Meetrapport

Snelheid

Jeroen
Stefan
22-03-2017

Inhoudsopgave

Doel

Het doel van dit meetrapport is om te kijken welke van de gekozen methodes beschreven in het implementatieplan de snelste is. Des te minder tijd onze code kost om te runnen, des te beter.

Hypothese

Wij vermoeden dat BGD het snelst is van alle methoden omdat BGD de kleinste kernel is. Onder andere omdat BGD geen ruisfiltering heeft, kan het mooi in een 1*3 kernel die 2x wordt gerund. In tegenstelling tot Sobel's 3x3 kernel die twee keer moet worden gebruikt.

Werkwijze

Wij zullen op snelheid testen door de ExternalDLL.exe. via een Python script 10.000 keer op dezelfde foto (debug.png, te vinden in de appendix) aan te roepen. Van elke run van de ExternalDLL.exe krijgen wij de tijd in microseconden terug van het edge detection gedeelte(zonder de thresholding) door vervroegt een exit() aan te roepen met als parameter de tijd die de edge-detection code heeft gekost. In the Python script voegen wij deze waarden toe aan een lijst waar wij het gemiddelde, het minimum en het maximum opvragen. De minimum en maximum gebruiken wij om mogelijke uitschieters te herkennen, maar we gebruiken het gemiddelde om over de uiteindelijke snelheid te oordelen.

Stappenplan:
Voor het uitvoeren van het python script is python3 nodig.

1. In het python script moet de "exe_path" variabele naar de Externalldll.exe
2. Daarna moet in de main functie van de code de variabelen "debugFileDir" en "pictureDir" aangepast worden.
3. Run het python programma (dat 10.000 keer het Externalldll.exe gaat uitvoeren en de tijden bijhoud)
4. Kijk na het runnen de gegevens na en houd de min, max en average bij in het tabel.

Voor de tests runnen wij deze op onze eigen PC één keer. Hieronder zijn de specificatie van onze computers

computer	**cpu**	**gpu**	**ram**	**OS**
Stefan	intel i7-4720hq 2.6Ghz	nvidia geforce gtx 960m	8Gb 1600Mhz	windows 10
Jeroen	Intel I5-6600K 3.50GHz	NVIDIA GeForce GTX 760	16Gb DDR4 2.4GHz	windows 10

Resultaten

	Average Millisecond/run		Minimum Millisecond/run		Maximum Millisecond/run	
	Stefan	Jeroen	Stefan	Jeroen	Stefan	Jeroen
Orginele code	0.996	0.7856205	0.8	0.685	1.894	19.529
Originele kernel	16.3950648	12.5989881	14.87	11.544	29.659	98.883
BGD	5.764	3.7060111	5.386	3.423	8.005	35.433
Sobel	5.730	2.4922526	5.131	2.184	7.864	40.24
Laplacian	3.545	2.309039	2.524	2.169	8.979	37.544

Verwerking

Wat opvalt is dat de maximum bij Jeroen erg extreem uitslaan en bij Stefan niet. Vermoedelijk omdat er in de achtergrond veel andere apps draaien. Omdat het gemiddelde en de minima er verder normaal uitzien, laten we dit feit verder liggen.

De originele code is tussen de ±3 en ±6 keer zo snel als BGD, Sobel en Laplacian. De originele kernel met onze code is ±16 keer zo traag als de orginele code, maar levert exact dezelfde resultaten op. Wij vermoeden dat dit komt omdat de orginele code gebruik maakt van hardware acceleratie via de GPU en onze code niet.

BGD is trager dan laplacian omdat BGD maar 1 richting op gaat(verticaal of horizontaal), Hierdoor moet voor elke pixel 2 keer de horizontale en de verticale richting apart berekend worden. De resultaten van deze berekening moet daarna samengevoegd worden tot de waarde van 1 pixel. Het samenvoegen is geen goedkope berekening, namelijk pythagoras. Het verschil tussen BGD en Laplacian is ±2 milliseconden terwijl laplacian maar ±3 milliseconden duurt,

Conclusie

Onze snelste code is niet sneller dan de code die origineel in het programma te vinden is omdat de originele code gebruikt maakt van hardware acceleratie via de GPU. Tenzij wij ook toegang krijgen tot deze functies zal onze code nooit sneller dan de al bestaande.
Wel is het duidelijk dat de BGD kernel niet sneller is dan Sobel en Prewitt omdat het samenvoegen van het plaatje duurder is dan de plekken in de kernel die wij niet mee hoeven te berekenen bij kernels van deze grootte.

Evaluatie

Het resultaat is niet zoals wij hadden verwacht. Wij dachten dat de originele code trager was. Maar dit kwam omdat wij destijds dit niet wisten dat de originele code gebruik maakte van GPU versnelling.
Ook was BGD niet sneller dan bijv. Laplacian om de redenen die wij dachten. Omdat wij de algoritmische kosten van pythagoras onderschat hebben.

Appendix

Debug.png (225x225)