# 1. Performance meting van verschillende RGBImageStudent implementaties

## 1.1. Namen en datum

Peter Bonnema; 21-04-2016

## 1.2. Doel

Tijdens het experiment worden de performance van verschillende implementaties van de klasse RGBImageStudent gemeten. Dit zal later de doorslaggevende factor zijn in de keuze voor één van de implementatie. De verschillen tussen de implementaties is vooral in de manier van pixeldata opslag. De performance van de volgende opslag methoden zijn gemeten:

1. De default implementatie (die waarschijnlijk een 1-D array op de heap gebruikt)
2. 2-D column-major array op de heap
3. 2-D row-major array op de heap
4. 2-D contiguous row-major array op de heap waarbij de individuele rijen achter elkaar in het geheugen staan. De reden hiervoor is dat dit het mogelijk maakt om zowel op basis van x-y coördinaten pixels te benaderen zoals bij normale 2-D array's als op basis van een index pixels te benaderen zonder omzetting van één naar de ander. Het idee is dat het tijd scheelt als er voor een groot aantal pixels deze omzetting niet gedaan hoeft te worden.

## 1.3. Hypothese

Bij beide 2-D row-major array's zal het tijdsefficienter zijn om een geheel plaatje horizontaal te doorlopen (alle pixels rij voor rij te lezen of overschrijven). Bij de 2-D column-major array zal het tijdsefficienter zijn om dit koloms-gewijs te doen. Dit komt doordat bij het ophalen van data uit het geheugen ook direct de omliggende data (de rij of kolom waar de pixel in zit) in de cache wordt gezet. Dit heeft echter alleen zin als de volgende pixel die opgevraagd wordt in de zelfde rij of kolom zit. Anders is dit een cache-miss en duurt het dus langer voordat de volgende pixel beschikbaar is. Als je bij een 2-D row-major array pixels van boven naar beneden benaderd zal dit continue het geval zijn. Het zelfde geldt voor column-major arrays.

Bij implementaties 2 en 3 zal de benadering van pixels op basis van een index langzamer zijn ten opzichte van de default implementatie omdat de index in dat geval eerst moet worden opgerekend naar een x en een y. Dit verschil zal er niet zijn tussen implementaties 1 en 4 omdat de pixels bij de 4e op beide manieren te benaderen zijn zonder omzetting.

Er wordt ook verwacht dat de benadering van pixels op basis van x en y bij de contiguous array zelfs nog sneller zal zijn dan bij de default implementatie omdat ook hier geen omzetting gedaan hoeft te worden.

## 1.4. Werkwijze

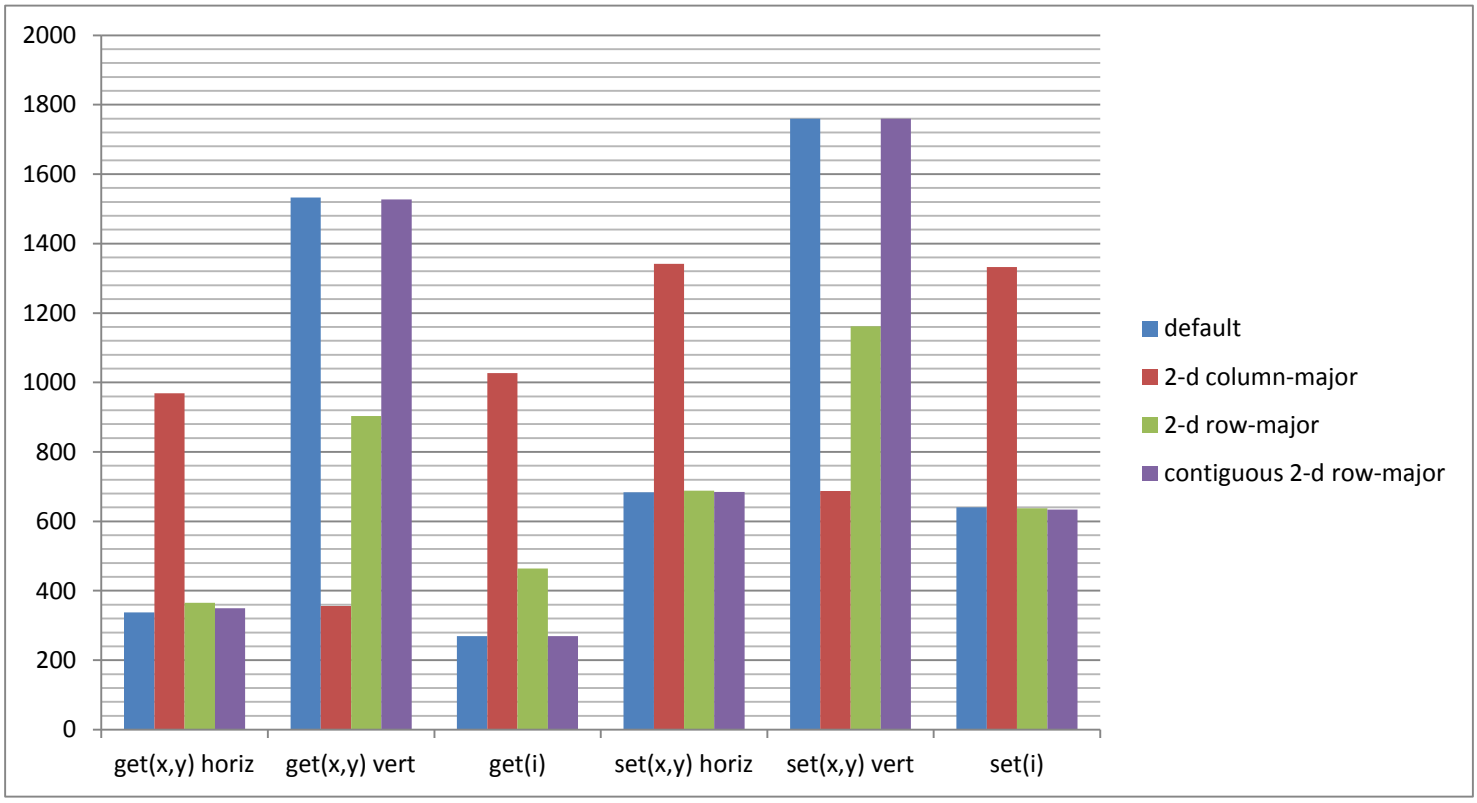
Er wordt een testplaatje genomen van een hoge resolutie zodat de verschillende caching mechanismen van de processor echt een rol gaan spelen. De gekozen resolutie is DCI 4K (4096x2160). De volgende bewerkingen worden 10 keer om de beurt uitgevoerd op het volledige plaatje met elk van de implementaties afzonderlijk en de tijd die het kost wordt gemeten met een timer API aangeleverd door de docent.

1. GetPixel(x, y) horizontaal: vraag alle pixels van links naar rechts om de beurt op.
2. GetPixel(x, y) verticaal: vraag alle pixels van boven naar benede om de beurt op.

3. GetPixel(i): vraag alle pixels op op basis van een index. De functie is gedefinieerd als dat i op loopt van links naar rechts. De pixel in de linker boven hoek is dus i=0 en de pixel rechts daarvan is i=1 en de pixel er onder is dus i=width.
4. SetPixel(x, y, nieuw_kleur) horizontaal: overschrijf alle pixels van links naar rechts om de beurt met nieuw_kleur.
5. SetPixel(x, y, nieuw_kleur) verticaal: overschrijf alle pixels van boven naar benede om de beurt met nieuwe_kleur.
6. SetPixel(i, nieuw_kleur): overschrijf alle pixels op op basis van een index.

## 1.5. Resultaten

Hieronder zijn de resultaten in een staafdiagram weergegeven. De verticale as geeft de gemeten tijd aan in milliseconden. De horizontale as geeft de verschillende bewerkingen weer hierboven zijn omschreven.




## 1.6. Verwerking

De gegevens hoeven niet verder verwerkt te worden. Ze kunnen in deze vorm direct worden geïnterpreteerd.

## 1.7. Conclusie

Wat direct opvalt is dat de 2-D column-major opslag methode in de meeste gevallen, 4 van de 6, het slechtst presteert. Alleen bij de verticale benadering van pixels is dit niet zo. Dit komt overeen met de hypthese.

Wat ook overeen komt met de hypthese is dat de row-major arrays slecht presteren het vertikaal doorlopen van het plaatje. Wat hier echter opvalt is dat dit effect op de default implementatie en de contiguous array vele male sterker is. Dit is niet voorspeld en er is hier voor alsnog geen verklaring voor gevonden.

De voorspelling dat de 2-D contiguous row-major array beter zou presteren bij GetPixel(i) dan de normale 2-D row-major array blijkt te kloppen. GetPixel(i) duurt bij de normale row-major array langer dan bij GetPixel(x, y) maar dit verschil is er bij de contiguous array niet. Daarin gedraagd deze zich dus als de default implementatie die een 1-D array gebruikt. Dit verschil is tussen de SetPixel() methoden echter niet te zien. Waarschijnlijk komt dit doordat het effect wordt overstemt door de vertraging die het overschrijven van pixels met zich mee brengt.

Er is geen verschil te zien tussen de 2-D contiguous array en de default implementatie bij GetPixel(x, y) horizontaal al was dit wel voorspeld. Dit voordeel van een 2-D contiguous row-major array ten opzichte van een 1-D implementatie valt dus weg. Dit valt ook te zien aan het feit dat de normale 2-D row-major array niet langzamer is dan de default implementatie. Hier is geen verklaring voor gevonden.

De keuze voor een implementatie valt nu op de normale 2-D row-major array omdat het negatieve effect van het vertikaal doorlopen van het plaatje bij de contiguous array zo veel groter is. Echter verandert de keuze als men van te voren weet dat het niet nodig zal zijn om plaatjes vertikaal te doorlopen. In dat geval is het voordeliger om voor de 2-D contiguous array of de default implementatie te kiezen. Hierbij moet genoemd worden dat de contiguous array en de default implementatie zich exact het zelfde gedragen. Er is op geen enkel punt een verschil gevonden.

## 1.8. Evaluatie

Het doel van dit experiment is om op basis van performance verschillen een implementatie of opslag methode te kiezen en dit is met de gepresenteerde gegevens mogelijk. Echter kan het ook zijn dat er met voorkennis over het gebruik van de Image klasse een betere keuze gemaakt kan worden. De voornemens over het gebruik kunnen hier natuurlijk op worden aangepast.

Elke test is 10 keer uitgevoerd om invloeden van het besturingssysteem te beperken. Daarnaast zijn de metingen verschillende malen uitgevoerd om de kans te minimalizeren dat de gegevens op welke manier dan ook exceptioneel zijn. Er is hierbij ook gelet op (niet-exceptionele) afwijkingen tussen individuele metingen maar deze bleken gering te zijn.