# 1 vs 2 dimensionale array

## Namen en datum

Wibren Wiersma, Pawel Niewolik; 27-04-2015

## Doel

In dit experiment wordt nagegaan welke vorm voor het opslaan van de foto het snelste is. Hierbij wordt een 1 dimensionale array met een 2 dimensionale array vergeleken.

## Hypothese

De verwachting is dat een 1 dimensionale array het snelste werkt, omdat deze minder geheugen instructies gebruikt. Dit omdat bij een 2 dimensionale array eerst het adres van de y lijst opgehaald moet worden voordat de pixel opgehaald kan worden.

## Werkwijze

Eerst zijn 20 testen met 1 dimensionale array list in de debug mode uitgevoerd, daarna in de release mode. Dezelfde testen worden met een 2 dimensionale array uitgevoerd. Uit die twintig metingen zal het gemiddelde worden genomen om te vergelijken.

De volgende code is gekozen om te testen. Om memory allocatie niet mee te nemen in de test is gekozen om van te voren al de image aan te maken. Maar om de memory allocatie van de array zelf wel mee te nemen wordt de image telkens weer terug gezet naar 0 bij 0. Bij het zetten van de afbeelding is gekozen om een copy te maken van een andere afbeelding waardoor er 20 keer een read en 20 keer een write wordt uit gevoerd.

RGBImageStudent \* copyImage = new RGBImageStudent();

for(int i = 0; i < 20; i++) {

bt.reset();

bt.start();

copyImage->set(\*image);

bt.stop();

std::cout << "Image copy time: " << bt.elapsedMicroSeconds() << "\n";

copyImage->set(0, 0);

}

## Resultaten

De waardes zijn in micro seconden

|  |  |  |  |
| --- | --- | --- | --- |
| **1D Debug** | **1D Release** | **2D Debug** | **2D Release** |
| 4049 | 126 | 14826 | 10867 |
| 3058 | 119 | 14847 | 11515 |
| 2987 | 118 | 15173 | 10663 |
| 3005 | 120 | 14790 | 10914 |
| 3001 | 124 | 14773 | 11137 |
| 2989 | 137 | 14702 | 10714 |
| 3039 | 123 | 14437 | 10867 |
| 2991 | 120 | 14312 | 10670 |
| 3002 | 119 | 14296 | 10774 |
| 2032 | 120 | 14566 | 10708 |
| 3026 | 125 | 14277 | 10694 |
| 3006 | 122 | 14275 | 10560 |
| 4389 | 122 | 14498 | 10294 |
| 3077 | 121 | 14390 | 10378 |
| 2994 | 120 | 14478 | 10154 |
| 3030 | 119 | 14276 | 10620 |
| 3003 | 119 | 14308 | 10507 |
| 3021 | 120 | 17762 | 10684 |
| 2999 | 121 | 14575 | 10460 |
| 3125 | 118 | 14379 | 10558 |

## Verwerking

Hieronder bevindt zich een grafiek met een gemiddelde opslagtijd van hetzelfde beeld bij de verschillende methoden.

## Conclusie

Uit de meting blijkt dat een 1 dimensionale array sowieso het snelst is, in release mode zelfs super snel.

## Evaluatie

Onze hypothese klopt. De hypothese was gemaakt op basis van kennis over assembly. Een 1 dimensionale array maakt minder geheugen aanvragen dan een 2 dimensionale array. Van een 2 dimensionale array bevat de eerste array alleen pointers naar de tweede array. In de tweede array zijn de pixels en informatie opgeslagen. Dus als een pixel verandert moet worden kost dat meer assembly instructies, dus in de praktijk meer tijd. Daarnaast kan er bij een 1 dimensionale array ook meer snelheidswinst geboekt worden met het kopiëren van data.