Meetrapport StepToIntensityImage speed

1. Namen en datum

Stefan van der Ham & Bas van Rossem, 6 april 2019.

2. Doel

We willen met dit experiment kijken welke van de twee implementaties sneller is. De default implementatie van de setpToIntensityImage tegenover onze implementatie van de setpToIntensityImage.

3. Hypothese

Wij verwachten tot onze implementatie 50% sneller is, omdat het een vrij simpele oplossing is. Wij verwachten dat wij maar een paar minuten bezig zijn, omdat greyscaling heel snel is en onze resultaten automatisch worden gegenereerd.

4. Werkwijze

4.1. Algemeen

Om de snelheid van de imageshell te testen wordt er gebruik gemaakt van de vision timer gemaakt door Arno Kamphuis (https://github.com/arnokamphuis/vision-timer). Hoe deze wordt geïmplementeerd wordt uitgelegd in 4.2 code uitleg.

De test wordt gedaan met afbeelding “child-1.png”, en wordt gedraaid in de debug mode van visual studio 2017.

4.2. Code uitleg

De imageshell wordt getest door de main, die de greyscaling uitvoert, meerde malen uit te voeren en die gegevens automatisch op te slaan in een .csv bestand.

We doen 5 tests achter elkaar waarbij het aantal keer dat de aangepaste main wordt uitgevoerd met een factor van wordt 5 vermenigvuldigd.

De aangepaste main voert zichzelf 10 keer uit. De aanpassing die aan de main is gemaakt is het volgende: zowel de default implementatie en de studentimplementatie worden baseN * mainiteratie maal uitgevoerd. Hierbij is baseN 5 * op welke iteratie van de test we zitten (1 t/m 5) en de mainiteratie is hoe vaak de main is uitgevoerd (1 t/m 10). De tijd die het programma erover doet om die vermenigvuldiging van die iteraties uit te voeren wordt opgeslagen het eerder besproken .csv bestand. Zo hebben wij volledig geautomatiseerde tests.

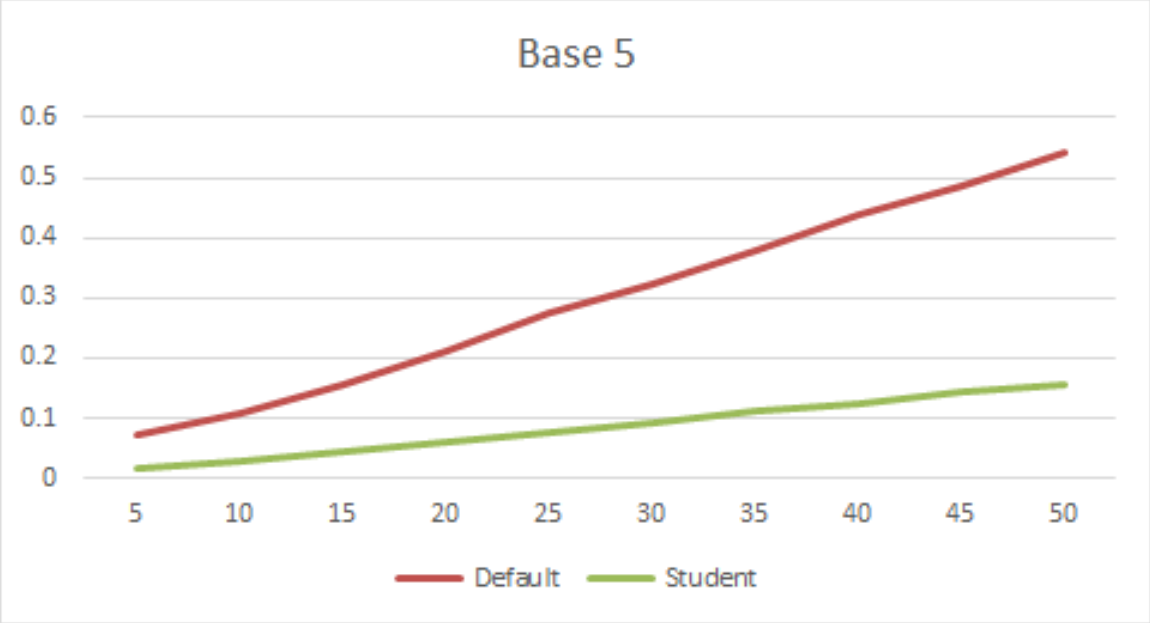
4.3. Code uitleg

Onze code heeft dus de volgende opbouw:

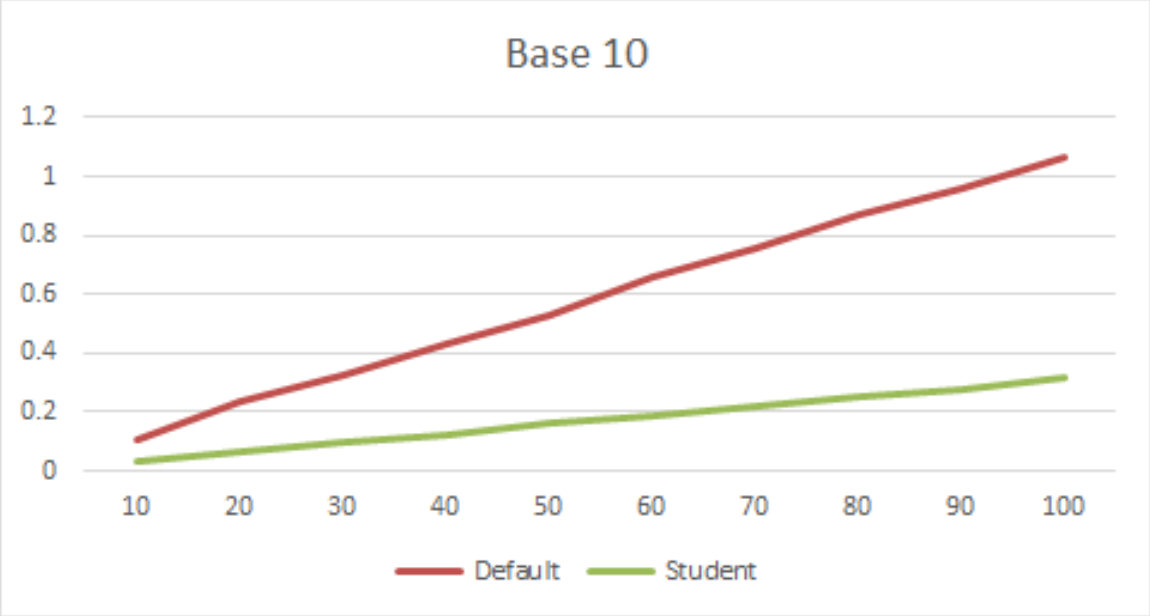
```
int main(int argc, char * argv[]) {
	for (int mainIteration = 1; mainIteration <= 5; mainIteration++) {
		baseN = 5 * mainIteration
		myBaseTimer = basetimer

		for (int iteratorCount = startingI; iteratorCount <= 10; iteratorCount++) {
			ImageFactory::setImplementation(ImageFactory::DEFAULT);

			myBaseTimer->start();
			for (int i = 0; i < baseN * iteratorCount; i++) {
				if (!executorStudent->executePreProcessingStep1(false)) {
					std::cout << "Step 1 failed!" << std::endl;
					return false;
				}
			}

			myBaseTimer->stop();

			resultFile.write(results)

				myBaseTimer->reset();

			ImageFactory::setImplementation(ImageFactory::STUDENT);

			myBaseTimer->start();

			for (int i = 0; i < baseN * iteratorCount; i++) {
				if (!executorStudent->executePreProcessingStep1(true)) {
					std::cout << "Step 1 failed!" << std::endl;
					return false;
				}
			}

			myBaseTimer->stop();

			resultFile.write(results)

		}
	}
}
```

5. Resultaten

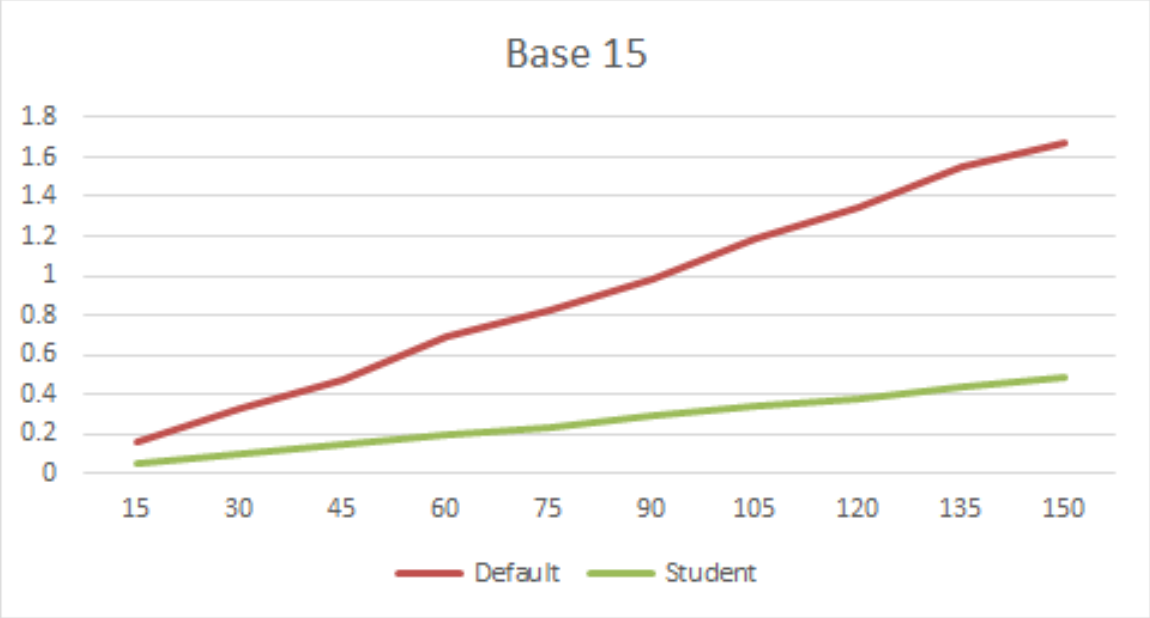
Base 5	***Cycles***	***Default (in seconden)***	***Student (in seconden)***	***Verschil (in seconden)***	***Gewogen verschil (in seconden per cycle)***
	5	0.0711942	0.0147523	0.0564419	0.01128838
	10	0.109603	0.0301407	0.0794623	0.00794623
	15	0.155237	0.0446803	0.1105567	0.007370447
	20	0.209851	0.0582643	0.1515867	0.007579335
	25	0.275406	0.0755018	0.1999042	0.007996168
	30	0.324032	0.093569	0.230463	0.0076821
	35	0.380613	0.110455	0.270158	0.0077188
	40	0.437505	0.121914	0.315591	0.007889775
	45	0.486853	0.142214	0.344639	0.007658644
	50	0.542646	0.157298	0.385348	0.00770696
Total	**275**	**2.9929402**	**0.8487894**	**2.1441508**	**0.008083684**
Student % sneller	**71.64028**				



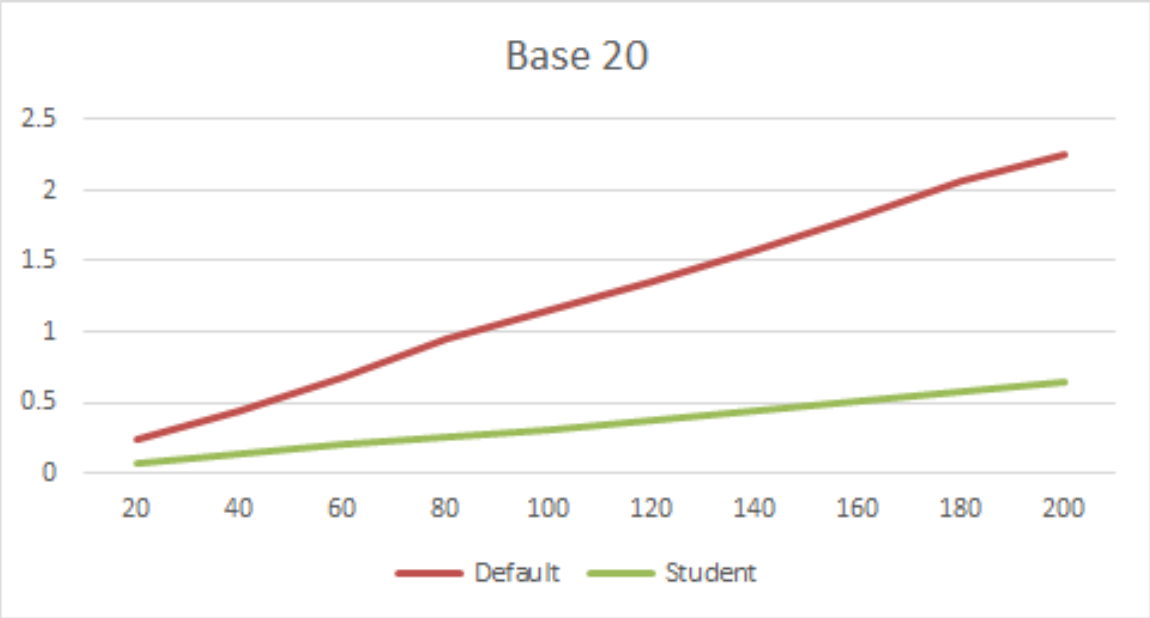
Base 10	***Cycles***	***Default (in seconden)***	***Student (in seconden)***	***Verschil (in seconden)***	***Gewogen verschil (in seconden per cycle)***
	10	0.107806	0.0311619	0.0766441	0.00766441
	20	0.235118	0.0659748	0.1691432	0.00845716
	30	0.323751	0.0986977	0.2250533	0.007501777
	40	0.433059	0.125047	0.308012	0.0077003
	50	0.530945	0.16325	0.367695	0.0073539
	60	0.655793	0.190843	0.46495	0.007749167
	70	0.75386	0.219424	0.534436	0.0076348
	80	0.869954	0.248945	0.621009	0.007762613
	90	0.960242	0.274066	0.686176	0.007624178
	100	1.06925	0.316958	0.752292	0.00752292
Total	**550**	**5.939778**	**1.7343674**	**4.2054106**	**0.007697122**
Student % sneller	**70.8008**				



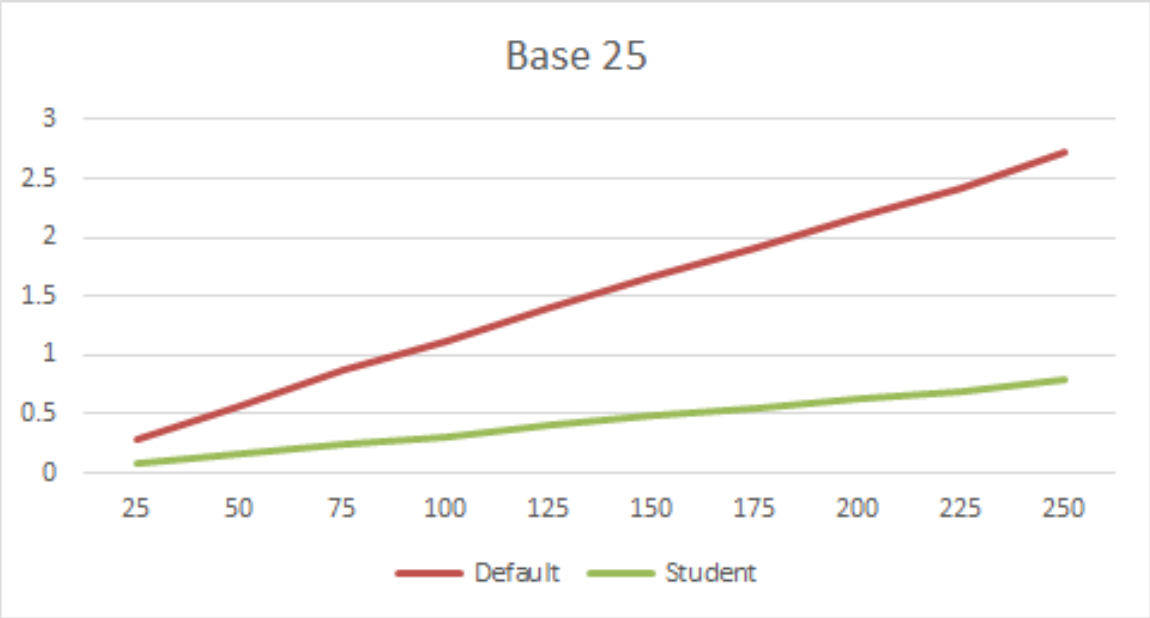
Base 15	***Cycles***	***Default (in seconden)***	***Student (in seconden)***	***Verschil (in seconden)***	***Gewogen verschil (in seconden per cycle)***
	15	0.158452	0.0466589	0.1117931	0.007452873
	30	0.32474	0.0924052	0.2323348	0.007744493
	45	0.476703	0.142003	0.3347	0.007437778
	60	0.69115	0.193031	0.498119	0.008301983
	75	0.822616	0.23665	0.585966	0.00781288
	90	0.981854	0.287708	0.694146	0.007712733
	105	1.18309	0.334434	0.848656	0.008082438
	120	1.3494	0.381192	0.968208	0.0080684
	135	1.54517	0.438379	1.106791	0.008198452
	150	1.67576	0.487798	1.187962	0.007919747
Total	**825**	**9.208935**	**2.6402591**	**6.5686759**	**0.007873178**
Student % sneller	**71.32938**				



Base 20	***Cycles***	***Default (in seconden)***	***Student (in seconden)***	***Verschil (in seconden)***	***Gewogen verschil (in seconden per cycle)***
	20	0.231021	0.0627158	0.1683052	0.00841526
	40	0.448332	0.130501	0.317831	0.007945775
	60	0.679356	0.205471	0.473885	0.007898083
	80	0.94898	0.256977	0.692003	0.008650038
	100	1.1609	0.313274	0.847626	0.00847626
	120	1.36133	0.378878	0.982452	0.0081871
	140	1.57819	0.437685	1.140505	0.008146464
	160	1.80965	0.509288	1.300362	0.008127263
	180	2.06759	0.58078	1.48681	0.008260056
	200	2.25535	0.648485	1.606865	0.008034325
Total	**1100**	**12.540699**	**3.5240548**	**9.0166442**	**0.008214062**
Student % sneller	**71.89906**				



Base 25	***Cycles***	***Default (in seconden)***	***Student (in seconden)***	***Verschil (in seconden)***	***Gewogen verschil (in seconden per cycle)***
	25	0.285139	0.0788945	0.2062445	0.00824978
	50	0.570283	0.162289	0.407994	0.00815988
	75	0.882061	0.247844	0.634217	0.008456227
	100	1.11328	0.314665	0.798615	0.00798615
	125	1.41144	0.403592	1.007848	0.008062784
	150	1.67207	0.484527	1.187543	0.007916953
	175	1.9068	0.541342	1.365458	0.007802617
	200	2.1691	0.623956	1.545144	0.00772572
	225	2.411	0.702436	1.708564	0.007593618
	250	2.7156	0.790469	1.925131	0.007700524
Total	**1375**	**15.136773**	**4.3500145**	**10.786759**	**0.007965425**
Student % sneller	**71.26194**				



6. Verwerking

Hier zijn alle totaalresultaten bij elkaar. Het totaal bij gewogen verschil is het gemiddelde van het verschil.

	Cycles	***Default (in seconden)***	***Student (in seconden)***	***Verschil (in seconden)***	***Gewogen verschil (in seconden per cycle)***
Base 5	275	2.9929402	0.8487894	2.1441508	0.008083684
Base 10	550	5.939778	1.7343674	4.2054106	0.007697122
Base 15	825	9.208935	2.6402591	6.5686759	0.007873178
Base 20	1100	12.540699	3.5240548	9.0166442	0.008214062
Base 25	1375	15.136773	4.3500145	10.786759	0.007965425
Total	**4125**	**45.819125**	**13.0974852**	**32.72164**	**0.007966694**
Student % sneller	**71.41481**				

Student % sneller is berekend door het totale verschil te delen door de totale default tijd.

7. Conclusie

De stepToIntensityImage implementatie van de student is 71.41% sneller dan de default implementatie. We kunnen dus concluderen dat de stepToIntensityImage implementatie die wij hebben gemaakt sneller is dan de default implementatie van de stepToIntensityImage.

8. Evaluatie

Het doel van dit experiment was om te kijken hoeveel sneller onze implementatie van de image shell was in vergelijking tot de default implementatie. We zaten er met onze hypothese 21 procentpunten naast, dus onze implementatie is best wat sneller dan we hadden verwacht.