// Homework 1

// Color to Greyscale Conversion

//A common way to represent color images is known as RGBA - the color

//is specified by how much Red, Grean and Blue is in it.

//The 'A' stands for Alpha and is used for transparency, it will be

//ignored in this homework.

//Each channel Red, Blue, Green and Alpha is represented by one byte.

//Since we are using one byte for each color there are 256 different

//possible values for each color. This means we use 4 bytes per pixel.

//Greyscale images are represented by a single intensity value per pixel

//which is one byte in size.

//To convert an image from color to grayscale one simple method is to

//set the intensity to the average of the RGB channels. But we will

//use a more sophisticated method that takes into account how the eye

//perceives color and weights the channels unequally.

//The eye responds most strongly to green followed by red and then blue.

//The NTSC (National Television System Committee) recommends the following

//formula for color to greyscale conversion:

//I = .299f \* R + .587f \* G + .114f \* B

//Notice the trailing f's on the numbers which indicate that they are

//single precision floating point constants and not double precision

//constants.

//You should fill in the kernel as well as set the block and grid sizes

//so that the entire image is processed.

#include "reference\_calc.cpp"

#include "utils.h"

#include <stdio.h>

\_\_global\_\_

void rgba\_to\_greyscale(const uchar4\* const rgbaImage,

unsigned char\* const greyImage,

int numRows, int numCols)

{

//TODO

//Fill in the kernel to convert from color to greyscale

//the mapping from components of a uchar4 to RGBA is:

// .x -> R ; .y -> G ; .z -> B ; .w -> A

//

//The output (greyImage) at each pixel should be the result of

//applying the formula: output = .299f \* R + .587f \* G + .114f \* B;

//Note: We will be ignoring the alpha channel for this conversion

//First create a mapping from the 2D block and grid locations

//to an absolute 2D location in the image, then use that to

//calculate a 1D offset

int tx = threadIdx.x + blockIdx.x \* blockDim.x;

int ty = threadIdx.y + blockIdx.y \* blockDim.y;

int k = tx + ty \* numCols;

greyImage[k] = rgbaImage[k].x \* 0.299f + rgbaImage[k].y \* 0.587f + rgbaImage[k].z \* 0.114f;

}

void your\_rgba\_to\_greyscale(const uchar4 \* const h\_rgbaImage, uchar4 \* const d\_rgbaImage,

unsigned char\* const d\_greyImage, size\_t numRows, size\_t numCols)

{

//You must fill in the correct sizes for the blockSize and gridSize

//currently only one block with one thread is being launched

int block\_width = 16;

int block\_height = 16;

int grid\_width = numCols / block\_width;

if (grid\_width \* block\_width < numCols)

grid\_width++;

int grid\_height = numRows / block\_height;

if (grid\_height \* block\_height < numRows)

grid\_height++;

const dim3 blockSize(block\_width, block\_height ); //TODO

const dim3 gridSize( grid\_width, grid\_height ); //TODO

rgba\_to\_greyscale<<<gridSize, blockSize>>>(d\_rgbaImage, d\_greyImage, numRows, numCols);

cudaDeviceSynchronize(); checkCudaErrors(cudaGetLastError());

}

