ECE 6397: Parallel Algorithms for GPUs
David Mayerich, Assistant Professor
Office: Engineering Building 2, W330
Email: mayerich@uh.edu

UNIVERSITY of HOUSTON
CULLEN COLLEGE of ENGINEERING

Programming Assignment 1: 2D Convolution of Images

Your first programming assignment is the implementation of a 2D Gaussian image convolution algorithm. Your program should work on very large images with very large kernels. Several images are provided on the course website (http://stim.ee.uh.edu/education/ece-6397-gpu-programming/).

A two-dimensional convolution on a continuous function can be specified as:

$$I(x,y) * K = \int_{u=-\infty}^{\infty} \int_{v=-\infty}^{\infty} K(u,v) \cdot I(x-u,\ y-v)du\ dv$$

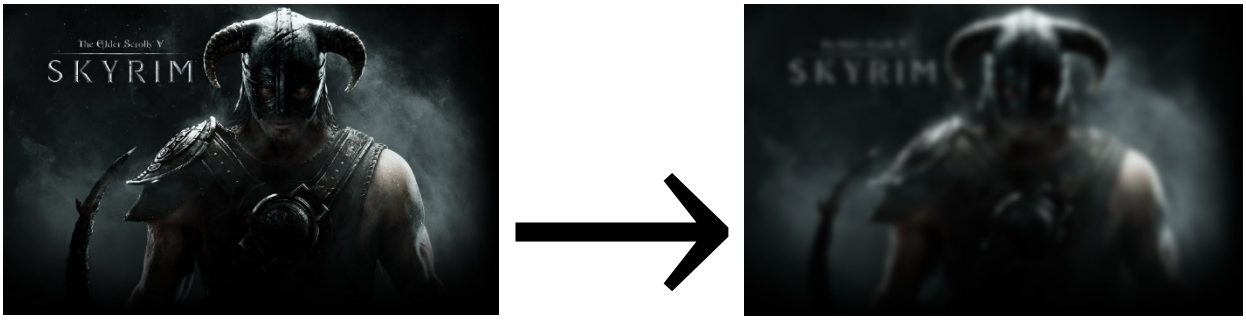
where $I(x,y)$ is a two-dimensional continuous signal and K is a two-dimensional continuous kernel. In the discrete case, the convolution can be specified as:

$$I(x,y) * K = \sum_{u=-\frac{k}{2}}^{\frac{k}{2}} \sum_{v=-\frac{k}{2}}^{\frac{k}{2}} K[u,v] \cdot I(x-u, y-v)$$

where $I(x,y)$ is now a discrete image of size $m \times n$: $I \in \mathbb{R}^{m \times n}$ and $K \in \mathbb{R}^{k \times k}$ is the kernel, where k is the size of the kernel along one dimension. We are assuming that K is square. You can apply a blur kernel to I by specifying K as a two-dimensional Gaussian function:

$$K(u,v) = \frac{1}{2\sigma^2\pi} e^{-\left(\frac{(u-\mu)^2+(v-\mu)^2}{2\sigma^2}\right)}$$

where u and v are the corresponding (x,y) coordinates in the kernel where both u and v are constrained by: $-\frac{k}{2} \leq u \leq \frac{k}{2}$. Convolving an image with this function will result in a blurred version of that image:



In general, a convolution can be specified using any number of kernel functions. For example, another common filter is the box filter:

$$K(u,v) = \frac{1}{k^2}$$

which is often used for image down-sampling. However, the Gaussian blur is commonly used because it has several useful properties. In particular, the Fourier transform of a Gaussian function is also a Gaussian. This is important because it becomes easy to calculate which frequencies in the original signal are attenuated and by how much. In addition, the Gaussian function is *separable*. This means that the convolution can be separated into two independent one-dimensional convolutions:

$$K = K_0 \cdot K_1 = \frac{1}{\sqrt{2\sigma^2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \cdot \frac{1}{\sqrt{2\sigma^2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}} = \frac{1}{2\sigma^2\pi} e^{-\left(\frac{(x-\mu)^2+(y-\mu)^2}{2\sigma^2}\right)}$$

Breaking a single multidimensional Gaussian convolution into smaller one-dimensional convolutions can significantly improve performance. For example, the computation time for a 2D convolution can be specified as $O(nk^2)$ where n is the number of pixels in the image and k is the size of the kernel (again assuming a square kernel). Reducing this to two 1D convolutions yields a computation time of $O(2nk)$, which can provide a dramatic performance increase.

In this programming assignment, you will implement a 2D Gaussian convolution that takes as input parameters the image filename and standard deviation of the convolution kernel:
>> convolve imageFile.ppm 40

For example, the above command will convolve an image named "imageFile.ppm" with a 2D Gaussian kernel with standard deviation $\sigma = 40$ pixels. The mean value for the Gaussian distribution within the kernel will always be $\mu = 0$.

Command line arguments are passed to your C/C++ main( ) function as text strings by the operating system. Reading them is straightforward:

```
#include <string>
int main(int argc, char* argv[]){
      if(argc != 3)                  //there should be three arguments
            return 1;                //exit and return an error
      std::string filename(argv[1]);
      int sigma = atoi(argv[0]);
      …
}
```

In addition, images can be read using the Netpbm PPM image format. This format is easy to read and the specification is provided on the Wikipedia page:

https://en.wikipedia.org/wiki/Netpbm_format

In short, the first few lines of text contain the image parameters, including image size, and the rest of the file stores the raw pixel data as a series of binary `unsigned char` values. Three values are stored per pixel, representing red, green, and blue color values. You can read, convert, and display images in PPM format using GIMP: https://www.gimp.org/

Your program will perform both a CPU and GPU based convolution (both should be implemented as separable filters). Output both (1) the result of the GPU based algorithm as an image and (2) the time required to calculate both the CPU and GPU algorithms. **Make sure that you handle edge cases robustly and comment your code!**

ECE 6397 – GPU Programming
Programming Assignment 1: Two-Dimensional Gaussian Image Convolution
Rubric

Name_____

Correctly reading input and producing output are crucial for programming assignments. Code that crashes without producing output or code that doesn't compile are not considered to produce results and are graded accordingly in the rubric.

Note: If you choose to turn in code that does not complete the assignment, partial credit is available in the implementation section IF YOU NOTIFY ME. Use the console (cout/printf) to describe the problem and use liberal comments in your source file to make grading easy. I will be going through it. Even better, see me before the assignment is due so we can discuss any problems preemptively.

CPU implementation **_____ / 30**

correct result (code compiles and executes)	_____ / 10
implementation / correct coding	_____ / 15
profiling/timing results	_____ / 5

GPU implementation **_____ / 50**

correct result (code compiles and executes)	_____ / 10
implementation / correct coding	_____ / 35
profiling / timing results	_____ / 5

comments (I expect about 1-2 lines per line of code) **_____ / 10**
(also note that comments help me understand where you are having difficulty and make grading easier – particularly when the code doesn't work)

profiling (timing results) **_____ / 10**