1. *// Homework 1*
2. *// Color to Greyscale Conversion*
4. *//A common way to represent color images is known as RGBA - the color*
5. *//is specified by how much Red, Green, and Blue is in it.*
6. *//The 'A' stands for Alpha and is used for transparency; it will be*
7. *//ignored in this homework.*
9. *//Each channel Red, Blue, Green, and Alpha is represented by one byte.*
10. *//Since we are using one byte for each color there are 256 different*
11. *//possible values for each color.  This means we use 4 bytes per pixel.*
13. *//Greyscale images are represented by a single intensity value per pixel*
14. *//which is one byte in size.*
16. *//To convert an image from color to grayscale one simple method is to*
17. *//set the intensity to the average of the RGB channels.  But we will*
18. *//use a more sophisticated method that takes into account how the eye*
19. *//perceives color and weights the channels unequally.*
21. *//The eye responds most strongly to green followed by red and then blue.*
22. *//The NTSC (National Television System Committee) recommends the following*
23. *//formula for color to greyscale conversion:*
25. *//I = .299f \* R + .587f \* G + .114f \* B*
27. *//Notice the trailing f's on the numbers which indicate that they are*
28. *//single precision floating point constants and not double precision*
29. *//constants.*
31. *//You should fill in the kernel as well as set the block and grid sizes*
32. *//so that the entire image is processed.*
34. #include "reference\_calc.cpp"
35. #include "utils.h"
36. #include <stdio.h>
38. \_\_global\_\_
39. void rgba\_to\_greyscale(const uchar4\* const rgbaImage,
40. unsigned char\* const greyImage,
41. int numRows, int numCols)
42. {
43. *//Fill in the kernel to convert from color to greyscale*
44. *//the mapping from components of a uchar4 to RGBA is:*
45. *// .x -> R ; .y -> G ; .z -> B ; .w -> A*
46. *//*
47. *//The output (greyImage) at each pixel should be the result of*
48. *//applying the formula: output = .299f \* R + .587f \* G + .114f \* B;*
49. *//Note: We will be ignoring the alpha channel for this conversion*
51. *//First create a mapping from the 2D block and grid locations*
52. *//to an absolute 2D location in the image, then use that to*
53. *//calculate a 1D offset*
54. int c = (blockIdx.x \* blockDim.x) + threadIdx.x;
55. int r = (blockIdx.y \* blockDim.y) + threadIdx.y;
57. uchar4 rgba = rgbaImage[r \* numCols + c];
58. float channelSum = .299f \* rgba.x + .587f \* rgba.y + .114f \* rgba.z;
59. greyImage[r \* numCols + c] = channelSum;
60. }
62. void your\_rgba\_to\_greyscale(const uchar4 \* const h\_rgbaImage, uchar4 \* const d\_rgbaImage,
63. unsigned char\* const d\_greyImage, size\_t numRows, size\_t numCols)
64. {
65. *//You must fill in the correct sizes for the blockSize and gridSize*
66. *//currently only one block with one thread is being launched*
67. const dim3 blockSize(16, 16, 1);
68. const dim3 gridSize((numCols/16) + 1, (numRows/16) + 1, 1);
69. rgba\_to\_greyscale<<<gridSize, blockSize>>>(d\_rgbaImage, d\_greyImage, numRows, numCols);
70. cudaDeviceSynchronize(); checkCudaErrors(cudaGetLastError());
71. }