; x < w; x += offsetx) {
			s = {0,0,0,0};
			for (i = 0; i < wScale; ++i) {
				for (j = 0; j < hScale; ++j){
					p = tex2D(tex, x * wScale + i, y * hScale + j);
					s.x += p.x;
					s.y += p.y;
					s.z += p.z;
				}
			}
			s.x /= n;
			s.y /= n;
			s.z /= n;

			out[y * w + x] = make_uchar4(s.x, s.y, s.z, s.w);
		}
	}
}
```

Результаты

1.

	Небольшой тест	Средний тест	Большой тест
<<<1, 32>>>, <<<1, 32>>>	5.31	3.99	14.09
<<<32, 32>>>, <<<32, 32>>>	2.40	7.97	19.30
<<<1, 128>>>, <<<1, 128>>>	3.98	4.06	16.52
<<<1, 128>>>, <<<32, 32>>>	3.51	8.75	19.72
<<<1, 256>>>, <<<1, 256>>>	4.12	4.40	17.95
<<<256, 256>>>, <<<32,32>>>	5.84	10.51	24.53
<<<1, 512>>>, <<<1, 512>>>	4.12	5.43	16.53
<<<512, 512>>>, <<<32, 32>>>	14.71	21.94	21.94
<<<1, 1024>>>, <<<1, 1024>>>	6.28	5.58	17.72

<<<1024, 1024>>>, <<<32, 32>>>	49.16	52.48	62.60

2.

	Небольшой тест	Средний тест	Большой тест
GPU<<<16, 16>>>, <<<16, 16>>>	2.54	4.25	22.24
CPU	86.80	44.83	40.22

Пример: input = 3840x2400, output = 120x75 — Большой тест

Выводы

Данный алгоритм устарел, сейчас он редко применяется, так как трудозатратен. Сложность возникла, когда у меня вместо сглаживания, картинка становилась более тусклой. Ошибка была в том, что я подсчитывал значения субпикселей в переменные типа char, поэтому у меня было переполнение, все решилось использованием int.