ECE453 Project 1

Richard Lee

Introduction

The goal of this project was to become familiar with CUDA C toolchain and how to use it to perform highly parallelizable tasks, such as image processing. The given task was to deblur a provided image (Figure 1a).

(a) Blurry Image

(b) Clear, Reference Image

Figure 1: Provided Images of size 960x960 Pixels

Sharpening Algorithm: Laplacian Kernel Image Sharpening

Image sharpening is the process of highlighting or enhancing detail in an image [1]. One form of sharpening involves increasing contrast. By increasing the difference in intensity between light and dark regions, the image can appear to have more detail than originally present. While the resulting image may appear to be clearer may be a less accurate representation of the original subject. As the specific algorithm was not the primary focus of this project, the specific algorithm achieves the desired deblurring, but may not be the algorithm that achieves the highest perceived sharpness.

We can measure differences intensity by finding the discrete difference between a pixel and its neighbors, which is equivalent to computing a derivative:

$$\frac{\partial f}{\partial x} = f(x+1, y) - f(x, y) \tag{1}$$

However, first-order derivatives are subject to noise and any minor pixel fluctuations. Thus, second derivatives, which use neighbors on both sides of the pixel of interest, are computed instead:

$$\frac{\partial^2 f}{\partial x^2} = \frac{\partial f}{\partial x_2} - \frac{\partial f}{\partial x_1} \tag{2}$$

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) - f(x, y) - (f(x, y) - f(x-1, y)) \tag{3}$$

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y) \tag{4}$$

Since images are two dimensional, we can compute the derivative in the y-direction, in the same way, looking at the neighboring pixels in the y-direction. This yields the final form:

$$\nabla^2 f = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y) \tag{5}$$

We can write this in the form of a mask, known as a Laplacian Mask:

$$L_m = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{6}$$

Performing a 2d convolution with the above mask yields an image with high intensity in regions of high contrast: in other words, the edges in this intermediate images are highlighted. From there, we can subtract this image from the original image to yield a sharpened version of the original image:

$$\text{sharpened image} = \text{image} - \text{Laplace(image)} \tag{7}$$

$$\text{sharpened image} = \text{image} - L_m * \text{image} \tag{8}$$

$$\text{sharpened image} = \text{image} * (1 - L_m) \tag{9}$$

$$\text{sharpened image} = \text{image} * (1 - \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}) \tag{10}$$

$$\text{sharpened image} = \text{image} * (\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}) \tag{11}$$

$$\text{sharpened image} = \text{image} * \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{12}$$

where $*$ denotes a two dimensional convolution across the image. We can extend this to larger masks, such as 5x5 or 7x7, the latter of which was used in for most of the sharpening and testing in this project:

$$\text{sharpened image} = \text{image} * \begin{bmatrix} 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & -1 & -1 & 13 & -1 & -1 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \end{bmatrix} \tag{13}$$

Implementation

Image Loading/Writing

In order to process the provided images, I first had to load them into memory. I considered using the most popular image libraries (libjpeg and libpng), but I wanted a single library that could handle both, and potentially even more, image formats. In the end I chose the portable stb_image, distributed with the stb library collection. The included `stb_image.h` allows for a single format-agnostic function call to load an image as a channels-last 1D array of `unsigned char` (for the PNG, I ignored the alpha channel, reading in only the RGB data). This was in stark contrast to the significant additional code that would be needed for the more established libjpeg or libpng.

For the writing the image to file, I used the `stb_image_write.h` in the same stb collection. Again, this was a single function call that writes an `unsigned char` array to file, although different function calls are needed for different formats.

Effectiveness Metric

I considered Sharpness to be a good metric to measure Effectiveness, so I used a method of measuring sharpness that could be used to estimate relative sharpness. In order to measure sharpness, I implemented a non-vectorized C version of the python code provided by @Robert Pollak on Stack Overflow [2]. I tested both the `np.gradient` and `np.diff` versions and found them to both report similar values, so I implemented the similar `np.diff` version, which did not require calculating second order gradients.

Error Measurement

In order to compute error, I implemented a function that looped through all the pixels of the image in question and a reference, calculating the sum of the mean squared errors. The function reports the error as a percentage.

Miscellaneous Implementation Details

In order to make testing other images easier, I used `libpopt`, which allowed for convenient argument parsing. If the program is called without any arguments, it will default to the provided "nasa_earth_blurry.png" for the blurry image and "nasa1R3107_1000x1000.jpg" as the reference image, and it will run the sharpening algorithm on both the cpu and gpu. However, the user can specify arguments to change this behavior. More detailed program usage can be found in the README of the project code or with the `--help` option passed to the program binary.

Results

Program Output for Provided Images

```
No arguments specified, defaulting to:
Kernel File:    kernel7x7.txt
Input Image:    nasa_earth_blurry.png
Ref Image:      nasa1R3107_1000x1000.jpg
Output Prefix:  output
Device Name:    both
Kernel Size:    7 by 7
Deconvolution kernel:
  0.00    0.00    0.00   -1.00    0.00    0.00    0.00
  0.00    0.00    0.00   -1.00    0.00    0.00    0.00
  0.00    0.00    0.00   -1.00    0.00    0.00    0.00
 -1.00   -1.00   -1.00   13.00   -1.00   -1.00   -1.00
  0.00    0.00    0.00   -1.00    0.00    0.00    0.00
  0.00    0.00    0.00   -1.00    0.00    0.00    0.00
  0.00    0.00    0.00   -1.00    0.00    0.00    0.00
Loaded nasa_earth_blurry.png with width of 960px, a height of 960px and 3 channels
Max Threads Per Block for device 0: 1024
CUDA kernel launch with 31 by 31 blocks of 32 by 32 threads
Elapsed time for GPU: 42.573250 ms
Elapsed time for CPU: 723.389709 ms
Loaded nasa1R3107_1000x1000.jpg with width of 960px, a height of 960px and 3 channels

Effectiveness (Sharpness) Metrics (Higher is Estimated to be Sharper):
Sharpness value of nasa1R3107_1000x1000.jpg                              0.055519
Sharpness value of nasa_earth_blurry.png                                 0.009995
Sharpness value of GPU kernel output                                     0.036856
Sharpness value of CPU kernel output                                     0.036856

Error Computations:
Percent MSE between nasa_earth_blurry.png and nasa1R3107_1000x1000.jpg  1.9759%
Percent MSE between GPU kernel output and nasa1R3107_1000x1000.jpg      2.3418%
Percent MSE between CPU kernel output and nasa1R3107_1000x1000.jpg      2.3418%
```

Listing 1: Output on my personal computer

```
No arguments specified, defaulting to:
Kernel File:    kernel7x7.txt
Input Image:    nasa_earth_blurry.png
Ref Image:      nasa1R3107_1000x1000.jpg
Output Prefix:  output
Device Name:    both
Kernel Size:    7 by 7
Deconvolution kernel:
  0.00    0.00    0.00   -1.00    0.00    0.00    0.00
  0.00    0.00    0.00   -1.00    0.00    0.00    0.00
  0.00    0.00    0.00   -1.00    0.00    0.00    0.00
 -1.00   -1.00   -1.00   13.00   -1.00   -1.00   -1.00
  0.00    0.00    0.00   -1.00    0.00    0.00    0.00
  0.00    0.00    0.00   -1.00    0.00    0.00    0.00
  0.00    0.00    0.00   -1.00    0.00    0.00    0.00
Loaded nasa_earth_blurry.png with width of 960px, a height of 960px and 3 channels
Max Threads Per Block for device 0: 1024
CUDA kernel launch with 31 by 31 blocks of 32 by 32 threads
Elapsed time for GPU: 436.028076 ms
Elapsed time for CPU: 2835.438721 ms
Loaded nasa1R3107_1000x1000.jpg with width of 960px, a height of 960px and 3 channels

Effectiveness (Sharpness) Metrics (Higher is Estimated to be Sharper):
Sharpness value of nasa1R3107_1000x1000.jpg                              0.055519
Sharpness value of nasa_earth_blurry.png                                 0.009995
Sharpness value of GPU kernel output                                     0.036856
Sharpness value of CPU kernel output                                     0.036856

Error Computations:
Percent MSE between nasa_earth_blurry.png and nasa1R3107_1000x1000.jpg  1.9759%
Percent MSE between GPU kernel output and nasa1R3107_1000x1000.jpg      2.3418%
Percent MSE between CPU kernel output and nasa1R3107_1000x1000.jpg      2.3418%
```

Listing 2: Output on the provided Jetson platform

Development and preliminary testing was conducted on my local machine running an AMD Ryzen 1700 with an Nvidia GTX 1080 (output in Listing 1), and final testing was conducted on the provided Jetson platform (output in Listing 2).

In both cases, the GPU implementation of the sharpening algorithm offered improvement over the CPU implementation, with my personal computer achieving an approximate 17x performance boost and the Jetson achieving an approximate 6.5x performance boost. On both platforms, the sharpness and mean squared error computation yield the exact same values.

There are not physically meaningful units to the sharpness metric; however, they can be used for relative comparison: the original reference image was the sharpest with a value of 0.0555, the blurred was far less sharp with a value of 0.0099, and the deblurred had a sharpness value of 0.0369, which indicates it is not quite as sharp as the original, but still an improvement over the blurred image.

Image Output for Provided Images

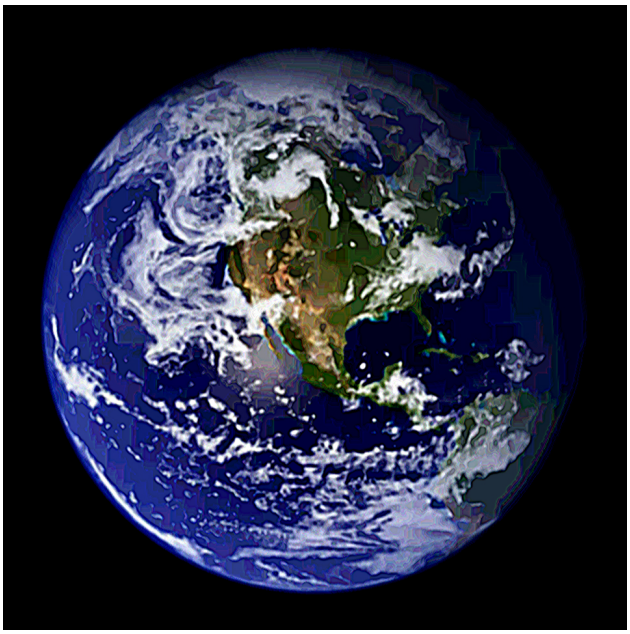

Figure 2: Output of the image after sharpening. This image is the output from the GPU, but the CPU image is equivalent.

(a) Blurry Image

(b) Clear, Reference Image

Figure 3: Provided Images of size 960x960 Pixels.

The details in the blurry picture are accentuated to give the appearance of increased sharpness; however, this detail is obtained from the blurry image, which is not necessarily an accurate representation of the fine details in the original image. This results in the sharpened output to be farther from the original than the

blurry image, as indicated by the error percentages in the previous section (the blurred image had a 1.9759% error while the sharpened output had a 2.34% error).

Program and Image Output for Another Test Image

In order to test the program functionality, I tried testing it on a different, larger image to see the result. The program output can be seen below, in Listing 3. The Jetson acheived an approximate 8x performance boost performing the sharpening on the GPU over the CPU. Like the provided image, the final error of the sharpened image is higher than that of the blurred image. The sharpened image can be seen in Figure 6.

```
Kernel Size:    7 by 7
Deconvolution kernel:
  0.00    0.00    0.00   -1.00    0.00    0.00    0.00
  0.00    0.00    0.00   -1.00    0.00    0.00    0.00
  0.00    0.00    0.00   -1.00    0.00    0.00    0.00
 -1.00   -1.00   -1.00   13.00   -1.00   -1.00   -1.00
  0.00    0.00    0.00   -1.00    0.00    0.00    0.00
  0.00    0.00    0.00   -1.00    0.00    0.00    0.00
  0.00    0.00    0.00   -1.00    0.00    0.00    0.00
Loaded tree-blur.jpg with width of 7680px, a height of 4320px and 3 channels
Max Threads Per Block for device 0: 1024
CUDA kernel launch with 241 by 136 blocks of 32 by 32 threads
Elapsed time for GPU: 12241.376953 ms
Elapsed time for CPU: 102155.914062 ms
Loaded tree-clear.jpg with width of 7680px, a height of 4320px and 3 channels

Effectiveness (Sharpness) Metrics (Higher is Estimated to be Sharper):
Sharpness value of tree-clear.jpg                                    0.192569
Sharpness value of tree-blur.jpg                                     0.019065
Sharpness value of GPU kernel output                                 0.124489
Sharpness value of CPU kernel output                                 0.124489

Error Computations:
Percent MSE between tree-blur.jpg and tree-clear.jpg           3.5979%
Percent MSE between GPU kernel output and tree-clear.jpg       5.7243%
Percent MSE between CPU kernel output and tree-clear.jpg       5.7243%
```

Listing 3: Program output for 7680x4320 pixel image on the provided Jetson.

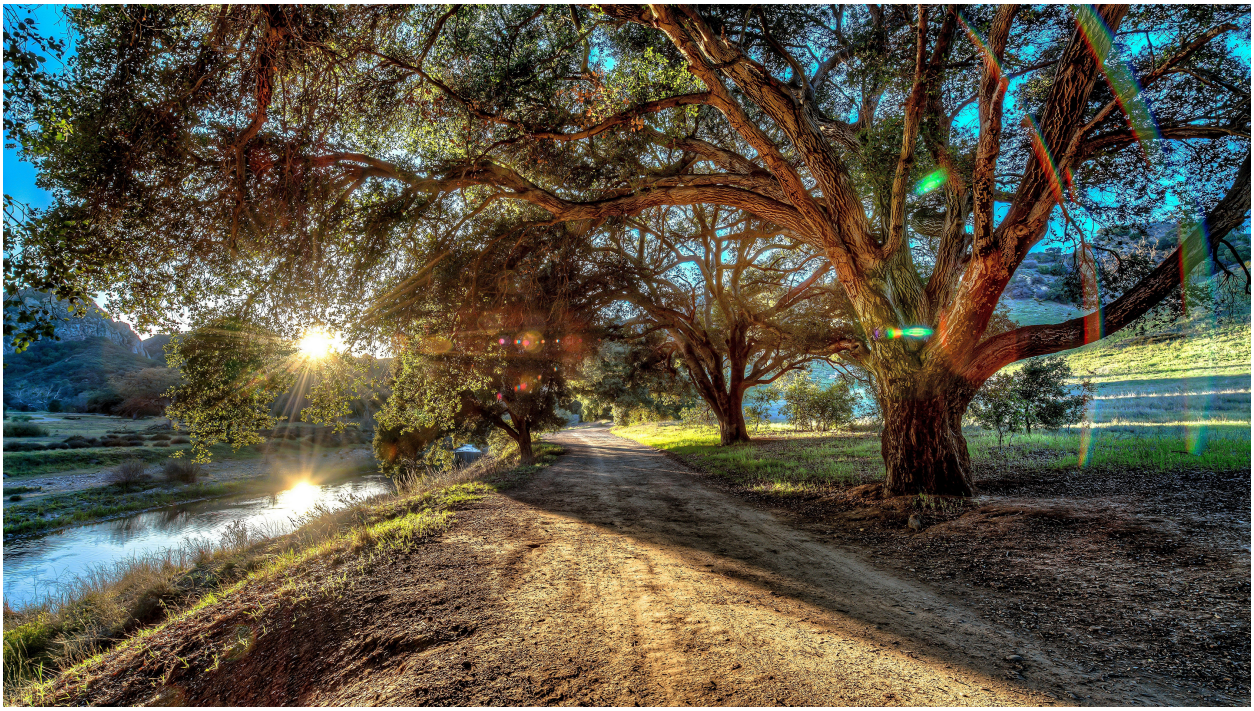

Figure 4: Original image of 7680x4320 pixels [3].

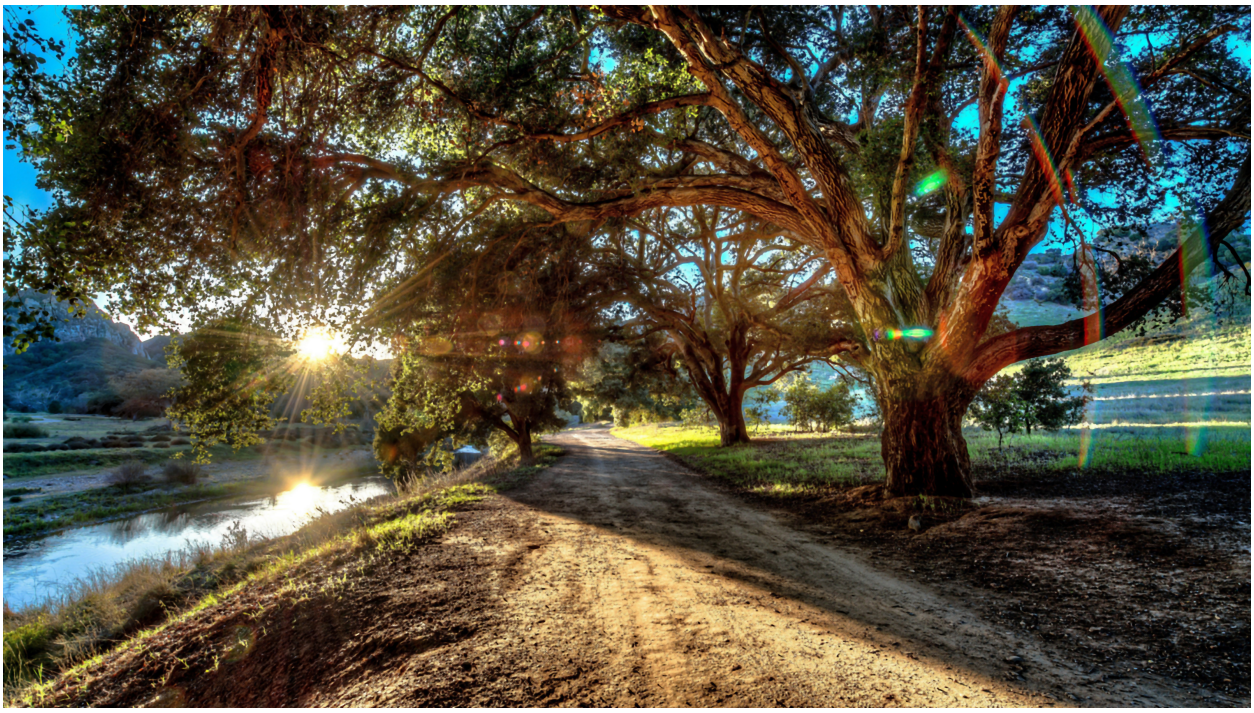

Figure 5: Blurred Image using a Median blur.

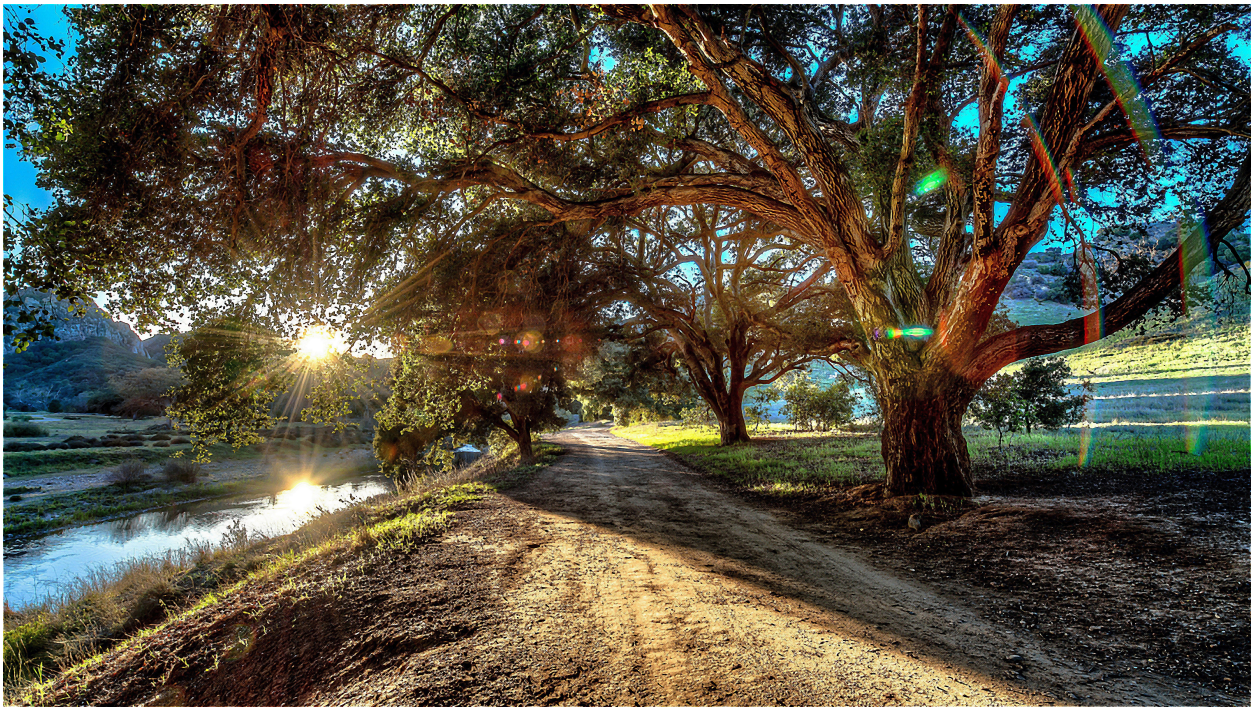

Figure 6: Sharpened image output from the GPU.

Conclusions

The program was relatively successful at sharpening the images as well as demonstrating the performance boost that is afforded by using GPUs for distributed tasks like image processing. The program was also successful in showing how to use CUDA C and write a kernel function to perform operations, as opposed to the traditional loop-based CPU methods.

References

[1] Hongbing Fan. Lecture 6 sharpening filters, 2012.

[2] Robert Pollak (https://stackoverflow.com/users/1389680/robert-pollak). Detect which image is sharper. Stack Overflow. URL: https://stackoverflow.com/a/26014796.

[3] Wallpaper Access (https://wallpaperaccess.com/download/8k-11750). 8k wallpapers, 2021.