Meetrapport Snelheid

Doel

In dit rapport wordt er gekeken naar de snelheid van de implementatie van de edge detection.
Dit rapport is er om een vergelijking te maken tussen de oude en de nieuwe implementatie.
De meetresultaten worden bepaald door de tijdmeting vanaf de aanroep van de functies tot het einde van de edge detection functie.

De vraag die daarbij hoort is als volgt:

Hoe snel is de door ons geïmplementeerd vorm van edge detection in vergelijking met de standaard implementatie?”

Hypothese

Eerlijk gezegd word er geen vooruitgang verwacht in snelheid, want er worden veel meer functies aangeroepen. Dit zou dus een stuk langer moeten duren. Standaard wordt de openCV library gebruikt, die erg snel en geoptimaliseerd is.

Werkwijze

Elke stap in de edge detection gaat gemeten worden in nanosecondes. Uiteindelijk worden deze waardes ook opgeteld om zo een makkelijke vergelijking te maken.

Bij iedere uitvoerende functie word een clock aangezet. Met een simpele

“auto begin = std::chrono::high_resolution_clock::now();”
Helaas kunnen we deze niet tussentijds uitprinten, want een cout is super traag. De waardes worden daarom opgeslagen in 5 verschillende long long ints.

Om de volledige tijd in nano seconden te printen kun je dan ook alle long long ints bij elkaar optellen.

Na het runnen van alle functies kunnen deze dan ook worden geprint.

Resultaten

Een gemiddelde nemen uit het 100 keer toepassen van het algoritme komt uit op de volgende resultaten:

Eigen implementatie:

```
Gaussian blur lambda creation (average out of 100 ) -> elapsed nano seconds = 474
Gaussian blur kernel generator (average out of 100 ) -> elapsed nano seconds = 8452
Edge kernel generator (average out of 100 ) -> elapsed nano seconds = 6309
IntensityImage to matrix conversion (average out of 100 ) -> elapsed nano seconds = 2548578
Gaussian blur convolution (average out of 100 ) -> elapsed nano seconds = 97353276
Edge kernel convolution (average out of 100 ) -> elapsed nano seconds = 310414278
Thresholding (average out of 100 ) -> elapsed nano seconds = 2669520
Matrix to IntensityImage ptr converting (average out of 100 ) -> elapsed nano seconds = 3756468
Total (average out of 100 ) -> elapsed nano seconds = 416757355
```

416.757.355

Standaard implementatie edge detection:

```
Creating empty math (average out of 100 ) -> elapsed nano seconds = 1571
IntensityImage to mat conversion (average out of 100 ) -> elapsed nano seconds = 4960270
Creating edge kernel (average out of 100 ) -> elapsed nano seconds = 21700
Creating empty math (average out of 100 ) -> elapsed nano seconds = 1402
Edge kernel convolution (average out of 100 ) -> elapsed nano seconds = 671029
Math to IntensityImage prt converting (average out of 100 ) -> elapsed nano seconds = 4846498
Total (average out of 100 ) -> elapsed nano seconds = 10502470
```

10.502.470

Standaard implementatie thresholding:

```
Creating empty math (average out of 100 ) -> elapsed nano seconds = 1298
IntensityImage to mat conversion (average out of 100 ) -> elapsed nano seconds = 4238660
Thresholding (average out of 100 ) -> elapsed nano seconds = 4238660
mat to IntensityImage conversion (average out of 100 ) -> elapsed nano seconds = 4246412
Total (average out of 100 ) -> elapsed nano seconds = 12725030
```

12.725.030 (Totaal 23.227.500)

Conclusie

Door de resultaten te bekijken kunnen we zien dat de eigen implementatie een stuk trager is, dit was zoals al eerder genoemd wel te verwachten. Natuurlijk zouden deze getallen afvlakken als je ze in plaats van 100 keer, 1000000 keer zou runnen. Maar dat is de CPU tijd het niet waard.

De hypothese is waar, we zijn een stuk trager, zo'n 18 keer trager. Dit zit grotendeels in de convolution die in de nieuwe implementatie extreem veel trager is.

De onderzoeksvraag valt dan ook te beantwoorden met het volgende antwoord; we zijn ongeveer 18x keer trager in het uitvoeren van het algoritme met onze eigen implementatie.

Evaluatie

Bij het maken van het implementatieplan was het doel duidelijk. Het doel om een werkende vorm van edge detection toe te passen is gelukt. Jammer genoeg is de convolution zeer traag, en daardoor ook onze gehele implementatie. Dit had verbeterd kunnen worden door een andere techniek van convolution toe te passen. Er zijn zeker andere manieren, waarvan we niet weten hoe deze werken. De implementatie zou dan teveel copy paste worden, en daar leer je niets van.