Verslag block-based filters

Jonas Van Der Donckt

16 mei 2018

1 Gebruik maken van global shared memory

Onderstaande code bevat hoe de filter matrix via global shared memory wordt doorgegeven aan de **sharpen** kernel. dit is op te merken aan het *__global* keyword voan de filterkernel. De afmetingen van de kernel worden ook meegegeven in de argumenten. Hierdoor is dit een redelijk generieke kernel. Deze kan dus ook gebruikt worden voor de **gaussian blur**.

```
1   __kernel void sharpen(__global const int* inputImage, __global int* outputImage,
        __global const char* filterKernel, const int filterWidth, const int filterHeight){
2
3       size_t y = get_global_id(1);
4       y = (y < filterHeight/2) ? 1 : y;
5       y = ((get_global_size(1) - 1 - filterHeight/2) < y) ? (get_global_size(1) - 1 -
            filterHeight/2) : y;
6
7       size_t x = get_global_id(0);
8       x = (x < filterWidth/2) ? filterWidth/2 : x;
9       x = ((get_global_size(0) - 1 - filterWidth/2) < x) ? (get_global_size(0) - 1 -
            filterWidth/2) : x;
10      int r = 0, g = 0, b = 0;
11      for (int row = 0; row < filterHeight; row++){
12          for (int col = 0; col < filterWidth; col++){
13              int color = inputImage[get_global_size(0) * (y - 1 + row) + (x - 1 + col)];
14              r += filterKernel[row*filterWidth + col] * (color & 0xFF0000);
15              g += filterKernel[row*filterWidth + col] * (color & 0x00FF00);
16              b += filterKernel[row*filterWidth + col] * (color & 0x0000FF);
17          }
18      }
19
20      // keeping the r, g and b values in bound
21      r = (r > 0xFF0000) ? 0xFF0000 : r;
22      r = (r < 0x00FFFF) ? 0 : r;
23      g = (g > 0xFF00) ? 0xFF00 : g;
24      g = (g < 0x00FF) ? 0 : g;
25      b = (b > 0xFF) ? 0xFF : b;
26      b = (b < 0) ? 0 : b;
27
28      const size_t rowcol = (get_global_id(1) * get_global_size(0) + get_global_id(0));
29      outputImage[rowcol] = (inputImage[rowcol] & 0xFF000000) | r | g | b;
30  }
```

2 Gebruik makend van local memory

Aangezien dit project redelijk laat (in het semester) viel, had ik niet veel tijd om alle parameters van de workgroups te onderzoeken. Er is slechts èèn filter uitgewerkt die dit principe hanteert.

3 Vergelijking uitvoeringstijd CPU-GPU

Voor elke filter werden er 6 resultaten opgenomen. De uitvoertijden zijn samengevat in onderstaande tabel.

	gem CPU	**stdv CPU**	**gem GPU**	**stdv GPU**	**(gem) CPU / GPU**
blur	35 579	360	4 352	596	8.17
blur loc mem	35 579	360	3 475	375	10.22
sharpen	45 409	3 650	3 005	23	15.11
median	154 564	6 320	5 042	65	30.86
sobel	69 235	1 953	3 086	518	22.43

Tabel 1: Alle gemmidelden en standaard deviaties in us

Het valt op dat er een significante hoeveelheid tijd gewonnen wordt door de algoritmes uit te voeren op de GPU. Uit de verkregen data blijkt dat de sharpen filter het best optimaliseerbaar is voor de GPU.

De *blur loc mem* filter doet beroep op het lokaal geheugen van de de workgroups. Hierdoor zal deze sneller werken. Er wordt ongeveer 25% snelheidswinst waargenomen t.o.v. het global shared memory algoritme.