



GPU Teaching Kit
Accelerated Computing



Lecture 3.3 – CUDA Parallelism Model

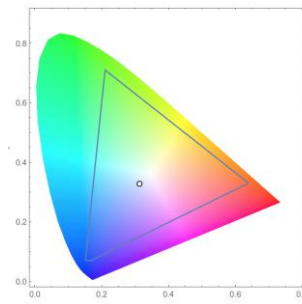
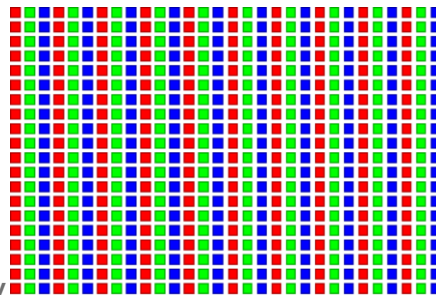
Color-to-Grayscale Image Processing Example

Objective

- To gain deeper understanding of multi-dimensional grid kernel configurations through a real-world use case

RGB Color Image Representation

- Each pixel in an image is an RGB value
- The format of an image's row is (r g b) (r g b) ... (r g b)
- RGB ranges are not distributed uniformly
- Many different color spaces, here we show the constants to convert to AdobeRGB color space
 - The vertical axis (y value) and horizontal axis (x value) show the fraction of the pixel intensity that should be allocated to G and B. The remaining fraction ($1-y-x$) of the pixel intensity that should be assigned to R
 - The triangle contains all the representable colors in this color space



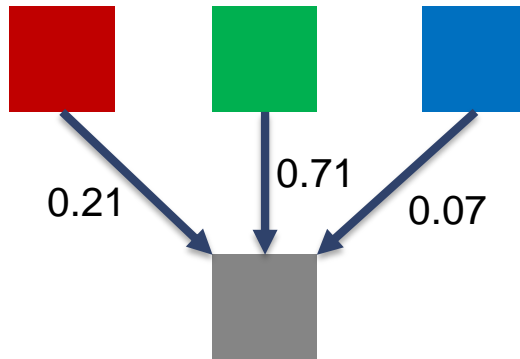
RGB to Grayscale Conversion



A grayscale digital image is an image in which the value of each pixel carries only intensity information.

Color Calculating Formula

- For each pixel (r g b) at (I, J) do:
 $\text{grayPixel}[I,J] = 0.21*r + 0.71*g + 0.07*b$
- This is just a dot product $\langle [r,g,b], [0.21,0.71,0.07] \rangle$ with the constants being specific to input RGB space



RGB to Grayscale Conversion Code

```
#define CHANNELS 3 // we have 3 channels corresponding to RGB
// The input image is encoded as unsigned characters [0, 255]
__global__ void colorConvert(unsigned char * grayImage,
                             unsigned char * rgbImage,
                             int width, int height) {
    int x = threadIdx.x + blockIdx.x * blockDim.x;
    int y = threadIdx.y + blockIdx.y * blockDim.y;

    if (x < width && y < height) {

    }
}
```

RGB to Grayscale Conversion Code

```
#define CHANNELS 3 // we have 3 channels corresponding to RGB
// The input image is encoded as unsigned characters [0, 255]
__global__ void colorConvert(unsigned char * grayImage,
                             unsigned char * rgbImage,
                             int width, int height) {
    int x = threadIdx.x + blockIdx.x * blockDim.x;
    int y = threadIdx.y + blockIdx.y * blockDim.y;

    if (x < width && y < height) {
        // get 1D coordinate for the grayscale image
        int grayOffset = y*width + x;
        // one can think of the RGB image having
        // CHANNEL times columns than the gray scale image
        int rgbOffset = grayOffset*CHANNELS;
        unsigned char r = rgbImage[rgbOffset]; // red value for pixel
        unsigned char g = rgbImage[rgbOffset + 1]; // green value for pixel
        unsigned char b = rgbImage[rgbOffset + 2]; // blue value for pixel
    }
}
```

RGB to Grayscale Conversion Code

```
#define CHANNELS 3 // we have 3 channels corresponding to RGB
// The input image is encoded as unsigned characters [0, 255]
__global__ void colorConvert(unsigned char * grayImage,
                             unsigned char * rgbImage,
                             int width, int height) {
    int x = threadIdx.x + blockIdx.x * blockDim.x;
    int y = threadIdx.y + blockIdx.y * blockDim.y;

    if (x < width && y < height) {
        // get 1D coordinate for the grayscale image
        int grayOffset = y*width + x;
        // one can think of the RGB image having
        // CHANNEL times columns than the gray scale image
        int rgbOffset = grayOffset*CHANNELS;
        unsigned char r = rgbImage[rgbOffset]; // red value for pixel
        unsigned char g = rgbImage[rgbOffset + 2]; // green value for pixel
        unsigned char b = rgbImage[rgbOffset + 3]; // blue value for pixel
        // perform the rescaling and store it
        // We multiply by floating point constants
        grayImage[grayOffset] = 0.21f*r + 0.71f*g + 0.07f*b;
    }
}
```




GPU Teaching Kit

Accelerated Computing



The GPU Teaching Kit is licensed by NVIDIA and the University of Illinois under the [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/).