CUDA Filter Application Design

# GUI Frontend

## User Input

## errorCheck(…): Helper function, shall be used to call and test various scenarios to when loading files or checking parameters

* + - 1. For loading files, the function shall check if the inputs are empty, or invalid.
         1. An error window shall appear if an error check fails.
      2. For checking parameters, the function shall check if the parameters are valid and not empty.
         1. An error window shall appear if an error check fails.
      3. For checking the weights, the function shall determine the network structure and possibly automatically input the parameters based on the network structure determined.
         1. An error window shall appear of an error check fails.
    1. **Loading files into the program**
       1. Loading via the load button
          1. A dialog box window shall appear with its own filter depending on the file to load to the program.

For flexibility, the user is able to select any file extension.

* + - * 1. File directory is loaded onto the designed textbox beside the load button.
      1. Loading via drag and drop
         1. A user shall drag a file to be loaded in to the program onto the textbox to be populated by the file directory.
         2. Three functions shall be used to load the file via drag and drop:

Textbox\_PreviewDragEnter(…): Used to determine whether the cursor is over a textbox that allows drag and drop.

Textbox\_PreviewDragOver(…): Used to override the default textbox behavior done by Windows.

Textbox\_PreviewDrop(…): Appends the filename onto the textbox.

* 1. **Gathering and saving pixel intensities from TIFF images**
     1. **This program shall use the Bitmiracle.LibTiff.Net library to accurately gather pixel intensity values into a multidimensional dimensional array.**
        1. An intensity value of 0 shall be considered as black, an intensity value of 255 shall be considered as white.
     2. **tiff2Array(…): Function used in order to gather intensity values from an image.**
     3. **writeToFile(…): Used in order to save the resulting intensity values back to an image once filtering algorithms have been executed.**

# Backend

## Image Acquisition

### 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-1)

##### 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]

### 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

### 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

### evFilter(…): Filter function for an EV filter, parameters similar to the erFilter(…) function for program flexibility. Takes an image array, kernel size integer, and character denoting statistical method as input, returns a filtered image array

### Function to call various transformation methods including mean, median, cut-off and multivalued methods.

### Methods are to allocate the array intensities to be used on the GPU kernel code. CPU bound code to be considered for machines without CUDA GPU parallelization capabilities removing the need for simulation/emulation.

* + 1. **knvFilter(…): Filter function for a KNV filter. Takes an image array, and through a series of loops returns a filtered image array.**
       1. Algorithm will replace discrepancies (high or low) within a grid of pixels based on the median value in the area evaluated.
       2. Processing will be done by the GPU using CUDA and the device will allocate the filtered data into an array with identical structure to the array that contained the original data.

### (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

### (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

### (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-1)