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.

Выполнила: Алексюнина Ю.В.

Группа:М8О-407Б

Преподаватели:

К.Г.Крашенинников, А.Ю. Морозов

Москва, 2021

Условие:

Цель работы. Научиться использовать GPU для обработки изображений. Использование текстурной памяти.

Формат изображений. Изображение является бинарным файлом, со следующей структурой:

width(w)	height(h)	r	g	b	a	r	g	b	a	r	g	b	a	...	r	g	b	a	r	g	b	a
4 байта, int	4 байта, int	4 байта, значение пикселя [1,1]				4 байта, значение пикселя [2,1]				4 байта, значение пикселя [3,1]				...	4 байта, значение пикселя [w - 1, h]				4 байта значение пикселя [w,h]			

В первых восьми байтах записывается размер изображения, далее построчно все значения пикселей, где

- r -- красная составляющая цвета пикселя
- g -- зеленая составляющая цвета пикселя
- b -- синяя составляющая цвета пикселя
- a -- значение альфа-канала пикселя

Пример картинки размером 2 на 2, синего цвета, в шестнадцатеричной записи:

```
02000000 02000000 0000FF00 0000FF00 0000FF00 0000FF00
```

Вариант: 3. Билинейная интерполяция.

Необходимо реализовать изменение размера изображения, используя билинейную интерполяцию. Использовать встроенную текстурную билинейную интерполяцию запрещается.

Входные данные. На первой строке задается путь к исходному изображению, на второй, путь к конечному изображению. На следующей строке, два числа wn и hn - новые размеры изображения. $w*h \le 4 * 10^8$.

Программное и аппаратное обеспечение

GPU:

- Name: GeForce GTX 750 Ti
- Compute capability: 5.0
- Графическая память: 4294967295
- 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: 5

Сведения о системе:
• Процессор: Intel Core i5-4460 3.20GHz
• ОЗУ: 16 ГБ
• HDD: 930 ГБ

Программное обеспечение:
• OS: Windows 8.1
• IDE: Visual Studio 2019
• Компилятор: nvcc

Метод решения:

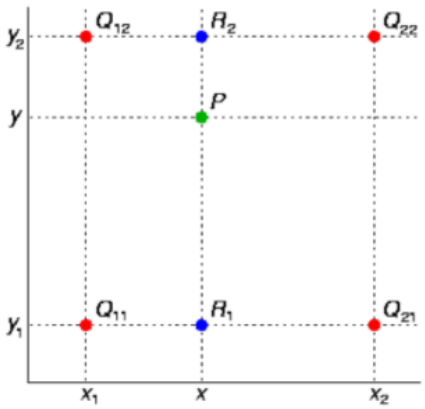
$$Q_{11} = (x_1, y_1), \quad Q_{12} = (x_1, y_2)$$
$$Q_{21} = (x_2, y_1), \quad Q_{22} = (x_2, y_2)$$

$$R_1 = (x, y_1), \quad R_2 = (x, y_2)$$

$$f(R_1) = \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$

Аналогично $f(R_2)$

$$f(P) = \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2)$$



Файл kernel.cu:

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <algorithm>

using namespace std;

#define CSC(call)
                \
```

```
do {
                \
     cudaError_t res = call;
                \
     if (res != cudaSuccess) {
                \
          fprintf(stderr, "ERROR in %s:%d. Message: %s\n",
     \
                    __FILE__, __LINE__, cudaGetErrorString(res));
          \
          exit(0);
                \
     }
                    \
} while(0)


// текстурная ссылка <тип элементов, размерность, режим
нормализации>
texture<uchar4, 2, cudaReadModeElementType> tex;


__global__ void bilinear_interp_kernel(uchar4* out, int w, int
h, int wn, int hn)
{
     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 x1, y1;
     uchar4 p1, p2, p3, p4;
     double r, g, b;
     double x2, y2;
     for (y = idy; y < hn; y += offsety)
          for (x = idx; x < wn; x += offsetx)
          {
```

```
                double h_d = (double)h / hn;
                double w_d = (double)w / wn;

                double x_n = (x + 0.5) * w_d - 0.5;
                double y_n = (y + 0.5) * h_d - 0.5;

                x1 = x_n; //x_r
                x2 = x_n - x1;//ceil(x_n);
                if (x2 < 0.0) //////////
                {
                     x2++;
                     x1--;
                }
                y1 = y_n;
                y2 = y_n - y1; //ceil(y_n);
                if (y2 < 0.0) /////////
                {
                     y2++;
                     y1--;
                }

                //printf("%f %f %f %f \n", x1, x2, y1, y2); +

               p1 = tex2D(tex, x1, y1);
                p2 = tex2D(tex, x1 + 1, y1);
                p3 = tex2D(tex, x1, y1 + 1);
                p4 = tex2D(tex, x1 + 1, y1 + 1);

                //r = (p1.x * (x2 - x_n) * (y2 - y_n) + p2.x *
(x_n - x1) * (y2 - y_n) + p3.x * (x2 - x_n) * (y_n - y1) + p4.x
* (x_n - x1) * (y_n - y1)) / ((x2 - x1) * (y2 - y1));
```

```
                //g = (p1.y * (x2 - x_n) * (y2 - y_n) + p2.y *
(x_n - x1) * (y2 - y_n) + p3.y * (x2 - x_n) * (y_n - y1) + p4.y
* (x_n - x1) * (y_n - y1)) / ((x2 - x1) * (y2 - y1));
                //b = (p1.z * (x2 - x_n) * (y2 - y_n) + p2.z *
(x_n - x1) * (y2 - y_n) + p3.z * (x2 - x_n) * (y_n - y1) + p4.z
* (x_n - x1) * (y_n - y1)) / ((x2 - x1) * (y2 - y1));


                r = p1.x * (1.0 - x2) * (1.0 - y2) + p2.x * x2 *
(1.0 - y2) + p3.x * (1.0 - x2) * y2 + p4.x * x2 * y2;
                g = p1.y * (1.0 - x2) * (1.0 - y2) + p2.y * x2 *
(1.0 - y2) + p3.y * (1.0 - x2) * y2 + p4.y * x2 * y2;
                b = p1.z * (1.0 - x2) * (1.0 - y2) + p2.z * x2 *
(1.0 - y2) + p3.z * (1.0 - x2) * y2 + p4.z * x2 * y2;


                //if (r > 255)
                //{
                     //printf("SHIT\n");
                //}


                out[y * wn + x] = make_uchar4(r, g, b, p1.w);
          }
}


int main() {
     int w, h, wn, hn;
     string inFile;
     string outFile;


     std::cin >> inFile >> outFile >> wn >> hn;


     FILE* fp = fopen(inFile.c_str(), "rb");
     //FILE* fp = fopen("in.data", "rb");
     fread(&w, sizeof(int), 1, fp);
     fread(&h, sizeof(int), 1, fp);
     uchar4* data = (uchar4*)malloc(sizeof(uchar4) * w * h);
```

```
    uchar4* data_ = (uchar4*)malloc(sizeof(uchar4) * wn * hn);
    fread(data, sizeof(uchar4), w * h, fp);
    fclose(fp);

    CSC(cudaSetDevice(0));

    // Подготовка данных для текстуры
    cudaArray* arr;
    cudaChannelFormatDesc ch = cudaCreateChannelDesc<uchar4>();
    CSC(cudaMallocArray(&arr, &ch, w, h));

    CSC(cudaMemcpyToArray(arr, 0, 0, data, sizeof(uchar4) * w *
h, cudaMemcpyHostToDevice));

    // Подготовка текстурной ссылки, настройка интерфейса работы
с данными
    tex.addressMode[0] = cudaAddressModeClamp;  // Политика
обработки выхода за границы по каждому измерению
    tex.addressMode[1] = cudaAddressModeClamp;
    tex.channelDesc = ch;
    tex.filterMode = cudaFilterModePoint;       // Без
интерполяции при обращении по дробным координатам
    tex.normalized = false;                             // Режим
нормализации координат: без нормализации

    // Связываем интерфейс с данными
    CSC(cudaBindTextureToArray(tex, arr, ch));

    uchar4* out;

    CSC(cudaMalloc(&out, sizeof(uchar4) * wn * hn));

    bilinear_interp_kernel << <dim3(16, 16), dim3(16, 16) >> >
(out, w, h, wn, hn);
```

```
    CSC(cudaMemcpy(data_, out, sizeof(uchar4) * wn * hn,
cudaMemcpyDeviceToHost));

    // Отвязываем данные от текстурной ссылки
    CSC(cudaUnbindTexture(tex));

    CSC(cudaFreeArray(arr));

    CSC(cudaFree(out));

    fp = fopen(outFile.c_str(), "wb");
    fwrite(&wn, sizeof(int), 1, fp);
    fwrite(&hn, sizeof(int), 1, fp);
    //data = (uchar4*)malloc(sizeof(uchar4) * wn * hn);
    fwrite(data_, sizeof(uchar4), wn * hn, fp);
    fclose(fp);

    free(data);
    free(data_);
    return 0;
}
```

Результаты:

	1000x1000	**200x200**
dim3(32,32)	**26.199ms**	**4.186ms**
dim3(64,64)	**3.011ms**	**2.032ms**
dim3(128,128)	**2.702ms**	**2.051ms**
dim3(256,256)	**2.702ms**	**1.894ms**
dim3(512,512)	**2.703ms**	**1.898ms**

Выводы:


黄山
徽菜楼