CIS 441/551: Project #1G

Gray Scale Shading and Edge Detection with CUDA

Instructions: You will be converting an image to grey scale and applying an edge detection filter to your output using CUDA 10.

1. You can use the skeleton code provided to you, called project1G.cu
2. You will be responsible for handling all memory allocations to and from the device. Don't forget to free memory!
3. You will also be responsible for writing the grey scale shader. You must use the correct luminosity equation (see below)
4. Comments are provided to you in the code to help you out. Read over them before you begin typing. You can change any code you wish, as long as your image is the same, **BUT** you only need to add code and the comments will tell you where to place certain things, ensuring that you have the correct order of operations (this is very important).
5. You should not expect a pixel perfect image. Expect the reference image to match by eye.
6. After you have successfully created the greyscale image you will want to start working on the Sobel Operator. The x and y kernels that you should use are already provided to you. You will implement the correct math operation using the correct indexes. You will have to do a double for loop to get this working. The rest is provided for you. Figuring out the right indexing is key here.
7. You will turn in a tarball with your code. Your code should generate two images: a greyscale image and a Sobel image.

Hints: Where to start? There is a lot of documentation around for cuda. I would suggest searching the internet and find a source that makes sense to you (sometimes it just takes another source). Some helpful links are Nvidia's documentation and The Supercomputing Blog. Both these resources contain all the information needed to complete the task.
Luminosity: The human eye does not see all colors with the same magnitude. The center of our color vision is near yellow-green, and colors closer to that will appear brighter. Thus we can't just average the RGB colors and get an accurate looking image (try it). The correct formula to use is

$$L = 0.21 * Red + 0.72 * Green + 0.07 * Blue \tag{1}$$

You can see here that we have higher preferences to green and the lowest preference to blue, just like in the natural world.
Sobel Filter: You will implement an edge detection algorithm on the grayscale image produced in part one. The sobel filter is a pair of masks (or matrices), X and Y, that detect horizontal and vertical edges respectively.

$$X = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$Y = \begin{bmatrix} -1 & -2 & 1 \\ 0 & 0 & 0 \\ -1 & 2 & 1 \end{bmatrix}$$

The filter works by convolving both the horizontal edge detection mask and vertical edge detection mask pixelwise over the input image to produce estimations for the gradient of the image around a pixel in the x and y directions.

The final transformed pixel value is based on the magnitude of the gradient vector. If the magnitude is greater than a user defined threshold value, then the final transformed pixel value is 255, otherwise it is 0. After convolving the masks over all pixels the output image will contain only edge information. Your task is to implement the sobel operator via an additional kernel in your program.

Andrew Ng's deep learning course provides a good visualization of the edge detection algorithm, here. In his example, he uses slightly different values for his vertical and horizontal masks, do not worry about that, stick with the masks shown above. This is another good resource showing how to compute the convolution of a matrix over an image patch. If you have any questions feel free contact Amnay at aamimeur@cs.uoregon.edu.

Compiling: Compile with the command

```
nvcc -o project1G{,.cu} `pkg-config --cflags --libs opencv`
```

This command will work on Ubuntu systems. If you are using another system then you will need to add the same cflags and libs that opencv requires. Note that we are using the back-tick, ‘, which is (probably) located at the top left of your keyboard, left of the 1 key.

Getting a GPU: If you don't have access to a GPU please email swalton2@uoregon.edu with the title "Alaska Access: 441" and an account will be created for you and instructions will be provided on how to access the GPU.

Checking with CPU: If you decide to check your result with a CPU you will not get a pixel perfect representation. It is normally a good idea to check results like this, but your images will have differences unless you take into account FMA instructions (on the GPU). More documentation can be found in the cuda documentation. You can also email Steven if you need to edit your code to perform this check.