**332:451 Final Project**

**Image Processing**

Eric Gilbert

Gregory Paton

30 December 2012

**Description**

For our project we have created programs to perform various image processing tasks. We’ve implemented resize, blur, and graphical drawing for bitmap images. All these tasks require processing a fair amount of data. Even a relatively small 512x512 bitmap image has over 250,000 pixels with 3 bytes per pixel (one for each of red, green, and blue). Processing this data sequentially is very time intensive, especially when the resolution increases. The HD standard 1080p consists of over 2 million pixels and manufacturers continue to up the standard. UHD 4k resolution is four times the data, over 8 million pixels. Furthermore, when dealing with video, even more processing time is required. Even a relatively low 24 frames per second would take an infeasible amount of time to process sequentially, especially for use in streaming applications that have become popular with the rise of cloud computing. One method to solve this problem is compression which trades image quality for a smaller storage size. However, a raw image is still required for processing.

This is where parallelization comes in. By distributing the image processing work over multiple threads, the overall computation time can be reduced dramatically. By reading from the original image and storing the results in a new image, all race conditions can be avoided. This makes the process of parallelization fairly straightforward. However, the process can be further optimized by understanding how memory is physically allocated and taking advantage of temporal and spatial locality. To stay competitive in the image processing industry you must have fast programs. In this project we explored exactly how that’s done.

**Goals**

The main goal of this project is to understand how image processing algorithms can be sped up using parallelization. We chose to use CUDA as it seemed like a natural choice for out application. GPUs are typically used for 3D graphics rendering, so using one to process a single image will be no problem.

Also, memory alignment is a huge aspect of image processing. By understanding how the image data is laid out in memory is crucial to optimizing our code. Another goal of ours is to understand how this works and write our program to take full advantage of it to minimize the processing time.

**Resize**