1. Snelheid van storagetypes Imageshell

1.1. Namen en datum

Dit meetrapport is gemaakt op 14 april door Thijs Hendrickx en Job Verhaar

1.2. Doel

Het doel van het experiment is om te kijken hoe een zo snel mogelijke imageshell gemaakt kan worden. Hierbij wordt gekeken naar de snelheid van het initialiseren van een image en naar het resizen van een image. De twee mogelijkheden die worden getest zijn met betrekking tot het opslaan van RGBData. Er is gekozen om het verschil te testen tussen een vector en een normale array.

1.3. Hypotheses

Onze hypothese is dat een array in het initialiseren sneller is. Als er bij de vector geen operator[] maar vector::at() wordt gebruikt is een array ook sneller. Wel betekent het gebruik van at() voor meer robuustheid van het programma. Voor het resizen echter verwachten wij dat de vector sneller is.

1.4. Werkwijze

Allereerst is er een extra klasse gemaakt voor het timen van een functie. Functies zijn als volgt:

```
class Clock {
public:
	static timepoint now() {
		return high_resolution_clock::now();
	}
	static double getMicroSecondsFrom(timepoint& p) {
		return double(duration_cast<microseconds>(now() - p).count());
	}
	static double getMilliSecondsFrom(timepoint& p) {
		return double(duration_cast<milliseconds>(now() - p).count());
	}
	static double getNanoSecondsFrom(timepoint& p) {
		return double(duration_cast<nanoseconds>(now() - p).count());
	}
};
```

Om de snelheid van een functie getest moet worden kan vervolgens het volgende worden gedaan:

```
auto timepointBefore = Clock::now();
//Execute function(s)
auto durationOfFunction = Clock::getNanoSecondsFrom(timepointBefore);
```

Het experiment dat uitgevoerd gaat worden is het testen van de snelheid van het initialiseren van een RGBImage. En het resizen van een image. Beide functies worden een bepaald aantal keer uitgevoerd in een for loop en de resultaten zullen worden bekeken; Voor deze test zijn twee aparte klassen gemaakt: *RGBImageArray* en *RGBImageVector*.

1.5. Resultaten

De resultaten van de constructor test waren als volgt. Alle constructors zijn 200 keer uitgevoerd en het gemiddelde hiervan is genomen. Tijden zijn in microseconden.

Image Size	Array	Vector
200 x 200	60.699975	63.964165
400 x 400	252.235660	275.702340
600 x 600	648.709280	761.636300
800 x 800	1039.830680	927.058460
1000 x 1000	1395.689755	1386.280430
1200 x 1200	2119.143150	2120.447695

Vervolgens is de snelheid van het resizen getest er is een afbeelding gemaakt van bepaalde afmeting en vervolgens 200 keer geresized.

Resize values	Array	Vector
200 x 200 -> 400x400	201.845540	202.808375
400 x 400 -> 200x200	90.508135	103.857525
1000 x 1000 -> 500x500	651.040840	590.863815
500 x 500 -> 1000x1000	1331.857745	1294.044860

1.6. Verwerking

Duidelijk uit de meetresultaten is dat er geen duidelijk verschil ios tussen de snelheid van het ene storage type tegenover de andere. De waarden fluctueren heel erg en er zit geen duidelijke lijn in vergelijking tot de snelheid. De meetresultaten zijn al een gemiddelde van 200 keer dezelfde functie dus de echte reden achter het fluctueren is niet bekend

1.7. Conclusie

Aan de hand van de meetresultaten en verwerking kan worden geconcludeerd dat er geen duidelijk verband en verschil zichtbaar is met de gebruikte meetmethoden. Zo is er in de prestatie van een vector resizen geen duidelijk verschil zichtbaar in beide tests.

1.8. Evaluatie

Van te voren werd verwacht dat met de gekozen onderzoeksmethoden er aangetoond kon worden dat een array sneller was om te initialiseren maar slomer was in het resizen van een array. Achteraf blijkt dit niet aan te tonen met de gekozen onderzoeksmethode. In de toekomst moet er misschien een uitgebreidere test gedaan worden met meer dan 200 iteraties om zo verschil nog meer te middelen.