GRAYSCALE - EDGE DETECTION
Labo Geavanceerde Computertechniek

(JLIZNM)

Jona Cappelle & Jonas Bolle

4 mei 2020

Sessie Datum:	9 Maart, 2020
Partners:	Jona Cappelle
	Jonas Bolle
Klas:	MELICTEES
Begeleider:	Stijn Crul

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

1 Inleiding

In dit labo van geavanceerde computertechniek gaan we een grayscale van een afbeelding maken en edge detection toepassen.

2 Probleemstelling

3 Analyse performantie

3.1 Grayscale

De bedoeling van het eerste deel van het labo is het omzetten van een kleurenafbeelding in een grayscale afbeelding. Het resultaat hiervan is terug te vinden in figuur 2. Bij de grayscale zijn er verschillende mogelijkheden om deze te implementeren. Men kan het gemiddelde nemen van de RGB waarden en dit gemiddelde toekennen aan elke R, G en B waarde. Er kan ook gewerkt worden met verschillende verhoudingen voor RGB waarden. In dit labo hebben we voor deze eerste optie gekozen.
De ingelezen afbeelding zetten we met 'lodepng' om naar een ééndimensionale array met structuur zoals weergegeven in 1.

image[i]	image[i+1]	image[i+2]	image[i+3]
Red	Green	Blue	Alpha

Figuur 1

Hier moeten we er rekening mee houden dat het 'alpha' kanaal, dat de opacity bepaald, altijd de waarde 255 moet hebben, om de afbeelding niet doorzichtig te maken.

3.1.1 Tijd op GPU

We meten de uitvoertijd op de GPU bij verschillende blocksizes. Hier werd het copiëren van de data van en naar de GPU niet mee in rekening gebracht. De laagste tijden worden bekomen bij blocksizes met veelvouden van 32. Een grafiek hiervan is terug te vinden in figuur 3.
De tijd die het duurt om de grayscale uit te voeren is:

$$t_{GPU} = 304\,\mu s$$

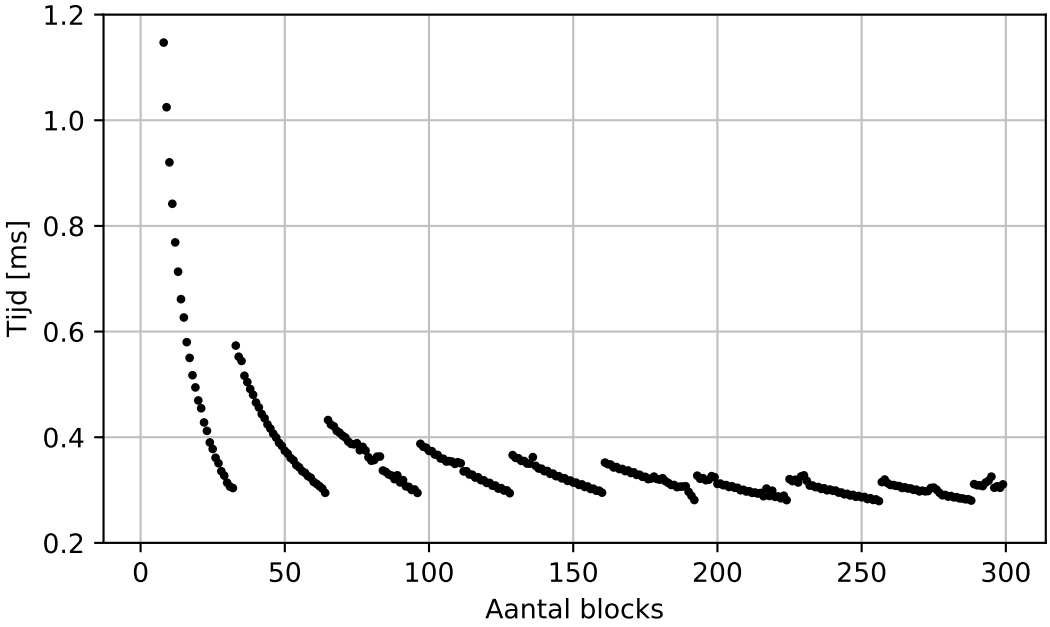
3.1.2 Tijd op CPU

Wanneer we een gray scale implementatie op de CPU schrijven, duurt het veel langer om deze uit te voeren namelijk:

$$t_{CPU} = 656\,\mu s$$

(a) Input (b) Output

Figuur 2: Resultaat grayscale



Figuur 3: Meting tijd op GPU bij verschillende bocksizes

Dit is meer dan dubbel zo lang dan de uitvoering op de GPU (zonder mee copiëren van de data). Bij een groot aantal fotos die omgezet moeten worden is de GPU met zijn parallelisatie veel sneller.

3.2 Edge detection

In het tweede deel van het labo gaan we edge detection toepassen op een afbeelding. Het resultaat van deze operatie is terug te vinden in figuur 4. Edge detection is een mooi vervolg op grayscale, daar we de grayscale toch ook nodig hebben om aan edge detection te doen. Voor de edge detection werd gewerkt met de gekende sobel filter. Het werkt op basis van een 3x3 matrix vermenigvuldiging met elke pixel van het beeld. Wanneer er veel verschil is naburige waarden van pixels, zal deze bewerking ofwel een zeer groot, of een zeer klein resultaat opleveren. We maken gebruik van twee 3x3 matrices, één voor de veranderingen in de x-richting te detecteren (G_x) en één voor de veranderingen in de y-richting te detecteren (G_y). Om deze waarden samen te voergen (G) en negatieve getallen te vermijden, wordt hier ook de wortel van de kwadraten van beide x en y resultaten genomen.

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

$$\mathbf{G} = \sqrt{{\mathbf{G}_x}^2 + {\mathbf{G}_y}^2}$$

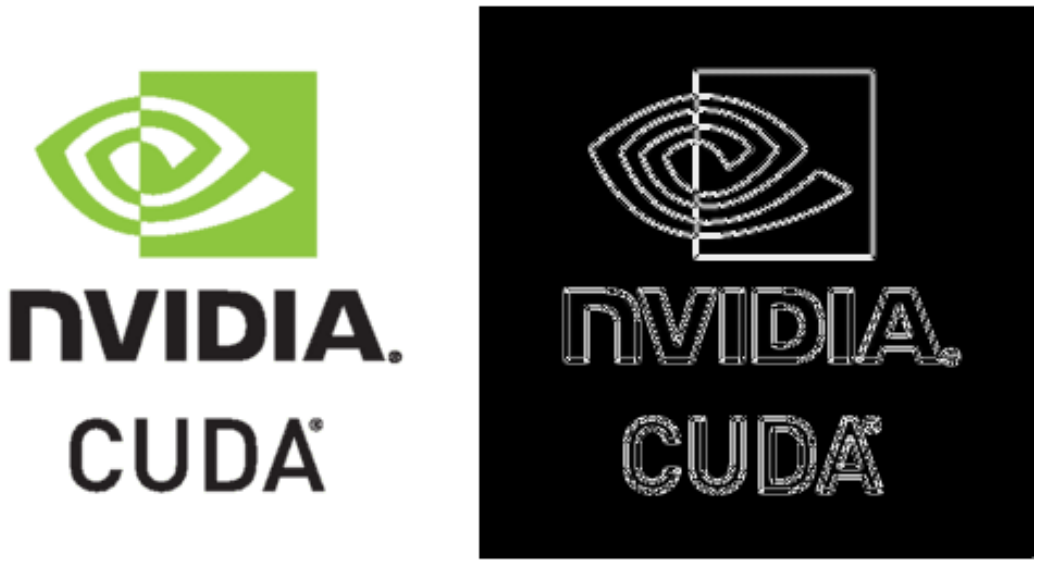
[1]

3.2.1 Tijd op GPU

Als we de edge detectie op de GPU uitvoeren, duurt het:

$$t_{GPU} = 336\,\mu\text{s}$$

3.2.2 Tijd op CPU



(a) Input (b) Output

Figuur 4: Resultaat edge detection

4 Besluit

A Grafieken

B CODE

```
1  ////////////////////////////////////////////////////////////////////////////
2  // GRAYSCALE - EDGE DETECTION -- Jona Cappelle -- Jonas Bolle
3  ////////////////////////////////////////////////////////////////////////////
4
5  // includes, system
6  #include <stdlib.h>
7  #include <stdio.h>
8  #include <string.h>
9  #include <math.h>
10
11 // includes CUDA
12 #include <cuda_runtime.h>
13
14 // includes, project
15 #include <helper_cuda.h>
16 #include <helper_functions.h> // helper functions for SDK examples
17
18 // own includes
19 #include "iostream"
20 #include "cstdlib"
21 #include "time.h"        // timing on cpu
22 #include "lodepng.h" // PNG afbeelding inlezen
23
24 extern "C"
25
26
27 ////////////////////////////////////////////////////////////////////////////
28 // SELECT GPU - CPU TIMING
29 // #define GPU
30 //#define CPU
31 ////////////////////////////////////////////////////////////////////////////
32 // RUN ADD - INV
33 // #define ADD
34 // #define INV
35 ////////////////////////////////////////////////////////////////////////////
36
37 // Helper function
38
39
40
41 void decodeOneStep(const char* filename) {
```

```
42          unsigned error;
43          unsigned char* image = 0;
44          unsigned width, height;
45
46          error = lodepng_decode32_file(&image, &width, &height,
            ↪  filename);
47          if(error) printf("error %u: %s\n", error,
            ↪  lodepng_error_text(error));
48
49          /*use image here*/
50
51          free(image);
52  }
53
54  void encodeOneStep(const char* filename, const unsigned char* image,
    ↪  unsigned width, unsigned height) {
55          /*Encode the image*/
56          unsigned error = lodepng_encode32_file(filename, image, width,
            ↪  height);
57
58          /*if there's an error, display it*/
59          if(error) printf("error %u: %s\n", error,
            ↪  lodepng_error_text(error));
60  }
61
62  ////////////////////////////////////////////////////////////////////////////////
63  // KERNEL GRAYSCALE
64  ////////////////////////////////////////////////////////////////////////////////
65  int BLOCKSIZE;
66
67  // GPU
68  //__global__ void grayscale(unsigned char* image, unsigned char*
    ↪  grayImage,unsigned width,unsigned height)
69  //{
70  //          int absolute_position_x =(blockIdx.x * blockDim.x) +
    ↪  threadIdx.x;
71  //          int absolute_position_y = (blockIdx.y * blockDim.y) +
    ↪  threadIdx.y;
72  //          if(absolute_position_x >= width || absolute_position_y >=
    ↪  height){
73  //                  return;
74  //          }
75  //
```

```
76  //        float channelSum = .299f * image[absolute_position_x +
    ↪  absolute_position_y * width]
77  //
    ↪  + .587f * image[(absolute_position_x + absolute_position_y *
    ↪  width)+1]
78  //
    ↪  + .114f * image[(absolute_position_x + absolute_position_y *
    ↪  width)+2];
79  //        grayImage[absolute_position_x + absolute_position_y * width] =
    ↪  channelSum;
80  //}
81
82  __global__ void grayscale(unsigned char* image, unsigned char*
    ↪  grayImage,unsigned width,unsigned height)
83  {
84          int j = (blockIdx.x*blockDim.x + threadIdx.x)*4;
85
86          if(j < width*height*4)
87          {
88                  grayImage[j] = (image[j]+image[j+1]+image[j+2])/3;
89                  grayImage[j+1] = (image[j]+image[j+1]+image[j+2])/3;
90                  grayImage[j+2] = (image[j]+image[j+1]+image[j+2])/3;
91                  grayImage[j+3] = 255;
92          }
93
94  }
95
96  // CPU
97  void grayscale_cpu(unsigned char* image, unsigned width, unsigned
    ↪  height)
98  {
99          printf("test1");
100
101         for(int j=0; j < (width*height*4); j+=4)
102         {
103                 image[j] = (image[j]+image[j+1]+image[j+2])/3;
104                 image[j+1] = (image[j]+image[j+1]+image[j+2])/3;
105                 image[j+2] = (image[j]+image[j+1]+image[j+2])/3;
106         }
107         printf("test2");
108 }
109
110
```

```
111
112 ////////////////////////////////////////////////////////////////////////////////
113 // Program main
114 ////////////////////////////////////////////////////////////////////////////////
115 int main()
116 {
117
118         printf("testest");
119
120
121         ////////////////////////////////////////////////////////////////////////////////
122         // Load PNG file
123         ////////////////////////////////////////////////////////////////////////////////
124         float millis = 0;
125         unsigned char *image_in, *image_out, *image_in_dev,
          ↪  *image_out_dev;
126         unsigned width, height, width_dev, height_dev;
127
128         printf("test-2");
129
130         const char* filename = "test.png";
131
132         unsigned error;
133         unsigned char* image = 0;
134
135
136         printf("test-1");
137
138
139
140         error = lodepng_decode32_file(&image, &width, &height,
          ↪  filename);
141         if(error) printf("error %u: %s\n", error,
          ↪  lodepng_error_text(error));
142
143
144
145         // allocate arrays on host
146         image_in = (unsigned char *)malloc(width*height*4 *
          ↪  sizeof(char));
147         image_out = (unsigned char *)malloc(width*height*4 *
          ↪  sizeof(char));
148
```

```
149
150 FILE *f = fopen("data.csv", "w");
151
152 for (int BLOCKSIZE = 1; BLOCKSIZE < 300; BLOCKSIZE++)
153 {
154         int nBlocks = (width*height*4) / BLOCKSIZE + ((width*height*4) %
        ↪  BLOCKSIZE == 0 ? 0 : 1);
155         printf("nBlocks: %d", nBlocks);
156         // image wordt goed geprint
157 //        for(int i=0; i<(width*height*4); i+=4)
158 //        {
159 //        printf("%u %u %u %u\n", image[i], image[i+1], image[i+2],
    ↪  image[i+3]);
160 //        }
161
162
163 //        StopWatchInterface *timer = 0;
164 //        sdkCreateTimer(&timer);
165 //        sdkResetTimer(&timer);
166 //        sdkStartTimer(&timer);
167
168 //        grayscale_cpu(image, width, height);
169
170 //        sdkStopTimer(&timer);
171 //        printf("Tijd: %f\n", sdkGetTimerValue(&timer));
172 //        sdkDeleteTimer(&timer);
173
174
175         //         // allocate arrays on device
176         cudaMalloc((void **)&image_in_dev, width*height*4 *
        ↪  sizeof(char));
177         cudaMalloc((void **)&image_out_dev, width*height*4 *
        ↪  sizeof(char));
178
179
180
181         cudaMemcpy(image_in_dev, image, width*height*4*sizeof(char),
        ↪  cudaMemcpyHostToDevice);
182         cudaMemcpy(image_out_dev, image_out,
        ↪  width*height*4*sizeof(char), cudaMemcpyHostToDevice);
183
184 //        unsigned *width_1 = &width;
```

```
185 //        cudaMemcpy(width_dev, width_1, sizeof(unsigned),
    ↪ cudaMemcpyHostToDevice);
186 //        cudaMemcpy(height_dev, &height, sizeof(unsigned),
    ↪ cudaMemcpyHostToDevice);
187
188         printf("Dit is van de CPU");
189         for(int i=100*4; i<4*120; i+=4)
190         {
191         printf("%u %u %u %u\n", image[i], image[i+1], image[i+2],
            ↪ image[i+3]);
192         }
193
194
195         // Record time on GPU with cuda events
196         cudaEvent_t start, stop;
197         cudaEventCreate(&start);
198         cudaEventCreate(&stop);
199
200
201
202
203         cudaEventRecord(start);
204         grayscale <<< nBlocks, BLOCKSIZE >>> ( image_in_dev,
            ↪ image_out_dev, width, height );
205         cudaEventRecord(stop);
206
207         cudaMemcpy(image_in, image_in_dev, width*height*4*sizeof(char),
            ↪ cudaMemcpyDeviceToHost);
208         cudaMemcpy(image_out, image_out_dev,
            ↪ width*height*4*sizeof(char), cudaMemcpyDeviceToHost);
209 //        cudaMemcpy(width, width_dev, sizeof(unsigned),
    ↪ cudaMemcpyDeviceToHost);
210 //        cudaMemcpy(height, height_dev, sizeof(unsigned),
    ↪ cudaMemcpyDeviceToHost);
211
212         printf("Dit is van de GPU");
213         for(int i=100*4; i<4*120; i+=4)
214                 {
215                 printf("%u %u %u %u\n", image_out[i], image_out[i+1],
                    ↪ image_out[i+2], image_out[i+3]);
216                 }
217
218         cudaEventSynchronize(stop);
```

```
219         cudaEventElapsedTime(&millis, start, stop);
220
221         printf("Tijd op GPU: %f\n", millis);
222
223         fprintf(f, "%d,%f\n", BLOCKSIZE, millis);
224
225 //        cudaEventDestroy(stop);
226 //        cudaEventDestroy(start);
227
228 }
229 fclose(f);
230
231         const char* output_filename = "output.png";
232         encodeOneStep(output_filename, image_out, width, height);
233
234
235         free(image_in);
236         free(image_out);
237 //        free(width);
238 //        free(height);
239         cudaFree(image_in_dev);
240         cudaFree(image_out_dev);
241 //        cudaFree(width_dev);
242 //        cudaFree(height_dev);
243
244         printf("Done!");
245
246         return 0;
247 }
```

Referenties

[1] "Sobel operator - Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Sobel{_}operator