1. Meetrapport tijd.

1.1. Namen en datum

Dit document is gemaakt door:

- Cris van de Nolle
- Ramon van Bemmel

Op 14 april 2019.

1.2. Doel

Het doel van dit experiment is het bepalen van de snelheid van het door ons gekozen algoritme ten opzichte van de standaard versie.

1.3. Hypothese

Wij verwachten dat onze implementatie een fractie langzamer zal zijn dan de standaard versie. Dit omdat wij een gewogen schaal hebben en dus waarschijnlijk een extra bewerking uitvoeren. We verwachten wel dat het verschil minimaal is.

1.4. Werkwijze.

Om de snelheid van het programma te meten hebben wij gebruik gemaakt van de chrono library. Met deze library kan er een timer gestart en gestopt worden op verschillende plekken in een programma. Hieronder is te zien hoe er gebruik wordt gemaakt van de library.

```
bool executeSteps(DLLExecution * executor) {
    //create start variabele and end  variable.
    std::chrono::time_point<std::chrono::system_clock> startTime, endTime;

    //Set start time.
    startTime = std::chrono::system_clock::now();

    //Execute the four Pre-processing steps
    if (!executor->executePreProcessingStep1(true)) {
        std::cout << "Pre-processing step 1 failed!" << std::endl;
        return false;
    }

    //Set end time.
    endTime = std::chrono::system_clock::now();
    //Calculate elapsed time
    std::chrono::duration<double> elapsedTime = endTime - startTime;
    //print time.
    std::cout << "Program duration in seconds:\t" << elapsedTime.count() << "\n";
```

We switchen tussen de standaard implementatie en die van ons door executeproProcessingStep1 op true of false te zetten in de main.cpp. Bij true wordt de implementatie van ons gestart en bij false wordt de standaard implementatie gestart.

```
//Execute the four Pre-processing steps
if (!executor->executePreProcessingStep1(true)) {
    std::cout << "Pre-processing step 1 failed!" << std::endl;
    return false;
}
```

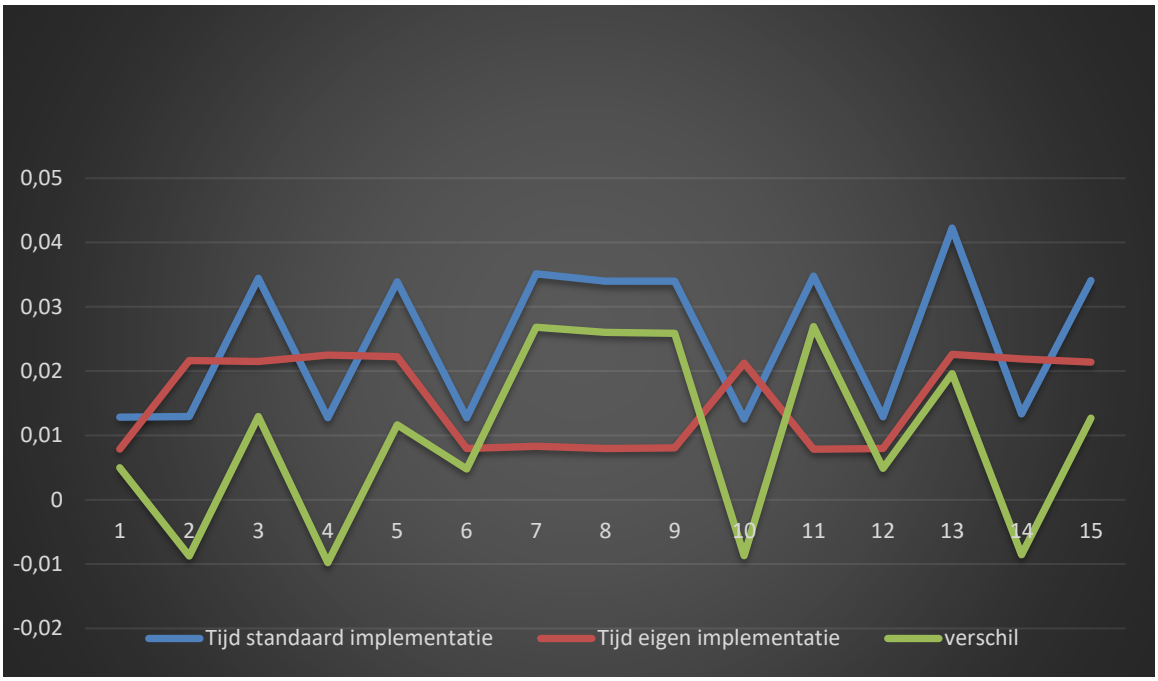
Om een betrouwbaar resultaat te krijgen moeten er meerdere tests gedaan worden en moet naar de gemiddelde waarde gekeken worden. Tijdens de test hebben wij gezorgd dat er geen onnodige andere programma's op de computer actief waren en gezorgd dat dit bij beide test hetzelfde was, zo kunnen andere programma's niet onze tests beïnvloeden.

1.5. Resultaten

Hieronder zijn de resultaten van de test te vinden in een tabel en grafiek. De tijd is steeds in secondes.

Test nummer	*Tijd standaard implementatie*	*Tijd eigen implementatie*	*verschil*
1	0,0128765	0,0078925	0,004984
2	0,0129389	0,0216756	-0,008737
3	0,0344667	0,0215345	0,0129322
4	0,0127504	0,0225176	-0,009767
5	0,0338871	0,0222426	0,0116445
6	0,0127517	0,0079634	0,0047883
7	0,0351598	0,0083069	0,0268529
8	0,0340222	0,00796	0,0260622
9	0,0340018	0,0081021	0,0258997
10	0,0125657	0,021277	-0,008711
11	0,0348095	0,0078723	0,0269372
12	0,0128833	0,0079852	0,0048981
13	0,0422563	0,0226227	0,0196336
14	0,013367	0,0219155	-0,008549
15	0,0341168	0,0213985	0,0127183

Gemiddelde tijd standaard implementatie: 0,024856
Gemiddelde tijd eigen implementatie: 0,0154178
Ons zelf geschreven implementatie is gemiddeld dus 0,0094382 seconde of te wel **9,4382 ms** sneller.



De test zijn gedraaid een laptop met de volgende specificaties:

Naam van besturingssysteem Microsoft Windows 10 Home
Systeemfabrikant ASUSTeK COMPUTER INC.
Systeemmodel N56VB
Processor Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz, 2401 MHz, 4 core('s), 8 logische processor(s)
Geïnstalleerd fysiek geheugen (RAM) 12,0 GB

1.6. Verwerking

Om de tijd te bereken hebben we in de code de start tijd van de eindtijd afgetrokken. De tijden van de tests hebben we bij elkaar opgeteld en door 15 gedeeld om de gemiddelde tijd te krijgen. De gemiddelde tijden hebben we van elkaar afgetrokken om zo het verschil in tijd te krijgen.

1.7. Conclusie

Uit deze test blijkt dat onze implementatie gemiddeld **9,4382 ms** sneller is dan de standaard implementatie. Uit het andere testrapport (meetrapport realistische grijswaarde) is ook gekomen dat onze implementatie betere resultaten opleverde met verschillende foto's.

1.8. Evaluatie

Wij hebben in onze test getest met het Luminosity algoritme wat een van de betere algoritmes is en die in het andere meetrapport de beste resultaten opleverde. Dit was ook gelijk het zwaarste algoritme en deze is dus alsnog sneller dan de standaard implementatie. Om een nog betrouwbaarder resultaat te krijgen zouden we deze tests ook nog op een andere pc kunnen doen en misschien de systeem belasting in de berekening mee kunnen nemen.