GPU Teaching Kit

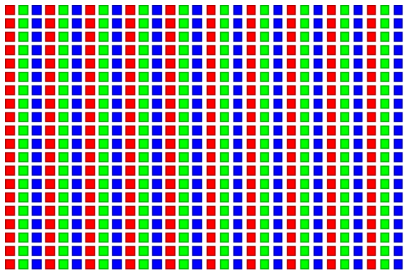
Accelerated Computing

Lecture 13 – CUDA Parallelism Model

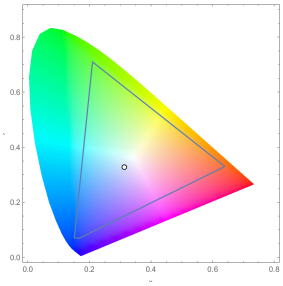
Examples

Example 1: Color-to-Grayscale Image Processing

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 AdbobeRGB 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

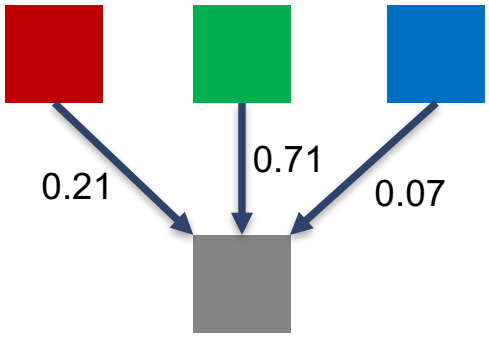


RGB to Grayscale Conversion

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:
  grayPixel[I,J] = 0.21*r + 0.71*g + 0.07*b
- This is just a dot product <[r,g,b],[0.21,0.71,0.07]> 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

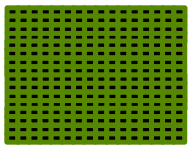
```
#define CHANNELS 3 // we have 3 channels corresponding to RGB
// The input image is encoded as unsigned characters [0, 255]
__global__ void colorConvert(unsigned char * grayImage,
                             unsigned char * rgbImage,
                             int width, int height) {
 int x =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

Example 2: Image Blur

Image Blurring

Blurring Box



Pixels processed by a thread block

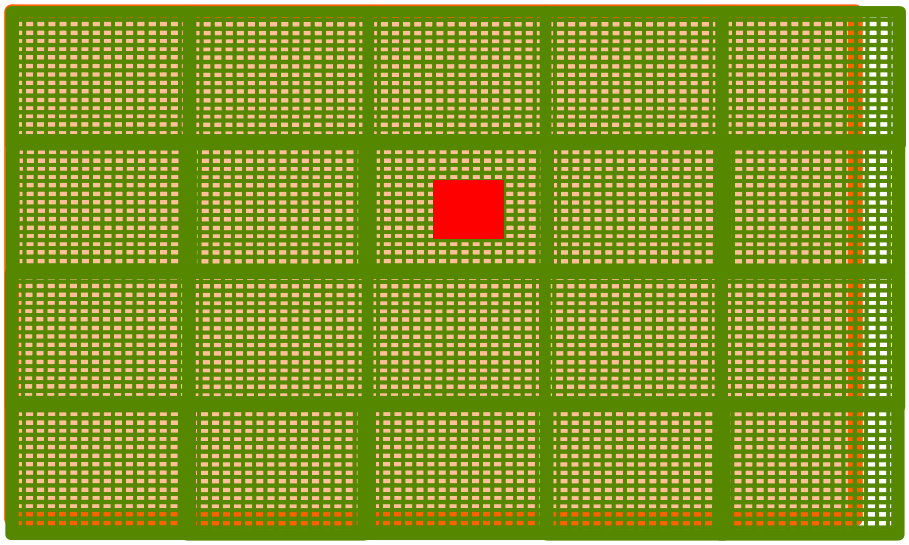


Image Blur as a 2D Kernel

```
 __global__
  void blurKernel(unsigned char * in, unsigned char * out,
int w, int h)
  {
    int Col  = blockIdx.x * blockDim.x + threadIdx.x;
    int Row  = blockIdx.y * blockDim.y + threadIdx.y;

    if (Col < w && Row < h) {
        ... // Rest of our kernel
    }
  }
```

```cuda
__global__
 void blurKernel(unsigned char * in, unsigned char * out, int w, int h) {
     int Col  = blockIdx.x * blockDim.x + threadIdx.x;
     int Row  = blockIdx.y * blockDim.y + threadIdx.y;

     if (Col < w && Row < h) {
         int pixVal = 0;
         int pixels = 0;

         // Get the average of the surrounding 2xBLUR_SIZE x 2xBLUR_SIZE box
         for(int blurRow = -BLUR_SIZE; blurRow < BLUR_SIZE+1; ++blurRow) {
             for(int blurCol = -BLUR_SIZE; blurCol < BLUR_SIZE+1; ++blurCol) {

                 int curRow = Row + blurRow;
                 int curCol = Col + blurCol;
                 // Verify we have a valid image pixel
                 if(curRow > -1 && curRow < h && curCol > -1 && curCol < w) {
                     pixVal += in[curRow * w + curCol];
                     pixels++; // Keep track of number of pixels in the accumulated total
                 }
             }
         }

         // Write our new pixel value out
         out[Row * w + Col] = (unsigned char)(pixVal / pixels);
     }
 }
```