1. CUDA Filter Application Design

# GUI Frontend

## I guess your stuff goes here?

# Backend

## Image Acquisition

* + - 1. getImage(...): Helper function, takes a libtiff image object as input and returns a multidimensional array containing pixel values for each channel, with the following scheme:
* Container is 3-dimensional, with symbols x, y and z representing the indices of the array[[1]](#footnote-0)
  + x: denotes the channel, or “slice”, of a digital image. For example, an image with 3 color channels (e.g. RGB) would have possible x indices 0, 1 and 2
  + y: denotes the row selected in the current slice. Corresponds to the height of the image
  + z: denotes the column selected in the current slice. Corresponds to the width of the image
* For each slice of an image of height **n** and width **m**, the pixel with **y** = 0 and **z** = 0 will be located at the uppermost left corner of the image, and the pixel with **y** = **n** - 1 and **z** = **m** - 1 would be at the bottommost right of the image.
* Thus, to get the level for the uppermost left pixel in the red channel of an RGB image, you would input imageArray[0][0][0]

1. filterImage(...): Helper function, takes the array containing pixel values, as well as filter configuration parameters, and calls the relevant filter function. Returns a multidimensional array containing the filtered pixel values of the image
2. erFilter(...): Filter function for an ER filter, takes an image array, kernel size integer, and character denoting statistical method as input, returns a filtered image array

* Equivalent of host code in CUDA configuration, responsible for allocating the image array to memory on GPU device, declaring number of threads and grid blocks, and invoking the CUDA kernel
* After GPU device filters the image, modified data is allocated from device memory into a multidimensional array with identical structure to the array used to contain the original image data

1. (global) kernel<<<grid, threads>>>(...): Actual kernel code, executed on each thread. Takes the image array (allocated in GPU memory) as input, returns nothing

* Contains the logic for the filter proper, without sequential loops
* Calls helper functions to create a submatrix, determine the rank of each pixel in the matrix
* Directly modifies the pixel values from the input image array

1. (device) subMatrix(...): Helper function, takes as input image kernel size k and current pixel of interest, returns a submatrix of size k by k, with the current pixel of interest in the “middle” of the matrix
2. (device) rankMatrix(...): Helper function, takes as input a submatrix of size k by k, returns a matrix of size k by k containing the ranks of each pixel in the subarray

1. Note that the convention of using symbols x and y to denote the row and column of a pixel value is eschewed due to the use of multiple-channel images [↑](#footnote-ref-0)