ENTREGABLE 2: GRAFICOS EN CUDA

AARON ROJAS Y DAVID PEIROTÉN

```
/**
*
* ARQUITECTURA DE COMPUTADORES
* 2º Grado en Ingenieria Informatica
* Curso: 2022 - 2023
*
* ENTREGA no.1 <Graficos en CUDA>
*
* EQUIPO : TE - C - 25
* MIEMBROS : Aaron Rojas Gutierrez y David Peiroten Herrero
*
*
*
*/

#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>
#include "gpu_bitmap.h"
// defines
#define ANCHO 512 // Dimension horizontal
#define ALTO 512 // Dimension vertical

__host__ void propiedades_Device(int deviceID);

// GLOBAL: funcion llamada desde el host y ejecutada en el device (kernel)
__global__ void kernel(unsigned char *imagen)
{
	// ** Kernel bidimensional multibloque **
	//
	// coordenada horizontal de cada hilo
	int x = threadIdx.x + blockIdx.x * blockDim.x;
	// coordenada vertical de cada hilo
	int y = threadIdx.y + blockIdx.y * blockDim.y;
	// indice global de cada hilo (indice lineal para acceder a la memoria)
	int myID = x + y * blockDim.x * gridDim.x;
	// cada hilo obtiene la posicion de su pixel
	int miPixel = myID * 4;
	// cada hilo rellena los 4 canales de su pixel con un valor arbitrario
	if (((blockIdx.x/4) + (blockIdx.y/4)) % 2 == 0) {
		imagen[miPixel + 0] = 0; // canal R
		imagen[miPixel + 1] = 0;// canal G
		imagen[miPixel + 2] = 0; // canal B
		imagen[miPixel + 3] = 0; // canal alfa
	}
	else
	{
		imagen[miPixel + 0] = 250; // canal R
		imagen[miPixel + 1] = 250;// canal G
		imagen[miPixel + 2] = 250; // canal B
		imagen[miPixel + 3] = 0; // canal alfa
	}
```

```
}
// MAIN: rutina principal ejecutada en el host
int main(int argc, char** argv)
{
	// buscando dispositivos
	int deviceCount;
	cudaGetDeviceCount(&deviceCount);
	if (deviceCount == 0)
	{
		printf("!!!!!No se han encontrado dispositivos CUDA!!!!!\n");
		printf("<pulsa [INTRO] para finalizar>");
		getchar();
		return 1;
	}
	else
	{
		printf("Se han encontrado <%d> dispositivos CUDA:\n", deviceCount);
		for (int id = 0; id < deviceCount; id++)
		{
			propiedades_Device(id);
		}
	}

	//Declarar variables y eventos para monitorizar el tiempi
	cudaEvent_t start;
	cudaEvent_t stop;

	//Creacion de eventos
	cudaEventCreate(&start);
	cudaEventCreate(&stop);

	// Declaracion del bitmap:
	// Inicializacion de la estructura RenderGPU
	RenderGPU foto(ANCHO, ALTO);

	// Tamaño del bitmap en bytes
	size_t size = foto.image_size();

	// Asignacion y reserva de la memoria en el host (framebuffer)
	unsigned char *host_bitmap = foto.get_ptr();

	// Reserva en el device
	unsigned char *dev_bitmap;
	cudaMalloc((void**)&dev_bitmap, size);

	// Lanzamos un kernel bidimensional con bloques de 256 hilos (16x16)
	dim3 hilosB(16, 16);

	// Calculamos el numero de bloques necesario (un hilo por cada pixel)
	dim3 Nbloques(ANCHO / 16, ALTO / 16);

	// Generamos el bitmap
	//Marcamos el tiempo de inicio
	cudaEventRecord(start, 0);
	kernel << <Nbloques, hilosB >> >(dev_bitmap);

	// Copiamos los datos desde la GPU hasta el framebuffer para visualizarlos
	cudaMemcpy(host_bitmap, dev_bitmap, size, cudaMemcpyDeviceToHost);

	//Marca de finalizacion

	cudaEventRecord(stop, 0);
```

David Peirotén
Aaron Rojas

```
	//Sincronizacion de eventos
	cudaEventSynchronize(stop);

	//Calculo de intentos en milisegundos
	float elapsedTime;

	cudaEventElapsedTime(&elapsedTime, start, stop);

	// Visualizacion y salida
	printf("> Tiempo de ejecucion: %f ms\n", elapsedTime);
	printf("\n...pulsa [ESC] para finalizar...\n");
	printf("*************************************************\n");
	// liberacion de recursos
	cudaEventDestroy(start);
	cudaEventDestroy(stop);
	foto.display_and_exit();

	return 0;
}
__host__ void propiedades_Device(int deviceID)
{
	cudaDeviceProp deviceProp;
	cudaGetDeviceProperties(&deviceProp, deviceID);
	// calculo del numero de cores (SP)
	int cudaCores = 0;
	int SM = deviceProp.multiProcessorCount;
	int major = deviceProp.major;
	int minor = deviceProp.minor;
	const char *archName;
	switch (major)
	{
	case 1:
		//TESLA
		archName = "TESLA";
		cudaCores = 8;
		break;
	case 2:
		//FERMI
		archName = "FERMI";
		if (minor == 0)
			cudaCores = 32;
		else
			cudaCores = 48;
		break;
	case 3:
		//KEPLER
		archName = "KEPLER";
		cudaCores = 192;
		break;
	case 5:
		//MAXWELL
		archName = "MAXWELL";
		cudaCores = 128;
		break;
	case 6:
		//PASCAL
		archName = "PASCAL";
		cudaCores = 64;
		break;
	case 7:
		//VOLTA(7.0) //TURING(7.5)
```

David Peirotén
Aaron Rojas

```
                cudaCores = 64;
                if (minor == 0)
                        archName = "VOLTA";
                else
                        archName = "TURING";
                break;
        case 8:
                // AMPERE
                archName = "AMPERE";
                cudaCores = 64;
                break;
        default:
                //ARQUITECTURA DESCONOCIDA
                archName = "DESCONOCIDA";
        }
        int rtV;
        cudaRuntimeGetVersion(&rtV);
        // presentacion de propiedades
        printf("*****************************************************\n");
        printf("DEVICE %d: %s\n", deviceID, deviceProp.name);
        printf("*****************************************************\n");
        printf("> CUDA Toolkit \t: %d.%d\n", rtV / 1000, (rtV % 1000) / 10);
        printf("> Arquitectura CUDA \t: %s\n", archName);
        printf("> Capacidad de Computo \t: %d.%d\n", major, minor);
        printf("> No. MultiProcesadores \t: %d\n", SM);
        printf("> MAX Hilos por bloque: %d\n", deviceProp.maxThreadsPerBlock);
        printf("> No. Nucleos CUDA (%dx%d) \t: %d\n", cudaCores, SM,
cudaCores*SM);
        printf("> Memoria Global (total) \t: %u MiB\n",
                deviceProp.totalGlobalMem / (1024 * 1024));
        printf("*****************************************************\n");
}
```