Lab 5: Convolution

Objective

The objective of this lab is to implement a tiled image convolution using both shared and constant memory.

Instructions

In this lab, we will implement of a 2D image convolution algorithm. We will have a constant convolution mask (of size kxk, where k is an odd number), but will have arbitrarily sized images. Assume the image dimensions are greater than kxk for this lab. Several images are provided on the data directory for our class on Rosie (`/data/cs4981_gpu_programming/lab-5`).

Convolution is used in many fields, such as image processing for image filtering. A standard image convolution formula for a kxk convolution filter `M` with an Image `I` can be specified as:

$$P_{i,j,c} = \sum_{u=-\frac{k}{2}}^{\frac{k}{2}} \sum_{v=-\frac{k}{2}}^{\frac{k}{2}} M_{u,v} \cdot I_{i+u,j+v,c}$$

where $P_{i,j,c}$ is the output pixel at location `i,j` in channel `c`, $I_{i,j,c}$ is the input pixel at location `i, j` in channel `c` (the number of channels will always be 3 for this lab corresponding to the RGB values), and $M_{u,v}$ is mask value at location `u,v`. We are assuming that M is square. You are encouraged to experiment with different filter types but should report results of performing convolution with at least a simple box blur filter with values $\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$ and an edge detection filter with values $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$.

Your program should take as input parameters the input image filename and the filenames of the output images for the CPU implementation and the GPU implementation:

```
>> ./lab5 input-image.ppm cpu-output.ppm gpu-output.pmm
```

For example, the above command will convolve an image named `input-image.ppm` with a 2D kernel and will output two image files, one from the sequential convolution implementation (`cpu-output.ppm`) and one from the parallel convolution implementation (`gpu-output.ppm`).

Input Data

The input is an interleaved image of `height x width x channels`. By interleaved, we mean that the element `I[y][x]` contains three values representing the RGB channels. This means that to index a particular element's value, you will have to do something like:

```
        index = (yIndex*width + xIndex)*channels + channelIndex;
```

For this assignment, the channel index is `0` for R, `1` for G, and `2` for B. So, to access the G value of `I[y][x]`, you should use the linearized expression `I[(yIndex*width+xIndex)*channels + 1]`. For simplicity, you can assume that channels are always set to `3`.

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

Sombrero Galaxy

Whirlpool Galaxy

Hereford Cathedral

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/. PPM file specifications:

- The first line of the file is a line of text saying "P6"
    - indicates that this is a binary ppm (color image) file
- The second line of the file is a comment starting with "#"
    - in general, there can be any number of comment lines – test the first character
- The next line of the file has two integers (in ASCII)
    - these specify the width and height of the image
- The last text line of the file has one integer (in ASCII)
    - this specifies the maximum intensity value – 255 for all sample images
- The rest of the file stores pixel data in binary
    - Each pixel is in lexicographic (row-major) order
    - The data type for each value is an `unsigned char`
    - The data is stored in RGB triples

Your program will perform both a CPU and GPU based convolution. Output both (1) the result of the GPU based algorithm as an image, (2) the result of the CPU based algorithm as an image, and (3) the time required to calculate both the CPU and GPU algorithms. Make sure that you handle edge cases robustly and comment your code.

Benchmarking

Time both your CPU and GPU convolution implementations with each image provided in the data folder. Using your favorite plotting tool, plot the runtimes in milliseconds. Your plot should have an abbreviation of the image name in the x-axis and runtimes for both the CPU and GPU implementations on the y-axis. Run the convolution with images in this order: hereford_256.ppm, hereford_512.ppm, bansberia.ppm, sombrero_galaxy.ppm, hereford.ppm, and whirlpool.ppm. Also, record the speedups obtained, computed as ratios of CPU/GPU execution times. Run your benchmarks multiple times and report average runtimes.

Questions

Answer the following questions.

1. Name 3 applications of convolution.
2. How many floating operations are being performed in your convolution kernel? Explain.
3. How many global memory reads are being performed by your kernel? Explain.
4. How many global memory writes are being performed by your kernel? Explain.
5. What is the minimum, maximum, and average number of real operations that a thread will perform? Real operations are those that directly contribute to the final output value.
6. What do you think happens as you increase the mask size (say to 1024) while you set the block dimensions to 16x16? What do you end up spending most of your time doing? Does that put other constraints on the way you'd write your algorithm (think of the shared/constant memory size)?
7. What is the identity mask?

Submission Instructions

You should upload a PDF of the answers to the above questions, plot of runtimes, original image used, the convolved image using the CPU implementation, the convolved image using the GPU implementation, source code, and the binary executable to Canvas as part of a zipped folder named as **your_msoe_username-lab5.zip**.

Pseudocode

A sequential pseudo code would look something like this:

```
maskWidth := k
maskRadius := maskWidth/2 # this is integer division

for i from 0 to height do

    for j from 0 to width do

        for k from 0 to channels

            accum := 0

            for y from -maskRadius to maskRadius do

                for x from -maskRadius to maskRadius do

                    xOffset := j + x

                    yOffset := i + y

                    if xOffset >= 0 && xOffset < width &&

                       yOffset >= 0 && yOffset < height then

                       imagePixel := I[(yOffset * width + xOffset) * channels + k]

                       maskValue := K[(y+maskRadius)*maskWidth+x+maskRadius]

                       accum += imagePixel * maskValue

                    end

                end

            end

             # pixels are in the range of 0 to 255
             P[(i * width + j)*channels + k] = clamp(accum, 0, 1)
        end
    end
end
```

where clamp is defined as

```
def clamp(x, lower, upper)

      return min(max(x, lower), upper)

end
```

Rubric

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
Documentation		**___________ / 10**
Questions		**___________ / 10**