Histogram Grade

Grade Summary (History) (/grade/history/59811)

Created:	less than a minute ago (2022-03-21 03:29:21 +0000 UTC)
Total Score:	100 out of 100 points
Coding Score:	100 out of 100 points
Questions Score:	0

Program Code

```
// Histogram Equalization

#include <wb.h>

#define wbCheck(stmt)
  do {
    cudaError_t err = stmt;
    if (err != cudaSuccess) {
      wbLog(ERROR, "Failed to run stmt ", #stmt);
      wbLog(ERROR, "Got CUDA error ...  ", cudaGetErrorString(err));
      return -1;
    }
  } while (0)


#define HISTOGRAM_LENGTH 256
#define chan_count 3
//implementation-independent code piece for unsigned char
typedef unsigned char uint8_t;
typedef unsigned int  uint_t;
#define TILE_WIDTH 32
```

```
#define RGB_MAX 255.0


/*
int next = (numAColumns + BLOCK_SIZE - 1) / BLOCK_SIZE;
float hold = 0;
*/
//For the CUDA Kernels
//Here I cast the image to an unsigned char:
//  PARTS SIMILAR TO SECTIONS FROM MP3

__global__ void float_to_uint8_t(float *input, uint8_t *output, int width
  int x = (blockIdx.x * blockDim.x) + threadIdx.x;
  int y = (blockIdx.y * blockDim.y) + threadIdx.y;

  if (x < width && y < height){
    int idx = blockIdx.z * (width * height) + y * (width) + x;
    output[idx] = (uint8_t) ((HISTOGRAM_LENGTH - 1) * input[idx]);
  }
}

//convert an input image from RGB color scale to grayscale
__global__ void color_to_dark(uint8_t *input, uint8_t *output, int width,
  int x = (blockIdx.x * blockDim.x) + threadIdx.x;
  int y = (blockIdx.y * blockDim.y) + threadIdx.y;

  if (x < width && y < height){
    int idx = y * (width) + x;
    uint8_t R = input[3 * idx + 0];
    uint8_t G = input[3 * idx + 1];
    uint8_t B = input[3 * idx + 2];
    output[idx] = (uint8_t) (0.07*B + 0.71*G + 0.21*R);
  }
}

//Get a histogram of the image
__global__ void dark_to_graph(uint8_t *input, uint_t *output, int width,
  __shared__ uint_t histogram[HISTOGRAM_LENGTH];

  int index_threads = threadIdx.x + threadIdx.y * blockDim.x;
  if (index_threads < HISTOGRAM_LENGTH) {
    histogram[index_threads] = 0;
  }

  __syncthreads();
  int x = blockIdx.x * blockDim.x + threadIdx.x;
  int y = blockIdx.y * blockDim.y + threadIdx.y;
  if (x < width && y < height) {
    int idx = y * (width) + x;
    uint8_t val = input[idx];
    //utilize atomic add function
    //B.14. Atomic Functions
    atomicAdd(&(histogram[val]), 1);
  }

  __syncthreads();
  if (index_threads < HISTOGRAM_LENGTH) {
    atomicAdd(&(output[index_threads]), histogram[index_threads]);
  }
}

//Compute the scan and prefix sum of the histogram to arrive at the histo
//We get a scan of histogram -> histogram equalization function
// Cumulative Distribution Function @ https://www.cs.umd.edu/class/fall20
// >> Brent-Kung derivived parallel inclusive scan algorithm
// >> http://www.sci.utah.edu/~acoste/uou/Image/project1/Arthur_COSTE_Pro
__global__ void scan_to_stat(uint_t *input, float *output, int width, int
  __shared__ uint_t cmlt_dist_func[HISTOGRAM_LENGTH];
  int x = threadIdx.x;
  cmlt_dist_func[x] = input[x];

  //Scan pt-1
  for (unsigned int scanner = 1; scanner <= HISTOGRAM_LENGTH / 2; scanner
    __syncthreads();
    int idx = (x + 1) * 2 * scanner - 1;
    if (idx < HISTOGRAM_LENGTH) {
      cmlt_dist_func[idx] += cmlt_dist_func[idx - scanner];
    }
  }
  //Scan pt-2
  for (int scanner = HISTOGRAM_LENGTH / 4; scanner > 0; scanner /= 2) {
    __syncthreads();
    int idx = (x + 1) * 2 * scanner - 1;
    if (idx + scanner < HISTOGRAM_LENGTH) {
      cmlt_dist_func[idx + scanner] += cmlt_dist_func[idx];
    }
  }
  __syncthreads();
  output[x] = cmlt_dist_func[x] / ((float) (width * height));
}
```

```


//Apply the histogram equalization function
//get color corrected image from input image
__global__ void equal_func(uint8_t *shift, float *cmlt_dist_func, int wid
  int x = blockIdx.x * blockDim.x + threadIdx.x;
  int y = blockIdx.y * blockDim.y + threadIdx.y;

  if (x < width && y < height) {
    int idx = blockIdx.z * (width * height) + y * (width) + x;
    uint8_t val = shift[idx];

    float equalized = 255 * (cmlt_dist_func[val] - cmlt_dist_func[0]) / (
    float clamped   = min(max(equalized, 0.0), 255.0);

    shift[idx] = (uint8_t) (clamped);
  }
}

//Cast back to float
__global__ void uint8_t_float(uint8_t *input, float *output, int width, i

  int x = blockIdx.x * blockDim.x + threadIdx.x;
  int y = blockIdx.y * blockDim.y + threadIdx.y;

  if (x < width && y < height) {
    int idx = blockIdx.z * (width * height) + y * (width) + x;
    output[idx] = (float) (input[idx] / 255.0);
  }
}

//@@ insert code here

int main(int argc, char **argv) {
  wbArg_t args;
  int imageWidth;
  int imageHeight;
  int imageChannels;
  wbImage_t inputImage;
  wbImage_t outputImage;
  float *hostInputImageData;
  float *hostOutputImageData;
  const char *inputImageFile;

  //@@ Insert more code here
```

```
float   *deviceImageFloat;
float   *deviceImagecmlt_dist_func;
uint_t  *deviceImageHistogram;
uint8_t *deviceImageUChar;
uint8_t *deviceImageUCharGrayScale;

args = wbArg_read(argc, argv); /* parse the input arguments */

inputImageFile = wbArg_getInputFile(args, 0);

wbTime_start(Generic, "Importing data and creating memory on host");
inputImage = wbImport(inputImageFile);
imageWidth = wbImage_getWidth(inputImage);
imageHeight = wbImage_getHeight(inputImage);
imageChannels = wbImage_getChannels(inputImage);

hostInputImageData  = wbImage_getData(inputImage);

outputImage = wbImage_new(imageWidth, imageHeight, imageChannels);

hostOutputImageData = wbImage_getData(outputImage);

wbTime_stop(Generic, "Importing data and creating memory on host");

//print width, height, and channel of image
printf("%d, %d, %d\n", imageWidth, imageHeight, imageChannels);

/*
Cuda Toolkilt Documentation - Programming Guide @ B.29
Assertion stops the kernel execution if expression is equal to zero.
Triggers a breakpoint withing a debugger
and the debugger can also be stopped to inspect the device's current st
*/
assert(imageChannels == chan_count);

//@@ @@//
//@@ Here I allocate GPU memory @@//
int imageArea = imageWidth * imageHeight;
int imageVol = imageWidth * imageHeight * imageChannels;
cudaMalloc((void**) &deviceImageFloat, imageVol * sizeof(float));
  //image grayscale
cudaMalloc((void**) &deviceImageUChar, imageVol * sizeof(uint8_t));
cudaMalloc((void**) &deviceImageUCharGrayScale, imageArea * sizeof(uint
  //the actual histogram
cudaMalloc((void**) &deviceImageHistogram, HISTOGRAM_LENGTH * sizeof(ui
cudaMemset((void**) &deviceImageHistogram, 0, HISTOGRAM_LENGTH * sizeof
  //the Cumulative Distribution Function
cudaMalloc((void**) &deviceImagecmlt_dist_func, HISTOGRAM_LENGTH * size

//@@ Here I copy memory to the GPU @@//
//it is the memory input into the GPU
cudaMemcpy(deviceImageFloat, hostInputImageData, imageVol * sizeof(floa
//@@ Initialize the grid and block dimensions here:
dim3 dimensionBlock;
dim3 dimensionGrid;

//for uint8_t
dimensionBlock = dim3(TILE_WIDTH, TILE_WIDTH, 1);
dimensionGrid = dim3(ceil(imageWidth/32.0), ceil(imageHeight/32.0), ima
//perform float to uint8_t:
float_to_uint8_t<<<dimensionGrid, dimensionBlock>>>(deviceImageFloat, d
cudaDeviceSynchronize();

//convert to grayscale
dimensionBlock = dim3(TILE_WIDTH, TILE_WIDTH, 1);
dimensionGrid = dim3(ceil(imageWidth/32.0), ceil(imageHeight/32.0), 1);

color_to_dark<<<dimensionGrid, dimensionBlock>>>(deviceImageUChar, devi
cudaDeviceSynchronize();

//convert to histogram
dimensionBlock = dim3(32, 32, 1);
dimensionGrid  = dim3(ceil(imageWidth/32.0), ceil(imageHeight/32.0), 1)

dark_to_graph<<<dimensionGrid, dimensionBlock>>>(deviceImageUCharGraySc
cudaDeviceSynchronize();

//convert to cdf
dimensionBlock = dim3(HISTOGRAM_LENGTH, 1, 1);
dimensionGrid  = dim3(1, 1, 1);

scan_to_stat<<<dimensionGrid, dimensionBlock>>>(deviceImageHistogram, d
cudaDeviceSynchronize();

//equalization function
dimensionBlock = dim3(32, 32, 1);
dimensionGrid  = dim3(ceil(imageWidth/32.0), ceil(imageHeight/32.0), im

equal_func<<<dimensionGrid, dimensionBlock>>>(deviceImageUChar, deviceI
cudaDeviceSynchronize();
```

```

  //convert to uint8
  dimensionBlock = dim3(32, 32, 1);
  dimensionGrid  = dim3(ceil(imageWidth/32.0), ceil(imageHeight/32.0), im

  uint8_t_float<<<dimensionGrid, dimensionBlock>>>(deviceImageUChar, devi
  cudaDeviceSynchronize();

  //@@ insert code here
  //CPU Operations follow

  //@@ Here I copy the output memory to the CPU
  cudaMemcpy(hostOutputImageData, deviceImageFloat, imageWidth * imageHei

  //@@ Here I check the output image solution and free GPU memory
  wbSolution(args, outputImage);
  cudaFree(deviceImageFloat);
  cudaFree(deviceImageUChar);
  cudaFree(deviceImageUCharGrayScale);
  cudaFree(deviceImageHistogram);
  cudaFree(deviceImagecmlt_dist_func);
  // Free CPU Memory
  free(hostInputImageData);
  free(hostOutputImageData);


  wbTime_stop(GPU, "Freeing GPU Memory");


  //@@ insert code here
  //DONE

  return 0;
}

```

Designed and architected by Abdul Dakkak (https://www.dakkak.dev/).