RGB Image Manipulation Using CUDA: Practice

To process an image in RGB, we just need to apply what we did for a grayscale image but three times, each for an RGB channel: have a matrix (vector) of data for channel R, G and B.

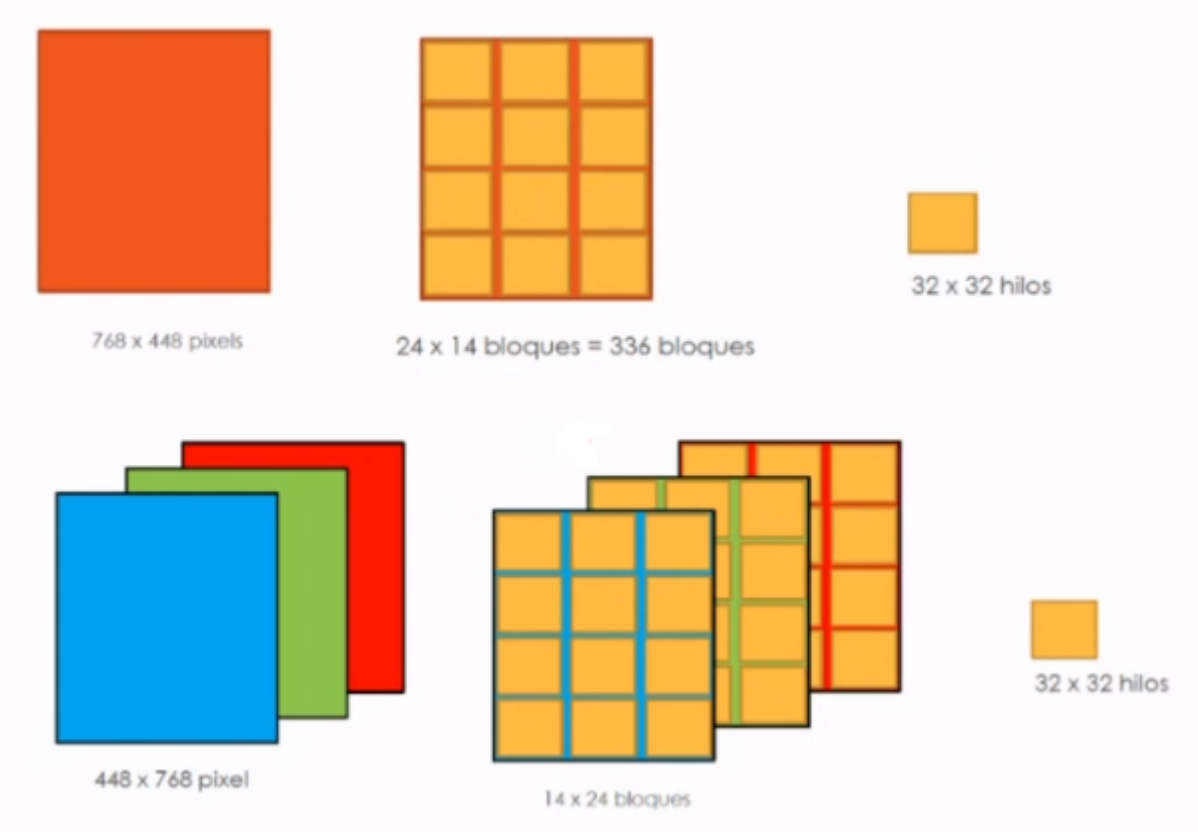

Figure 1: image

Lab 11

Write a program in c/c++ using CUDA in which you implement a kernel to modify the contrast of an RGB image, and another to modify the brightness, considering the following requirements:

- Blocks of 32 x 32 threads
- The kernel for complement, contrast and brightness should be:
  - `__global__ void complement(uchar* R, uchar* G, uchar* B)`
  - `__global__ void contrast(uchar* R, uchar* G, uchar* B, float fc)`
  - `__global__ void brightness(uchar* R, uchar* G, uchar* B, float fb)`
- Include error management with a function:

- `__host__ void checkCUDAError(const char* msg)`

Input

Figure 2: img

Solution

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

#include <stdio.h>
#include <stdlib.h>
#include <opencv2/opencv.hpp>

__host__ void checkCUDAError(const char* msg) {
    cudaError_t error;
    cudaDeviceSynchronize();
    error = cudaGetLastError();
    if (error != cudaSuccess) {
        printf("ERROR %d: %s (%s)\n", error,
           cudaGetErrorString(error), msg);
    }
}

__global__ void complement(uchar* R, uchar* G, uchar* B) {
    // locate my current block row
    int threads_per_block = blockDim.x * blockDim.y;
```

```
20      int threads_per_row = threads_per_block * gridDim.x;
21      int row_offset = threads_per_row * blockIdx.y;
22
23      // locate my current block column
24      int block_offset = blockIdx.x * threads_per_block;
25      int threadId_inside = blockDim.x * threadIdx.y + threadIdx
            .x;
26
27      int gId = row_offset + block_offset + threadId_inside;
28      R[gId] = 255 - R[gId];
29      G[gId] = 255 - G[gId];
30      B[gId] = 255 - B[gId];
31  }
32
33  __global__ void contrast(uchar* R, uchar* G, uchar* B, float
        fc) {
34      // locate my current block row
35      int threads_per_block = blockDim.x * blockDim.y;
36      int threads_per_row = threads_per_block * gridDim.x;
37      int row_offset = threads_per_row * blockIdx.y;
38
39      // locate my current block column
40      int block_offset = blockIdx.x * threads_per_block;
41      int threadId_inside = blockDim.x * threadIdx.y + threadIdx
            .x;
42
43      int gId = row_offset + block_offset + threadId_inside;
44      if (fc * R[gId] > 255) { R[gId] = 255; } else { R[gId] =
            fc * R[gId]; }
45      if (fc * G[gId] > 255) { G[gId] = 255; } else { G[gId] =
            fc * G[gId]; }
46      if (fc * B[gId] > 255) { B[gId] = 255; } else { B[gId] =
            fc * B[gId]; }
47  }
48
49  __global__ void brightness(uchar* R, uchar* G, uchar* B, float
        fb) {
50      // locate my current block row
51      int threads_per_block = blockDim.x * blockDim.y;
52      int threads_per_row = threads_per_block * gridDim.x;
53      int row_offset = threads_per_row * blockIdx.y;
54
55      // locate my current block column
56      int block_offset = blockIdx.x * threads_per_block;
57      int threadId_inside = blockDim.x * threadIdx.y + threadIdx
            .x;
58
59      int gId = row_offset + block_offset + threadId_inside;
60      if (fb >= 0){
61      if (fb + R[gId] > 255) { R[gId] = 255; } else { R[gId] =
            fb + R[gId]; }
```

```
62      if (fb + G[gId] > 255) { G[gId] = 255; } else { G[gId] =
            fb + G[gId]; }
63      if (fb + B[gId] > 255) { B[gId] = 255; } else { B[gId] =
            fb + B[gId]; }
64      }
65
66      if (fb < 0) {
67      if (fb + R[gId] < 0) { R[gId] = 0; } else { R[gId] = fb +
            R[gId]; }
68      if (fb + G[gId] < 0) { G[gId] = 0; } else { G[gId] = fb +
            G[gId]; }
69      if (fb + B[gId] < 0) { B[gId] = 0; } else { B[gId] = fb +
            B[gId]; }
70      }
71  }
72
73  using namespace cv;
74  int main() {
75
76      Mat img = imread("antenaRGB.jpg");
77
78      const int R = img.rows;
79      const int C = img.cols;
80
81      Mat imgComp(img.rows, img.cols, img.type());
82      Mat imgCont(img.rows, img.cols, img.type());
83      Mat imgBright(img.rows, img.cols, img.type());
84      uchar* host_r, * host_g, * host_b, * dev_r1, * dev_g1, *
            dev_b1, * dev_r2, * dev_g2, * dev_b2, * dev_r3, *
            dev_g3, * dev_b3;
85      host_r = (uchar*)malloc(sizeof(uchar) * R * C);
86      host_g = (uchar*)malloc(sizeof(uchar) * R * C);
87      host_b = (uchar*)malloc(sizeof(uchar) * R * C);
88
89      cudaMalloc((void**)&dev_r1, sizeof(uchar) * R * C);
90      checkCUDAError("Error at malloc dev_r1");
91      cudaMalloc((void**)&dev_g1, sizeof(uchar) * R * C);
92      checkCUDAError("Error at malloc dev_g1");
93      cudaMalloc((void**)&dev_b1, sizeof(uchar) * R * C);
94      checkCUDAError("Error at malloc dev_b1");
95
96      cudaMalloc((void**)&dev_r2, sizeof(uchar) * R * C);
97      checkCUDAError("Error at malloc dev_r2");
98      cudaMalloc((void**)&dev_g2, sizeof(uchar) * R * C);
99      checkCUDAError("Error at malloc dev_g2");
100     cudaMalloc((void**)&dev_b2, sizeof(uchar) * R * C);
101     checkCUDAError("Error at malloc dev_b2");
102
103     cudaMalloc((void**)&dev_r3, sizeof(uchar) * R * C);
104     checkCUDAError("Error at malloc dev_r3");
105     cudaMalloc((void**)&dev_g3, sizeof(uchar) * R * C);
```

```
106     checkCUDAError("Error at malloc dev_g3");
107     cudaMalloc((void**)&dev_b3, sizeof(uchar) * R * C);
108     checkCUDAError("Error at malloc dev_b3");
109
110     // matrix as vector
111     for (int i = 0; i < R; i++) {
112         for (int j = 0; j < C; j++) {
113             Vec3b pix = img.at<Vec3b>(i, j);
114             host_r[i * C + j] = pix[2];
115             host_g[i * C + j] = pix[1];
116             host_b[i * C + j] = pix[0];
117         }
118     }
119     cudaMemcpy(dev_r1, host_r, sizeof(uchar) * R * C,
        cudaMemcpyHostToDevice);
120     checkCUDAError("Error at memcpy host_r -> dev_r1");
121     cudaMemcpy(dev_g1, host_g, sizeof(uchar) * R * C,
        cudaMemcpyHostToDevice);
122     checkCUDAError("Error at memcpy host_r -> dev_g1");
123     cudaMemcpy(dev_b1, host_b, sizeof(uchar) * R * C,
        cudaMemcpyHostToDevice);
124     checkCUDAError("Error at memcpy host_r -> dev_b1");
125
126     cudaMemcpy(dev_r2, host_r, sizeof(uchar) * R * C,
        cudaMemcpyHostToDevice);
127     checkCUDAError("Error at memcpy host_r -> dev_r2");
128     cudaMemcpy(dev_g2, host_g, sizeof(uchar) * R * C,
        cudaMemcpyHostToDevice);
129     checkCUDAError("Error at memcpy host_r -> dev_g2");
130     cudaMemcpy(dev_b2, host_b, sizeof(uchar) * R * C,
        cudaMemcpyHostToDevice);
131     checkCUDAError("Error at memcpy host_r -> dev_b2");
132
133     cudaMemcpy(dev_r3, host_r, sizeof(uchar) * R * C,
        cudaMemcpyHostToDevice);
134     checkCUDAError("Error at memcpy host_r -> dev_r3");
135     cudaMemcpy(dev_g3, host_g, sizeof(uchar) * R * C,
        cudaMemcpyHostToDevice);
136     checkCUDAError("Error at memcpy host_r -> dev_g3");
137     cudaMemcpy(dev_b3, host_b, sizeof(uchar) * R * C,
        cudaMemcpyHostToDevice);
138     checkCUDAError("Error at memcpy host_r -> dev_b3");
139
140     dim3 block(32, 32);
141     dim3 grid(C / 32, R / 32);
142
143     complement << < grid, block >> > (dev_r1, dev_g1, dev_b1);
144     cudaDeviceSynchronize();
145     checkCUDAError("Error at kernel complement");
146
```

```
147     cudaMemcpy(host_r, dev_r1, sizeof(uchar) * R * C,
            cudaMemcpyDeviceToHost);
148     checkCUDAError("Error at memcpy host_r <- dev_r1");
149     cudaMemcpy(host_g, dev_g1, sizeof(uchar) * R * C,
            cudaMemcpyDeviceToHost);
150     checkCUDAError("Error at memcpy host_r <- dev_g1");
151     cudaMemcpy(host_b, dev_b1, sizeof(uchar) * R * C,
            cudaMemcpyDeviceToHost);
152     checkCUDAError("Error at memcpy host_r <- dev_b1");
153
154     for (int i = 0; i < R; i++) {
155         for (int j = 0; j < C; j++) {
156             imgComp.at<Vec3b>(i, j)[0] = host_b[i * C + j];
157             imgComp.at<Vec3b>(i, j)[1] = host_g[i * C + j];
158             imgComp.at<Vec3b>(i, j)[2] = host_r[i * C + j];
159         }
160     }
161
162     contrast << < grid, block >> > (dev_r2, dev_g2, dev_b2,
            0.5);
163     cudaDeviceSynchronize();
164     checkCUDAError("Error at kernel contrast");
165
166     cudaMemcpy(host_r, dev_r2, sizeof(uchar) * R * C,
            cudaMemcpyDeviceToHost);
167     checkCUDAError("Error at memcpy host_r <- dev_r2");
168     cudaMemcpy(host_g, dev_g2, sizeof(uchar) * R * C,
            cudaMemcpyDeviceToHost);
169     checkCUDAError("Error at memcpy host_r <- dev_g2");
170     cudaMemcpy(host_b, dev_b2, sizeof(uchar) * R * C,
            cudaMemcpyDeviceToHost);
171     checkCUDAError("Error at memcpy host_r <- dev_b2");
172
173     for (int i = 0; i < R; i++) {
174         for (int j = 0; j < C; j++) {
175             imgCont.at<Vec3b>(i, j)[0] = host_b[i * C + j];
176             imgCont.at<Vec3b>(i, j)[1] = host_g[i * C + j];
177             imgCont.at<Vec3b>(i, j)[2] = host_r[i * C + j];
178         }
179     }
180
181     brightness << < grid, block >> > (dev_r3, dev_g3, dev_b3,
            100);
182     cudaDeviceSynchronize();
183     checkCUDAError("Error at kernel brightness");
184
185     cudaMemcpy(host_r, dev_r3, sizeof(uchar) * R * C,
            cudaMemcpyDeviceToHost);
186     checkCUDAError("Error at memcpy host_r <- dev_r3");
187     cudaMemcpy(host_g, dev_g3, sizeof(uchar) * R * C,
            cudaMemcpyDeviceToHost);
```

```
188     checkCUDAError("Error at memcpy host_r <- dev_g3");
189     cudaMemcpy(host_b, dev_b3, sizeof(uchar) * R * C,
            cudaMemcpyDeviceToHost);
190     checkCUDAError("Error at memcpy host_r <- dev_b3");
191
192     for (int i = 0; i < R; i++) {
193         for (int j = 0; j < C; j++) {
194             imgBright.at<Vec3b>(i, j)[0] = host_b[i * C + j];
195             imgBright.at<Vec3b>(i, j)[1] = host_g[i * C + j];
196             imgBright.at<Vec3b>(i, j)[2] = host_r[i * C + j];
197         }
198     }
199
200     imshow("Image", img);
201     imshow("Image Complement", imgComp);
202     imshow("Image Contrast", imgCont);
203     imshow("Image Brightness", imgBright);
204     waitKey(0);
205
206     free(host_r);
207     free(host_g);
208     free(host_b);
209     cudaFree(dev_r1);
210     cudaFree(dev_g1);
211     cudaFree(dev_b1);
212     cudaFree(dev_r2);
213     cudaFree(dev_g2);
214     cudaFree(dev_b2);
215     cudaFree(dev_r3);
216     cudaFree(dev_g3);
217     cudaFree(dev_b3);
218
219     return 0;
220 }
```

Output

Complement

Figure 3: img

Contrast

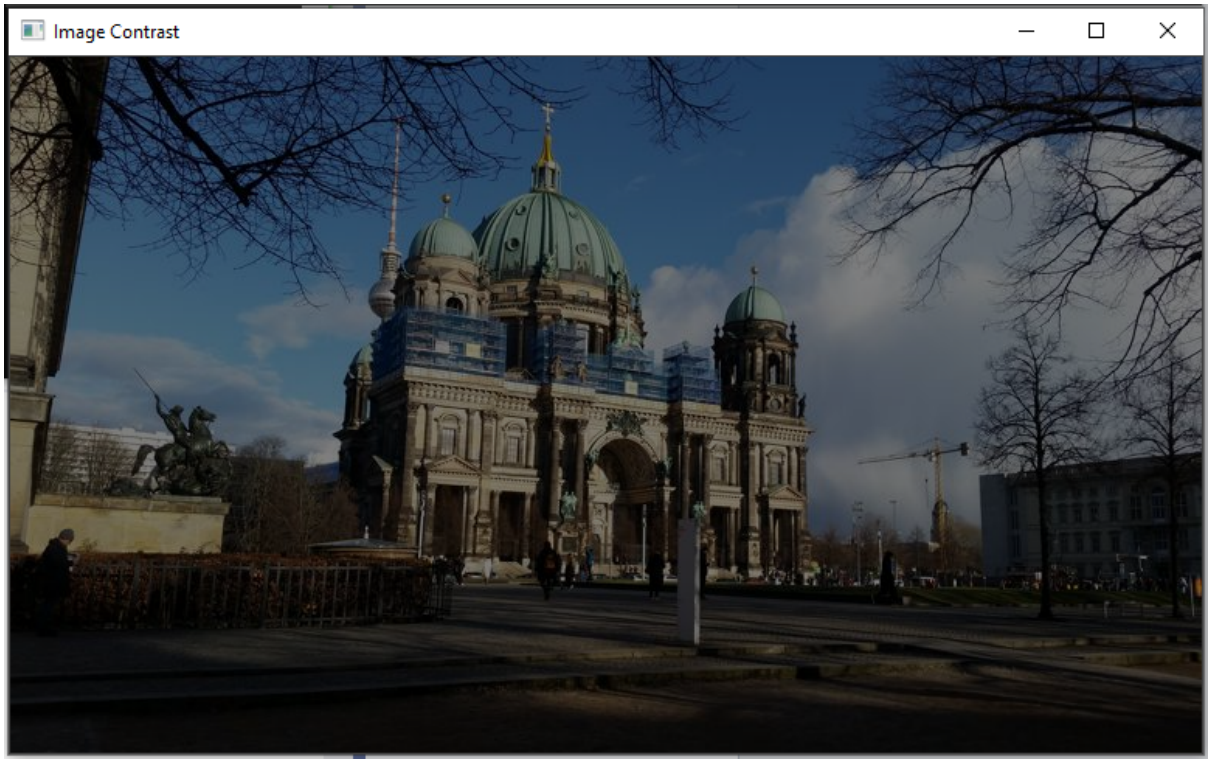

Figure 4: img

Brightness

Figure 5: img