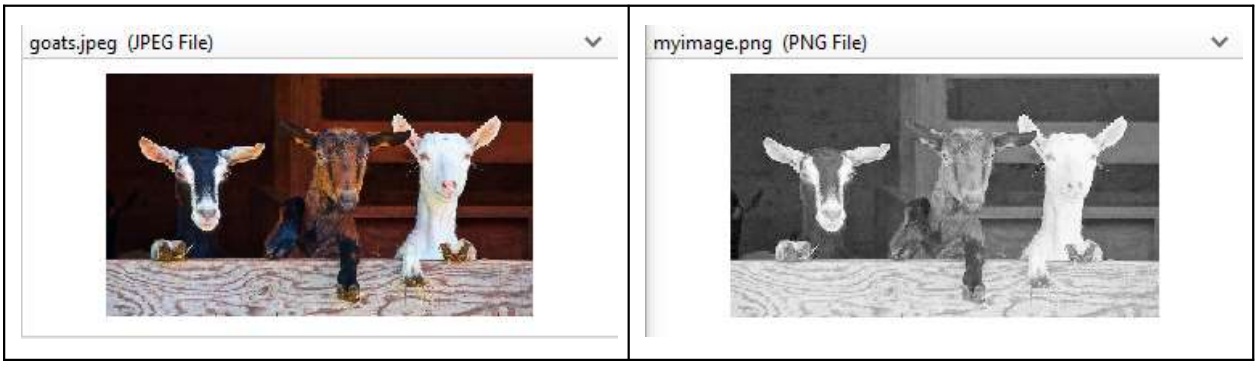
Nolan Anderson | Week 2 Assignment 1 | CPE 613 | Janruary 21 2023

- Implement the RGB to grayscale kernel and run it on a test JPEG image. Use MATLAB or Python functions to generate the RGB data from an image and view the before and after results. Upload a brief PDF report describing your process and showing your results.
- Use this image: goats.jpeg
- In Matlab, load it via rgb = imread('goats.jpeg') to get the RGB in a 456 x 810 x 3 uint8.
- Run imshow(rgb) to convince yourself this is an image of goats.
- Write this out to a file and load it into your program.
- Copy it up to the device.
- Convert to grayscale on the device.
- Copy the result down.
- Write the result to a file.
- Load the file into Matlab.
- Run imshow on the loaded data to see the grayscale image.

Before and after images (matlab)



Theory / Process

First, I used matlab (see the next page for code) to convert the given image to a text file. I then import the text file into the cuda program and create 4 matrix pointers, one for each rgb value and an empty gray matrix. These will be the destination matrices in the cudamemcpy call. To get the source for cudamemcpy, I create 4 additional empty matrices that are the size of the original matrix. I populate these matrices with the data in the text file. Next, I allocate space on the device using the first set of matrices with size matrixsize. Using these two sets, I copy the data to the device. Next, I create the dim3 thread and block count. There are 1024 threads per block. Afterwards I call the kernel. This uses the individual rgb value matrices to do the calculation instead of having one big matrix with all the values. I then copy the data back to the host and output to the text file. See the matlab code below on how I converted the text file back into an image.

Matlab Code

```
% Image to text
rgb = imread('goats.jpeg');
writematrix(rgb,'goats.txt', 'delimiter', ' ');
% Text to image
filename = 'grayscale.txt'
grayImage = uint8(importdata(filename));
imwrite(grayImage, 'myimage.png');
```

Cuda Code

```
/*
    Nolan Anderson
    CPE 613 Week 2 Assignment 2
    This program takes in an image that has been converted to a text file.
    This text file has the RGB values. I converted the file in matlab using the following code:
    rgb = imread('goats.jpeg');
    writematrix(rgb,'goats.txt', 'delimiter', ' ');
    ^^^^^ MAKE SURE TO PUT THE DEMINSIONS ON THE FIRST ROW OF THE TEXT FILE ^^^^^
    ----------------------------------------------------
    After this program, you should be able to run the following in matlab and see the grayscale
image:
    filename = 'graygoats.txt'
    grayImage = uint8(importdata(filename));
    imwrite(grayImage, 'myimage.png');
    ----------------------------------------------------
    To compile, first 'load module cuda'
    Then: 'nvcc graygoats.cu -o goats'

    Next, 'run_gpu goats' and use all default values.
    This should result in a grayscale data of your original image.
*/

#include <fstream>
#include <iostream>
using namespace std;

// RGB to gray kernel. Pretty much just what was in the slides.
__global__ void rgb2gray_kernel(unsigned char* r, unsigned char* g, unsigned char* b, unsigned char*
gray, int matrixHeight, int matrixWidth){
    unsigned int row = blockIdx.y*blockDim.y + threadIdx.y;     // Row and column values
    unsigned int col = blockIdx.x*blockDim.x + threadIdx.x;
    unsigned int index = row*matrixWidth + col;
    if(row < matrixHeight && col < matrixWidth)                // Make sure we don't go out of bounds
on the row & column
        gray[index] = r[index] *3/10 + g[index]*6/10 + b[index]*1/10; // General calculation for RGB
-> gray values.
}

int main(){
    // Allocate matrices, first using matrixWidth and matrixHeight.
    // The data is coming from goats.txt, or any image whose data is in
    // text format.
    ifstream inFile;                        // input file
    int matrixWidth, matrixHeight;          // matrixWidth and matrixHeight of the matrix.
```

```
    inFile.open("goats.txt");            // open the input file
    inFile >> matrixWidth >> matrixHeight;  // pull the matrix deminsions out of the file. (on the
first line)
    const int matrixSize = matrixWidth * matrixHeight; // Obtain the matrix size.
    unsigned char  *r, *g, *b, *gray;       // Create the pointers to hold the rgb data and the
grayscale data.

    // Allocate a matrix for each color.
    unsigned char grayMatrix[matrixSize], rMatrix[matrixSize], gMatrix[matrixSize],
bMatrix[matrixSize];

    // Populate each matrix with the input file values
    int temp;
    for(int r = 0; r < matrixSize; r++){    // Loop through number of values in matrix
        inFile >> temp;
        rMatrix[r] = temp;                // Assign to each matrix.
        inFile >> temp;
        gMatrix[r] = temp;
        inFile >> temp;
        bMatrix[r] = temp;
    }

    // Allocate space on the device for rgb and gray matrices.
    // The size is the size of an unsigned char, multiplied by the size of the matrix.
    cudaMalloc((void**)&r, matrixSize*sizeof(unsigned char));
    cudaMalloc((void**)&g, matrixSize*sizeof(unsigned char));
    cudaMalloc((void**)&b, matrixSize*sizeof(unsigned char));
    cudaMalloc((void**)&gray, matrixSize*sizeof(unsigned char));

    // Copy the rgb values to the device. Make sure to leave out the gray value. This one will only
be
    // copied back to the host.
    cudaMemcpy(r, &rMatrix, matrixSize * sizeof(unsigned char), cudaMemcpyHostToDevice);
    cudaMemcpy(g, &gMatrix, matrixSize * sizeof(unsigned char), cudaMemcpyHostToDevice);
    cudaMemcpy(b, &bMatrix, matrixSize * sizeof(unsigned char), cudaMemcpyHostToDevice);

    // Create the dim3. 1024x1024 threads per block. Just using the standard number of blocks from
    // the slides.
    dim3 numThreadsPerBlock(1024,1024);
    dim3 numBlocks((matrixWidth + numThreadsPerBlock.x - 1)/numThreadsPerBlock.x,
                   (matrixHeight + numThreadsPerBlock.y - 1)/numThreadsPerBlock.y);

    // Call the kernel, passing the appropriate values in. We need to add the dim3 in the <<< >>> so
the
    // warps / kernel / SMs know how to allocate their data. Gray is the output and
matrixheight/width are
    // there to support the calculations and indexing.
    rgb2gray_kernel <<< numThreadsPerBlock, numBlocks >>> (r, g, b, gray, matrixHeight, matrixWidth);

    // Copy the data back to host from the device.
    cudaMemcpy(grayMatrix, gray, matrixSize*sizeof(unsigned char), cudaMemcpyDeviceToHost);

    // Write the data to an outfile. In this case we're going back to a text file.
    ofstream outFile;               // osftream variable
    outFile.open("graygoats.txt");  // open the file
```

```
    for(int row = 0; row < matrixHeight; row++){        // Loop through the rows
        for(int col = 0; col < matrixWidth; col++)      // Loop through the columns
            outFile << +grayMatrix[row*matrixWidth + col] << " ";   // go to index to get current
column in row.
        outFile << endl;    // Move to a new line (we're done with the previous row)
    }

    // Close the files and exit program.
    inFile.close();
    outFile.close();
    return 0;
}
```