ECE8780: Assignment 2

TA: David Langbehn
Instructor: Dr. Melissa C. Smith

February 2020

1 Problem Definition

In this lab, you'll be implementing a Gaussian Blur algorithm. Gaussian blur is an example of a **convolution** operation where the *kernel* is Gaussian. The kernel is given by the following equation:

$$G(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1}$$

In Equation 1 σ is a fixed constant and is generally treated as input to the algorithm. Visually, the convolution operation with a filter generated using equation above results in a *smooth* image with reduced visual detail. Algorithmically, there are three steps to the procedure:

```
Algorithm 1 Gaussian Blurring Algorithm
1. Separate the input RGBA image into three channels R, G, B. This is a simple map operation from a uchar4 pixel to unsigned char.
2. Apply Gaussian blur filter to each channel by convolving a region of the image with the filter. The filter is provided in code.
3. Recombine the blurred channel into one single uchar4 image. This is the reverse operation of the first one and is called a gather operation.
```

This lab involves significantly more effort on your part. The necessary components for proper working of this lab are described in the next section. For your reference, you may want to look up the convolution operation on wikipedia or opencv website.

2 Code Deliverables

You are provided with a `problem_set_2.zip` attachment. You're to implement the following kernels/piece of code:

- `Gaussian Blur` kernel: You must write the convolution kernel taking care of boundary conditions.
- `Separate Channels` kernel: You're given a colored RGBA image. You need to separate these colors into respective R,G,B channels.
- `Recombine Channels` kernel: Once you obtain the necessary blurred channels, you need to recombine them into a proper RGBA image for output.
- `main.cpp`: You need to setup *any* data copies that you want on the device. Take care that the output image to be written is a `uchar4` image.

3 Report Deliverables

Once you've completed the implementation of the kernels and the data copies you need to profile your code on *at least* two different GPU hardware and *at least* three different image sizes. For full credit you need to profile the following parameters:

- `Memcpy` times: Profile how much time it took to copy the data over to the GPU.
- Kernel Execution Time: Profile the kernel execution time of **all three kernels**.
- Effect of varying threads per block: Profile if varying the number of threads per block from $s \in [16, 1024]$ affects the kenrel run times.

This data **must be** compiled in form of either neat tables or visual graphs. Screenshots will not be provided with any credit. Feel free to reach out to the TA on his email or meet him during his office hours.