**ECE569 Project Proposal**

**Shadow Removal**

**Cole Nelson**  
**Thai Le**  
**Enzio Velazqueoni**

[nelso680@arizona.edu](mailto:nelso680@arizona.edu)[thaile82@arizona.edu](mailto:thaile82@arizona.edu)[evelazqueoni@arizona.edu](mailto:evelazqueoni@arizona.edu)

**Group #1 on D2L**

|  |  |
| --- | --- |
| Brief (less than 200 words) technical abstract of the proposed project | With the current rise of automation, and more notably AI; computer vision, detection, and monitoring systems will see an increased utilization in our lifetime. As more systems become managed by computers such as agriculture, surveillance, and satellite imagery, the need to remove impurities in images is a natural progression. These algorithms need to be precise, and robust, but also fast to be lucrative to be applied in the places they are called in. If we can re-design a shadow removal algorithm that can operate faster while maintaining image fidelity, then we could process higher pixel count images, or process smaller resolution images quicker. The goal of this project is to be able to process these images faster, but also process images with a higher pixel count. |
| Brief (less than 200 words) discussion of anticipated benefits | There are two benefits that are expected of this project, the first being able to process shadow removal faster. The second would be to allow for images with a higher pixel to be processed in the same amount of time as the previous algorithm. |
| A maximum of eight key words that describe the project | Optimization, shadow-removal, parallel computing, Cuda-C |

# Identification and significance of the problem or opportunity

## Statement of Problem

The increase in digital content, videos/images will naturally drive an increase in automated processing algorithms to process this content. To further tackle this demand, algorithms need to be accurate and fast to be useful. Having robustness while maintaining speed will make these algorithms lucrative in the fields, they are applied in. Impurities such as shadows in an image can prove an obstacle to a typical computer vision algorithm, as pixel color changes, while retaining enough features that the shadow may be considered an object. While easily overcome by human intuition, these are novel problems for a computer algorithm.

Referencing [1], the problems addressed in this paper address the situation of contact sensing in a Controlled Environment Plant Production (CEPP). Where the need is to view plant physical characteristics to identify plant physical health. The paper mentions a situation of tip-burn (a calcium deficiency) of plants that is difficult to detect when the plant is obscured by a shadow. Shadows in a CEPP occur via clouds, light intensity changes as the day progresses, and airflow as described in the paper. With all these dynamic shadow disturbances in the plant data, removing these prior to observation is necessary to produce accurate results. [1] references two approaches to shadow removal revolve around RGB, entropy minimization, and color invariance. However, these implementations are only viable in static light-angle stations and do not address the true complexity of the problem.

The need for GPU implementations then comes in to overcome the limitations described in [1]. The introduction of parallelism streamlines the pre-processing steps, while also opening the door to real-time applications as well. Without GPU-based parallelism, there would be a significant bottleneck in preparing shadow-removed images for functional viewing. This of course is a costly situation to be in not only to develop non-GPU algorithms to achieve a fraction of the speed GPU-based algorithms can do easily. Especially as image data increases, the pixel count number goes up, further increasing the pixel computation time for serial implementations. This is made a less significant factor via parallelized threads.

## Problem Background

Shadow removal using a GPU for parallelization is necessary if many images will be processed continually and with real-time information. If the shadow removal is done serially, it will be expected to take a lot longer making it not feasible for a user to get real up-to-date information. The algorithm will be chromaticity-based due to the algorithm being the best for a high throughput and low latency. A chromatic algorithm is where a user switches the color formats to a different color space in the hope that the intensity of the lighting becomes more separatable. The more separation allows for quicker solutions for each pixel, but it does depend on the color space being switched to not being overly noisy.

## Current Level of Technology

We are going to be basing our implementations on the foundation made in [1]. The paper’s goal was to provide a low-latency, high-throughput shadow removal algorithm that implements a chromaticity-based algorithm. Which revolves around pixel-based transformation. Multiple applications for chromaticity algorithms exist that are referenced in [1]. The original paper has around five phases of steps to complete the task: Color space transformation, thresholding and Otsu’s method, convolution, erosion, and result integration. The algorithm flowchart can be seen in figure 1.

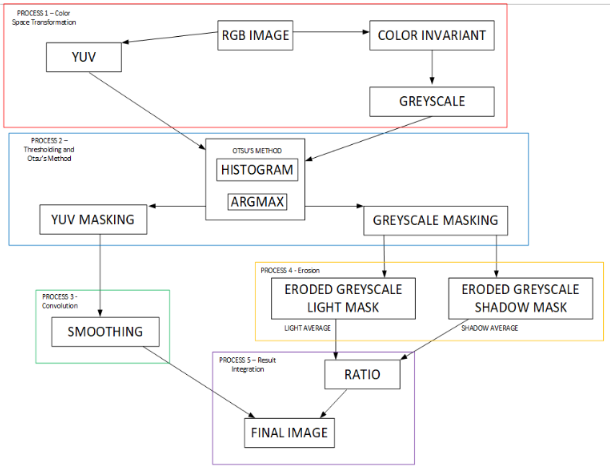


Figure 1: Algorithmic flow chart used in "Accelerated Shadow Detection and Removal Method," [1]

Serial code via MATLAB is provided alongside [1] for our reference. While we do not need to write the code from scratch, the code provided will act as a guide for our efforts until we have replicated a parallel version that achieves the same results discovered in the referenced paper. Once done our optimizations to increase processing time will be made, and then testing on larger resolution images will be performed to test if we can handle those in acceptable time.

## Limitations of Current Technology

While chromaticity is the fastest algorithm available, it is stated to be vulnerable to noise and can fail to detect darker shadows. While we intend to stick to a chromaticity-based algorithm, we are open to a different approach should something more fruitful be introduced as the class progresses. Another limitation we have is our physical GPU. Our testing will most likely be done on the Tesla P100, which only has a maximum thread count of 1024. While this should not be an issue, it is the main driver of our project, so we are forced to work within its hardware.

## Proposed Solution

Processes 1, 2, 3, and 4 will all utilize tiling to some degree. Process 1 will benefit from coalesced reads/writes. For phase three, specifically using a tiling matrix multiplication method will be tried here. Since we’ll be multiplying the summation of a selected image pixel with its neighbors against the kernel matrix, a speedup should be expected by tiling these matrix operations. For phase four we will focus on using the one-dimensional array in shared memory mentioned in [1]. For thefinal phase of results integration, because this process is originally a four-kernel implementation in [1], it may be ideal to try applying kernel fusion across these four kernels. Kernel fusion is essentially taking sequential kernels that run in a sequence one after one another, and finding where elements of these kernels can be fused [3].

### Background

A Chromaticity algorithm is a belief that the objects color is already specified and can be differentiated from how a shadow darkens an object [1]. A chromaticity algorithm should be expected to change color space such as switching to YUV. YUV is a method that is meant to separate an object's brightness from its chroma signals. The main issue that can be seen in this type of algorithm is that it may not be able to remove a shadow if it's too dark.

### Statement of Solution

Processes 1, 2, 3, and 4 will all utilize tiling to some degree. Tilling can be used as a solution to shadow removal with a Chromaticity algorithm as all pixels should undergo the same steps with a relatively similar time frame. Tiling is block loading global memory into shared and as occurs only once it will help minimize the global reads. Tiling will allow for better memory access load times as the thread will have access to shared memory which is more efficient to load than that of global memory. Tiling works by having multiple different threads pool together and request memory at the same time. This minimization of requests will lower the overall requests for loads meaning more time can be spent on computation. This will only be possible if the functions being run are similar enough and executed on time. Seeing how shadow removal is essentially repeated functions on different pixels, it fits the necessary criteria for time. Depending on what phase we are in will dictate how we implement the specific tiling. Either in shared or in global memory as well. For instance, phase three will use tiling in terms of matrix multiplication because we are multiplying a summation matrix with the kernel matrix.

Process 1 will benefit from coalesced reads/writes. The reason for the benefit will be because this is where an initial load accessing global memory of the input image will be the most beneficial and the main place it can occur at. Since the image resolution is large, many threads will be allocated to this operation. Having them accessed in out-of-order fashion will drastically slow down memory access times, especially from the global memory.

For phase four we will focus on using the one-dimensional array in shared memory mentioned in [1]. A one-dimensional array is faster as it makes it so that the data is only being read once while for the two-dimensional array has it being read twice. This should provide a speed up of 2 times faster than a two-dimensional array. It is possible to do a one-dimensional array on phase 4 as the grayscale value will only be one number per pixel rather than the three for RGB.

For thefinal phase, kernel fusion aims to optimize our writes/reads to memory. Any form of data-reuse that we can achieve at the end of one kernel that can be put into shared memory for the usage of the next will expect to see speedups here.

### How Solution Overcomes Current Technology

While the inherent chromaticity issues may persist in our work due to the nature of the algorithm, our solution should overcome block-loading throughput. By increasing the thread dimensions from 2D to 3D, we will load more data, and more threads will have more access to that data. With this transition, the approach of tiling may become more feasible as the RGB values get placed into the third dimension, allowing us to tile our row/column (first two dimensions) for streamlined calculation. Tiling we expect will reduce the global memory reads by a large factor, giving us a more effective memory utilization.

### Validation and Evaluation Strategy

To verify functionality: does our code successfully remove shadows in a manner that is on par with the source code?

To conduct performance review: Using the profiler tools, we will examine the execution time and memory usage.

The metrics for all of this of course will revolve around the existing source code and results gathered in [1]. We will base our evaluation and validation efforts entirely on using [1] as our foundation. Any performance increases/decreases will be compared against the source code.

# Technical objectives

## 2.1 Project Milestones and Tasks

For each task indicate the lead person who will be responsible for completing that task.

### 2.1.1 Milestone 1: Re-create serial source code in parallel form

* Study and understand serial code so a strong foundation of what our program is doing and how, is obtained.
* Create the CUDA C/C++ versions of each section that will run the parallelized code:
* Colorspace Transformation kerel: Thai
* Thresholding and Otsu’s Method: Thai
* Convolution: Cole
* Erosion: Enzio
* Result Integration: Enzio

### Milestone 2: Apply Optimizations to Process 1&2

Apply GPU optimization strategies to the various kernels in the code.

* Colorspace Transformation kernel: Thai
  + Applying coalesced read and write.
* Thresholding and Otsu’s Method: Thai
  + Histogram Generation Optimization
  + Argmax Optimization

### Milestone 3: Apply Optimizations to Process 3&4

Apply GPU optimization strategies to the various kernels in the code.

* Convolution
  + Tiling Matrix Multiplication
* Erosion
  + Tiling matrix, 1D array

### Milestone 4: Apply Optimizations to Process 5

Apply GPU optimization strategies to the various kernels in the code.

* Result Integration
  + Kernel Fusion

### Milestone 5: Debug and Optimize

* Run our code through the debugger
  + Catch non-obvious issues with the kernels here
    - Issues with memory, timing, things that don’t halt execution but can stand in the way of maximum performance.
* Run code through profiler
  + See how the performance is and address issues.
* Go backs or refactoring if needed will be done at this phase.

### Milestone 5: Gathering Results and Analyze

* Once code optimizations complete, run the final tests and compile results
* Compare results to the benchmarks posted in [1]
  + Does our code achieve better speedups? Similar speedups?
  + Where does our code fall short?
  + Where does our code improve?
* A visual comparison of the images will be performed as well.
  + We can verify the RGB space of the output image with the original image. Should there be a consistent fill-in of the shadow-space with the RGB values of the shadows surrounding pixels, a “good” visual can be observed.

## Work Plan Schedule

Table: Task Schedule [note: this style is called Table Title]

| Task | Description |  | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Milestone 1 | Recreating source code | 2 Weeks | |  |  |  |  |  |  |  |
| Milestone 2 | Apply Optimizations to Process 1&2 |  |  | 2 Week | |  |  |  |  |  |
| Milestone 3 | Milestone 3: Apply Optimizations to Process 3&4 |  |  |  | 2 Weeks | |  |  |  |  |
| Milestone 4 | Milestone 4: Apply Optimizations to Process 5 |  |  |  |  |  | 1 Week |  |  |  |
| Milestone 5 | Debug and Profiler tools utilized |  |  |  |  |  |  | 1 Week |  |  |
| Milestone 6 | Analyze Results |  |  |  |  |  |  |  | <=1 Week |  |
|  | Write Final Report |  |  |  |  |  |  |  |  | 1 Week |

**References**

[1] E. Richter, R. Raettig, J. Mack, S. Valancius, B. Unal and A. Akoglu, "Accelerated Shadow Detection and Removal Method," 2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA), Abu Dhabi, United Arab Emirates, 2019, pp. 1-8, doi: 10.1109/AICCSA47632.2019.9035242. keywords: {Convolution;Kernel;Graphics processing units;Gray-scale;Mathematical model;Computer vision;Histograms},

[2] M. F. Lubis and A. Muis, "Fast shadow removal algorithm for river garbage pollution monitoring system," *2015 International Conference on Quality in Research (QiR)*, Lombok, Indonesia, 2015, pp. 175-179, doi: 10.1109/QiR.2015.7374922.

keywords: {Image segmentation;Rivers;Pollution;Discrete Fourier transforms;Monitoring;Parallel processing;Image color analysis;shadow removal;parallel processing;retinex PDE;NVIDIA;river pollution},

[3] Cook, Shane. (2013). *CUDA programming a developer’s guide to parallel computing with GPUs* (1st edition). Elsevier/MK.

[4] J. A. Stratton et al., "Optimization and architecture effects on GPU computing workload performance," 2012 Innovative Parallel Computing (InPar), San Jose, CA, USA, 2012, pp. 1-10, doi: 10.1109/InPar.2012.6339605. keywords: {Privatization;Abstracts;Random access memory;Bandwidth;Computer architecture;Compaction;GPU;CUDA;Optimization},