Перевод цветного изображения в градации серого на CUDA

GPU Teaching Kit – Accelerated Computing

# Цели

Цель этой лабораторной работы - перевести RGB изображение в изображение в градациях серого. Входные данные – тройка RGB чисел с плавающей точкой, которую студент должен преобразовать в единственное значение с плавающей точкой – интенсивность серого цвета. Ниже представлен псевдокод алгоритма:

for ii from 0 to height do  
 for jj from 0 to width do  
 idx = ii \* width + jj  
 # количество каналов - 3  
 r = input[3\*idx]  
 g = input[3\*idx + 1]  
 b = input[3\*idx + 2]  
 grayImage[idx] = (0.21\*r + 0.71\*g + 0.07\*b)  
 end  
end

# Предварительные требования

Прежде чем начать, убедитесь, что:

* вы посмотрели все видео модуля

# Формат изображения

Для тех, кто работает на своей собственной системе, входные изображения хранятся в формате PPM P6, в то время как выходное изображение хранится в формате PPM P5. Студенты могут конвертировать свои собственные изображения в PPM. Самый простой вариант – задействовать сторонние инструменты. На системах Unix bmptoppm конвертирует BMP в PPM.

# Ход выполнения

Отредактируйте код, чтобы выполнить следующие действия:

* выделить память на устройстве
* скопировать память хоста на устройство
* инициализировать размерности блока нитей и ядра
* запустить ядро CUDA
* скопировать результат на хост
* освободить память устройства

Места, куда следует вставить код, помечены комментариями //@@.

# Инструкция по установке

Последнюю версию исходного кода лабораторной, наряду со скриптами сборки, можно найти в репозитории [Bitbucket](https://bitbucket.org/hwuligans/gputeachingkit-labs/src/master/Module3). Инструкции к Cmake и сборки лабораторной можно найти в файле [README](https://bitbucket.org/hwuligans/gputeachingkit-labs/src/master/README.md) в корневой директории репозитория.

Исполняемый файл, являющийся результатом компиляции лабораторной, можно запустить следующей командой:

./ImageColorToGrayscale\_Template -e <expected.ppm> -i <input.ppm> \  
 -o <output.ppm> -t image

где <expected.ppm> – имя ожидаемого результата, <input.ppm> – входной набор данных и <output.ppm> – необязательный путь для результатов. Набор данных можно сгенерировать, используя генератор, собранный во время компиляции.

# Вопросы

1. Сколько операций с плавающей точкой выполняется в вашем ядре? ОБЪЯСНИТЬ.
2. Какой формат будет более эффективным для преобразования цвета: 2D матрица, где каждый элемент представляет собой значение RGB, или 3D матрица, где каждый срез по оси Z обозначает цвет? То есть будет ли лучше чередование цветов? Вы можете назвать приложение, где противоположный вариант будет лучшим?
3. Сколько чтений из глобальной памяти производится вашим ядром? ОБЪЯСНИТЬ.
4. Сколько записей в глобальную память производится вашим ядром? ОБЪЯСНИТЬ.
5. Опишите, какие оптимизации можно сделать в вашем ядре, чтобы достичь увеличения производительности.
6. Назовите три применения преобразования цветов.

# Шаблон кода

Представленный код предлагается как отправная точка. Импорты, экспорты и проверка решения уже представлены в коде. Требуется вставить свой код в области, обозначенные //@@. Остальной код трогать не нужно. Руководство описывает функционал методов с префиксом wb\*.

#include <wb.h>  
  
#define wbCheck(stmt) \  
 do { \  
 cudaError\_t err = stmt; \  
 if (err != cudaSuccess) { \  
 wbLog(ERROR, "Failed to run stmt ", #stmt); \  
 wbLog(ERROR, "Got CUDA error ... ", cudaGetErrorString(err)); \  
 return -1; \  
 } \  
 } while (0)  
  
//@@ Место для вставки кода  
  
int main(int argc, char \*argv[]) {  
 wbArg\_t args;  
 int imageChannels;  
 int imageWidth;  
 int imageHeight;  
 char \*inputImageFile;  
 wbImage\_t inputImage;  
 wbImage\_t outputImage;  
 float \*hostInputImageData;  
 float \*hostOutputImageData;  
 float \*deviceInputImageData;  
 float \*deviceOutputImageData;  
  
 args = wbArg\_read(argc, argv); /\* чтение входных аргументов \*/  
  
 inputImageFile = wbArg\_getInputFile(args, 0);  
  
 inputImage = wbImport(inputImageFile);  
  
 imageWidth = wbImage\_getWidth(inputImage);  
 imageHeight = wbImage\_getHeight(inputImage);  
 // В данной лабораторной значение равно 3  
 imageChannels = wbImage\_getChannels(inputImage);  
  
 // Так как изображение монохромное, оно содержит только 1 канал  
 outputImage = wbImage\_new(imageWidth, imageHeight, 1);  
  
 hostInputImageData = wbImage\_getData(inputImage);  
 hostOutputImageData = wbImage\_getData(outputImage);  
  
 wbTime\_start(GPU, "Doing GPU Computation (memory + compute)");  
  
 wbTime\_start(GPU, "Doing GPU memory allocation");  
 cudaMalloc((void \*\*)&deviceInputImageData,  
 imageWidth \* imageHeight \* imageChannels \* sizeof(float));  
 cudaMalloc((void \*\*)&deviceOutputImageData,  
 imageWidth \* imageHeight \* sizeof(float));  
 wbTime\_stop(GPU, "Doing GPU memory allocation");  
  
 wbTime\_start(Copy, "Copying data to the GPU");  
 cudaMemcpy(deviceInputImageData, hostInputImageData,  
 imageWidth \* imageHeight \* imageChannels \* sizeof(float),  
 cudaMemcpyHostToDevice);  
 wbTime\_stop(Copy, "Copying data to the GPU");  
  
 ///////////////////////////////////////////////////////  
 wbTime\_start(Compute, "Doing the computation on the GPU");  
 //@@ Место для вставки кода  
  
 wbTime\_stop(Compute, "Doing the computation on the GPU");  
  
 ///////////////////////////////////////////////////////  
 wbTime\_start(Copy, "Copying data from the GPU");  
 cudaMemcpy(hostOutputImageData, deviceOutputImageData,  
 imageWidth \* imageHeight \* sizeof(float),  
 cudaMemcpyDeviceToHost);  
 wbTime\_stop(Copy, "Copying data from the GPU");  
  
 wbTime\_stop(GPU, "Doing GPU Computation (memory + compute)");  
  
 wbSolution(args, outputImage);  
  
 cudaFree(deviceInputImageData);  
 cudaFree(deviceOutputImageData);  
  
 wbImage\_delete(outputImage);  
 wbImage\_delete(inputImage);  
  
 return 0;  
}