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: В.И. Лобов
Группа: 8О-406Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2019

Условие

Цель работы: Научиться использовать GPU для обработки изображений. Использование текстурной памяти.
Вариант 5. Выделение контуров. Метод Робертса.

Входные данные: На первой строке задается путь к исходному изображению, на второй, путь к конечному изображению. $w * h \leq 10$.

Программное и аппаратное обеспечение

GPU:

Название: GeForce GTX 1060
Compute apability: 6.1
Графическая память: 2075328512
Разделяемая память: 49152
Константная память: 65536
Количество регистров на блок: 65536
Максимальное количество блоков: (2147483647, 65535, 65535)
Максимальное количество нитей: (1024, 1024, 64)
Количество мультипроцессоров: 10

Сведения о системе:

Процессор: Intel Core i7-8700k 4.5GHz
Оперативная память: 16Gb
HDD: 1Tb
Операционная система: Ubuntu 18.04
IDE: Nsight
Компилятор: nvcc

Метод решения

Метод Робертса выделения контуров изображения.
Основой метода является вычисление градиента изображения, каждый пиксель которого имеет яркость img[i][j], в каждой точке изображения.

$$|\nabla| = \sqrt{Gx^2 + Gy^2},$$
$$Gx(i, j) = img[i + 1][j + 1] — img[i][j],$$
$$Gy(i, j) = img[i + 1][j] — img[i][j + 1];$$

Применив оператор Робертса ко всему изображению, получим изображение в оттенках серого, в котором границы объектов светлее, чем всё остальное изображение.

Описание программы

Создадим объект текстуры и загрузим в него данные изображения. Затем, каждый поток вычисляет значения модуля градиента изображения в отдельных пикселях и записывает их в соответствующий массив выходного изображения. Значение яркости каждого пикселя находится по соответствующей формуле.

```
#include <stdio.h>
#include <math.h>


texture<uchar4, 2, cudaReadModeElementType> tex;

__device__ int Intensity(uchar4 p) {
	return p.x * 0.299 + p.y * 0.587 + p.z * 0.114;
}

__global__ void kernel(uchar4 *dst, int w, int 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;
	int x, y;
	int Gx, Gy;
	int grad;
	for (x = idx; x < w; x += offsetx) {
		for (y = idy; y < h; y += offsety) {
			Gx = Intensity(tex2D(tex, x, y)) -
			    Intensity(tex2D(tex, x + 1, y + 1));
			Gy = Intensity(tex2D(tex, x, y + 1)) -
			    Intensity(tex2D(tex, x + 1, y));
			grad = min(int(sqrt(float(Gx * Gx + Gy * Gy))), 255);
			dst[y * w + x] = make_uchar4(grad, grad, grad, 1);
		}
	}
}

int main() {
	char input_file[1024], output_file[1024];
	scanf("%s", input_file);
	scanf("%s", output_file);

	int w, h;
	FILE *in = fopen(input_file, "rb");
	fread(&w, sizeof(uchar4), 1 , in);
	fread(&h, sizeof(uchar4), 1 , in);
	uchar4 *data = (uchar4*)malloc(sizeof(uchar4) * h * w);
	fread(data, sizeof(uchar4), h * w, in);
	fclose(in);

	cudaArray *arr;
	cudaChannelFormatDesc ch = cudaCreateChannelDesc<uchar4>();
	cudaMallocArray(&arr, &ch, w, h);
	cudaMemcpyToArray(arr, 0, 0, data, sizeof(uchar4) * h * w,
cudaMemcpyHostToDevice);

	tex.addressMode[0] = cudaAddressModeClamp;
	tex.addressMode[1] = cudaAddressModeClamp;
	tex.channelDesc = ch;
	tex.filterMode = cudaFilterModePoint;
	tex.normalized = false;
```

```
    cudaBindTextureToArray(tex, arr, ch);
    uchar4 *dev_data;
    cudaMalloc(&dev_data, sizeof(uchar4) * h * w);
    kernel<<<dim3(16, 16), dim3(16, 16)>>>(dev_data, w, h);
    cudaMemcpy(data, dev_data, sizeof(uchar4) * h * w, cudaMemcpyDeviceToHost);

    FILE *out = fopen(output_file, "wb");
    fwrite(&w, sizeof(uchar4), 1, out);
    fwrite(&h, sizeof(uchar4), 1, out);
    fwrite(data, sizeof(uchar4), h * w, out);
    fclose(out);

    cudaUnbindTexture(tex);
    cudaFreeArray(arr);
    cudaFree(dev_data);
    free(data);

    return 0;
}
```

Результаты:

Размер картинки: 4000x4000

Конфигурация	Время(мс)
CPU	647.89
(16, 16), (16, 16)	7.778976
(64, 64), (32, 32)	7.421664
(256, 256), (32, 32)	9.147520
(512, 512), (16, 16)	8.148096

Выводы

Выполнив данную лабораторную работу, я научился работать с текстурной памятью — специальной ReadOnly памятью, предназначенной для хранения изображений и работы с ними. Работать с ней довольно удобно — при обращении к индексу за границами изображения возвращается значение другого пикселя(например, значение на границе

изображения). Протестировав программу на различных конфигурациях, можно сделать вывод, что вне зависимости от количества блоков в гриде время работы находится в пределах погрешности.