Verslag pixel-per-pixel filters

Jonas Van Der Donckt

8 mei 2018

1 Het framework

Na het schrijven van de code, kwam ik tot de conclusie dat het framework niet naar behoren werkte. De uitvoertijd van elk algoritme was 10 000-den miliseconden. Dit terwijl het resultaat zeer snel weergegeven werd.

Een mede linux gebruiker die de module vorig jaar opnam, wist me te vertellen dat de uitvoertijden in microseconden weergegeven worden.

Bij de uitvoering op de CPU gebeurde het af en toe dat de thread van fysieke core gewisseld werd. Dit zorgt ervoor dat de CPU verhouding tussen de gemiddelde-waarde en deviate aanzienlijk groter is dan deze verhouding bij de GPU tijden. Hierdoor moet men de CPU output met een korreltje zout nemen.

2 Optimalistaite van het invert algoritme

Voor het invert algoritme op de GPU werd volgende code gebruikt.

```
1   __kernel void invert(__global const int* inputImage, __global int* outputImage){
2           const size_t rowcol = (get_global_id(1) * get_global_size(0) +
                get_global_id(0));
3           int color = inputImage[rowcol];
4
5           int red  = (color & 0xFF0000) >> 16;
6           int green = (color & 0x00FF00) >> 8;
7           int blue = (color & 0x0000FF) >> 0;
8           int alpha_non_shift = (color & 0xFF000000);
9
10          red   = (255 - red)  << 16;
11          green = (255 - green) << 8;
12          blue  = (255 - blue) << 0;
13          int newColor = alpha_non_shift | red | green | blue;
14          outputImage[rowcol] = newColor;"
15      }
```

De aandachtige lezer zal al opgemerkt hebben dat er hier 3 keer afgetrokken wordt. Er vinden ook 6 bit shifts en 7 logische operaties plaats in dit stuk code. De gemiddelde uitvoertijd is 51 991 us.

Na volgende optimalistie worden er slecht 1 aftrekking, 0 bit shifts en 3 logische operaties uitgevoerd.

```
1       __kernel void invert(__global const int* inputImage, __global int* outputImage){
2           const size_t rowcol = (get_global_id(1) * get_global_size(0) +
                get_global_id(0));
3           int color = inputImage[rowcol];
4             int rgb  =  0xFFFFFF - (color & 0xFFFFFF);
5             int alpha_non_shift = color & 0xFF000000;
6           int newColor = alpha_non_shift | rgb;
7           outputImage[rowcol] = newColor;
8         }
```

Er wordt nu een uitvoertijd van 29 202 us bekomen. Bij beide metingen van de uitvoertijden was de standaard deviate relatief groot. Er kan dus geen concreet besluit uit de data worden opgemaakt.

3 Ontwerpkeuzes bij de greenscreen filter

Er is gekozen om de wortel operatie bij het berekenen van de kleur afstand niet uit te voeren.Hierdoor wordt er (zowel bij de GPU als CPU) wat tijd bespaard.

Het is dan ook logisch dat de minimum en maximum delta gekwadrateerd moeten worden.

4 Ontwerpkeuzes bij de hue shift filter

De hue-kleurruimte wordt bij de berekening niet naar graden geconvergeerd. Hierdoor wordt er 1 vermenigvuldig en 1 deel operatie bespaard. De code is wel wat minder leesbaar geworden.

5 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**
invert	192 859	1128 262	44 206	29 018	4.36
desaturate	237 092	118 717	34 129	18 010	6.95
green screen	99 938	63 781	11 238	5 281	8.89
hue shift	99 314	1 472	12 838	518	7.74

Tabel 1: Alle gemmidelden en standaard deviaties in us

Het valt op dat er inderdaad tijd gewonnen wordt door de algoritmes uit te voeren op de GPU. Uit de verkregen data blijkt dat de green screen filter het best optimaliseerbaar is voor de GPU. Door de grote standaardeviaties worden er geen verdere conclusies getrokken.